Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження методології DevOps для розробки та підтримки веб-застосунків

Виконав: студент VI курсу, групи СНнм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Мачужак А.В.
(підпис) (прізвище та ініціали)

Керівник _____ Готович В.А.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Мацюк О.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Стадник Н.Б.
(підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Мачужаку Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методології DevOps для розробки та підтримки веб-застосунків.

Керівник роботи Готович Володимир Анатолійович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 22 » листопада 2022 року № 4/7-950

2. Термін подання студентом завершеної роботи 23 травня 2023р.

3. Вихідні дані до роботи Наукові публікації про DevOps методології та інструменти, переваги CI/CD методології та Docker контейнерів, технічна документація, інтернет-джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз DevOps методологій та інструментів для автоматизації процесів при розробці веб-застосунків. 2. Обґрунтування вибору інструментів для моніторингу та автоматизації процесів при розробці веб-застосунку «Файловий менеджер». 3. Впровадження методів DevOps для автоматизації та моніторингу проекту «Файловий менеджер» та їхній аналіз. 4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Мацюк О.В., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 14 листопада 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	14.11.2022-15.11.2022	*Виконано*
2.	Підбір наукових джерел про DevOps методології	16.11.2022-27.11.2022	*Виконано*
	та інструменти, переваги CI/CD та Docker-контейнерів		
	в DevOps		
3.	Опрацювання наукових публікацій та збір даних по	28.11.2022-25.12.2022	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	09.01.2023-12.03.2023	*Виконано*
	роботи		
5.	Оформлення розділу «Аналіз DevOps методологій	13.03.2023-19.03.2023	*Виконано*
	та інструментів для автоматизації процесів при		
	розробці веб-застосунків»		
6.	Оформлення розділу «Обгрунтування вибору	20.03.2023-26.03.2023	*Виконано*
	інструментів для моніторингу та		
	автоматизації процесів при розробці веб-		
	застосунку «Файловий менеджер»»		
7.	Оформлення розділу «Впровадження методів DevOps	27.03.2023-02.04.2023	*Виконано*
	для автоматизації та моніторингу проекту «Файловий		
	менеджер»»		
8.	Виконання завдання до підрозділу «Охорона праці»	10.04.2023-16.04.2023	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	17.04.2023-23.04.2023	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	24.04.2023-30.04.2023	*Виконано*
11.	Нормоконтроль	01.05.2023-07.05.2023	*Виконано*
12.	Перевірка на плагіат	08.05.2023	*Виконано*
13.	Попередній захист кваліфікаційної роботи	15.05.2023	*Виконано*
14.	Захист кваліфікаційної роботи	23.05.2023	

Студент _____ Мачужак А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження методології DevOps для розробки та підтримки веб-застосунків // Кваліфікаційна робота освітнього рівня «Магістр» // Мачужак Андрій Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНнм-61 // Тернопіль, 2023 // С. 114, рис. – 31, табл. – 2, кресл. – , додат. – 3, бібліогр. – 59.

Ключові слова: методологія DevOps, CI/CD, GitHub Actions, Google Cloud Platform, контейнер Docker, віртуальна машина, метрика для веб-додатку.

Кваліфікаційна робота присвячена дослідженню та застосуванню практик та методологій DevOps для автоматизації процесів розробки і розгортання програмного забезпечення при створенні веб-додатків.

В першому розділі кваліфікаційної роботи наведено основні визначення та принципи DevOps. Описано різницю між DevOps та традиційними методологіями. Розглянуто основні практики та інструменти DevOps. Проаналізовано та досліджено переваги застосування CI/CD методології та Docker-контейнерів для пришвидшення та оптимізації процесів розробки та розгортання програмного забезпечення.

В другому розділі проведено аналіз інструментів, які застосовано в практичній частині роботи, зокрема, CI/CD, Docker та інструментів моніторингу середовища розгортання програмного продукту. Також розглянуто процеси реєстрації доменного імені та отримання SSL-сертифікатів, необхідних для розгортання програмного продукту.

В третьому розділі описано результати практичної реалізації методології CI/CD та впровадження моніторингу додатку в процесі розробки конкретного програмного продукту. Проаналізовано результати практичного застосування методологій DevOps.

Об’єкт дослідження: автоматизація процесів розробки, тестування та розгортання програмного забезпечення при створенні веб-додатків.

Предмет дослідження: застосування методів та інструментів автоматизації процесів розробки, тестування та розгортання веб-додатків на прикладі конкретного програмного рішення.

ANNOTATION

DevOps Methodology Research for Developing and Support of Web-Applications // Qualification work of the educational level "Master" // Andrii Machuzhak // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNnm-61 group // Ternopil, 2023 // P. 114, fig. – 31, tables – 2, chair. – , annexes – 3, references. – 59.

Key words: DevOps methodology, CI/CD, GitHub Actions, Google Cloud Platform, Docker container, virtual machine, metric for web application.

The qualification work is devoted to the research and application of DevOps practices and methodologies for automating the development and deployment of software during the creation of web applications.

The first section of the qualification work provides the basic definitions and principles of DevOps. The difference between DevOps and traditional methodologies is described. Basic DevOps practices and tools are covered. The benefits of using the CI/CD methodology and Docker containers to speed up and optimize software development and deployment processes have been analyzed and investigated.

The second section analyzes the tools used in the practical part of the work, in particular, CI/CD, Docker and tools for monitoring the software product deployment environment. The processes of registering a domain name and obtaining SSL certificates, which are required for the deployment of a software product, are also covered.

The third section describes the results of the practical implementation of the CI/CD methodology and the implementation of application monitoring in the process of developing a specific software product. The results of the practical application of DevOps methodologies are analyzed.

Study object: automation of software development, testing and deployment processes when creating web applications.

Study subject: application of methods and tools for automating the development, testing and deployment processes of web applications on the base of the example of a particular software solution.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Agile – підхід до управління проектами, який передбачає розбиття проекту на етапи та підкреслює постійну співпрацю та вдосконалення.

CD (англ. Continuous Deployment/Delivery) – метод автоматизації процесів, за якого команди розробляють програмне забезпечення за короткий проміжок часу, забезпечуючи надійні випуски в будь-який час.

Certbot – безкоштовний програмний інструмент з відкритим кодом для автоматичного використання сертифікатів Let's Encrypt на веб-сайтах, що використовують HTTPS.

CI (англ. Continuous Integration) – практика перевірки та тестування програмного коду.

Cloud Logging – повністю керована служба, яка працює в масштабі та може приймати дані журналів додатків і платформ, а також користувацькі дані журналів із середовищ платформи Google Kubernetes, віртуальних машин та інших служб всередині та за межами інфраструктури Google Cloud.

Cloud SQL – служба баз даних, керована та підтримувана постачальником, яка допомагає налаштовувати, підтримувати, керувати та адмініструвати реляційні бази даних на Google Cloud Platform.

Compute/Compute Engine – налаштовувана обчислювальна служба, яка дозволяє створювати та запускати віртуальні машини в інфраструктурі Google.

DevOps – набір практик, який поєднує розробку програмного забезпечення (Dev) та IT-операції (Ops).

Docker – інструментарій для створення та управління Linux-контейнерами, які є ізольованими та переносними.

Dockerfile – текстовий файл без розширення, який містить набір інструкцій для інструмента Docker.

GCP (англ. Google Cloud Provider) – набір служб хмарних обчислень, який працює на тій самій інфраструктурі, що й Google, внутрішньо відносно до своїх

кінцевих користувачів, як-от пошук Google, Gmail, Google Диск і YouTube.

GitHub – сервіс для зберігання розробницького коду у системі контролю версій Git, яку розробив Лінус Торвальдс.

GitHub Actions – сервіс в GitHub, який дозволяє автоматизувати робочі процеси розробки програмного забезпечення прямо у репозиторії.

IaC (англ. Infrastructure as a code) – ключова практика DevOps, яка передбачає управління інфраструктурою, такою як мережі, обчислювальні сервіси, бази даних, сховища та топологія підключення.

Kubernetes – інструмент для автоматичного розгортання, масштабування та керування додатками у вигляді контейнерів.

PR (англ. Pull request) – відкривання нової гілки в системі контролю версій із запитом на злиття до іншої гілки.

SDLC (англ. Software development lifecycle) – процес, який описує етапи, пов'язані з розробкою програмного продукту, від планування та аналізу до тестування та розгортання.

SSH (англ. Secure Shell) – захищений протокол, що дозволяє отримати віддалений доступ до серверів і пристроїв.

SSL (англ. Secure Sockets Layer) – протокол шифрування, який захищає дані, що передаються між сервером і браузером.

VM (англ. Virtual Machine) – віртуальна реалізація комп’ютерної системи, яка відтворює програмно такі ресурси як процесор, оперативна пам'ять, дисків та пристроїв вводу і виводу.

Waterfall – покроковий метод розробки програмних продуктів, в якому без попередніх кроків не буде виконаний наступний крок, а також він напряму залежить від результату попереднього етапу.

Баг (англ. bug) – непередбачена та непротестована помилка в програмному забезпеченні.

Деплой (англ. deploy) – процес розгортання та запуску веб-додатку або сайту в його робочому середовищі, тобто на сервері або хостингу.

Методологія (англ. methodology) – система методів і принципів, що використовуються в певній дисципліні або галузі навчання.

Моніторинг (англ. monitoring) – процес спостереження та перевірки прогресу або якості чогось протягом певного періоду часу.

ПЗ – програмне забезпечення.

Проксі (англ. proxy) – сервер, який виступає посередником між клієнтом та іншим сервером.

ЗМІСТ

ВСТУП

Актуальність теми. На сьогоднішній день ефективність процесів розробки та оновлення програмного забезпечення досягається за рахунок застосування теоретично і практично обгрунтованих методологій та інструментальних засобів, об’єднаних під загальною назвою DevOps. Від коректності вибору та використання конкретного набору цих методів і засобів при розробці та підтримці програмного продукту залежить загальна ефективність процесів розробки/розгортання та підтримки/оновлення версій програмного забезпечення. Тому актуальною є задача аналізу відомих методів та інструментів DevOps з метою максимального ефективного їх використання на практиці з врахуванням особливостей конкретного програмного рішення, яке розробляється і підтримується командами розробки та супроводу.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Магістр» є дослідження та застосування на практиці найпопулярніших методів сімейства DevOps та відповідних інструментів. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

1. Проаналізувати найпопулярніші на сьогоднішній день методології та інструментальні засоби DevOps, зокрема, такі як continuous integration та continuous delivery (CI/CD).
2. Проаналізувати основні методи тестування та моніторингу програмного забезпечення, що використовуються в складі DevOps.
3. Виконати порівняння існуючих DevOps-інструментів для автоматизації процесів розробки, тестування, розгортання та моніторингу програмного забезпечення.
4. Застосувати на практиці методи та інструменти DevOps при розробці веб-додатку «Файловий менеджер» та оцінити їх ефективність, переваги та недоліки.

Для успішного виконання цих завдань необхідними є розуміння технічних та методологічних аспектів DevOps а також навики практичного використання

основних на сьогодні методів та інструментів розробки програмного забезпечення.

Об’єкт дослідження: автоматизація процесів розробки, тестування та розгортання програмного забезпечення при створенні веб-додатків.

Предмет дослідження: застосування методів та інструментів автоматизації процесів розробки, тестування та розгортання веб-додатків на прикладі конкретного програмного рішення.

Наукова новизна одержаних результатів кваліфікаційної роботи полягає у тому, що набула подальшого розвитку методологія DevOps, зокрема з використанням принципу CI/CD, інструментів контейнеризації, підходу “інфраструктура як код”, моніторингу середовища розгортання програмного рішення.

Практичне значення одержаних результатів. Здійснено на практиці автоматизацію процесів розробки, тестування та розгортання веб-додатків на приклад конкретного веб-додатку «Файловий менеджер». При цьому проведено порівняльний аналіз використаних інструментів і методів, що дозволило виявити їх переваги та недоліки в конкретних випадках.

Апробація результатів магістерської роботи. Основні результати проведених досліджень та отриманих результатів обговорювались на X Науково-технічній конференції «Інформаційні моделі, системи та технології» (м. Тернопіль, 7-8 грудня 2022 р.) та на XI Міжнародній науково-технічній конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (м. Тернопіль, 7-8 грудня 2022 р.)

Публікації. Основні результати кваліфікаційної роботи опубліковано у двох працях конференцій (див. додатки А, Б).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку літератури з 59 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 131 сторінку, з них 114 сторінок основного тексту, який містить 31 рисунків та 2 таблиці.

1 АНАЛІЗ МЕТОДОЛОГІЙ DEVOPS ТА ІНСТРУМЕНТІВ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ РОЗРОБКИ, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Коротка історія DevOps. Основні означення, концепції, принципи

DevOps – це відносно новий термін, який з'явився в індустрії розробки програмного забезпечення (підмножині галузі комп’ютерних наук) за останнє десятиліття. Він поєднує практики розробки програмного забезпечення (Dev) та IT-операцій (Ops) для покращення співпраці та інтеграції між командами розробки та розгортання програмного продукту. Кінцевою метою DevOps є надання програмних продуктів замовникові більш швидко, надійно та якісно.

Витоки DevOps можна простежити до початку 2000-х років, коли Agile методологія набирала популярність [1]. Agile розробка базувалася на наборі цінностей та принципів, які підкреслювали співпрацю, гнучкість та постійне вдосконалення. Маніфест Agile стверджував, що робоче програмне забезпечення є основним мірилом прогресу, і що люди та взаємодії важливіші за процеси та інструменти.

У міру того, як формулювання принципів Agile набирало обертів, розробники почали розуміти, що їм потрібно тісніше співпрацювати з операційними командами, щоб забезпечити належне розгортання та обслуговування їх програмного забезпечення. Це призвело до появи концепції «DevOps», яка вперше була придумана Патріком Дебуа в 2009 році [2].

Дебуа організував першу конференцію DevOpsDays у 2009 році, яка зібрала розробників і адміністраторів для обговорення шляхів поліпшення співпраці та інтеграції між двома командами. З тих пір DevOps-рух швидко зростав, з конференціями, мітапами та спільнотами, що з'являються по всьому світу.

Практики DevOps зараз широко застосовуються в індустрії програмного забезпечення, і багато організацій визнають переваги інтеграції команд

розробників та операцій. DevOps також породив ряд пов'язаних практик, таких як безперервна інтеграція, безперервна доставка та інфраструктура як код [3].

Впровадження DevOps було обумовлено низкою факторів, включаючи зростаючий попит на швидшу розробку програмного забезпечення, зростання хмарних обчислень та необхідність більшої масштабованості та стійкості в сучасних програмних системах. DevOps також був прийнятий організаціями, які прагнуть покращити свої можливості доставки програмного забезпечення та підвищити ефективність своїх IT-операцій.

DevOps охоплює численний ряд практик та концепцій, пов'язаних з розробкою програмного забезпечення та IT-операціями. Ключові концепції DevOps наступні [4]:

- Співпраця між командами розробників та IT-операцій. DevOps підкреслює співпрацю та комунікацію між командами розробки програмного забезпечення та IT-операцій. Працюючи разом, команди можуть надавати програмні продукти швидше, надійніше та якісніше.

- Безперервна доставка (розгортання). DevOps зосереджується на автоматизації процесу розгортання програмного забезпечення для досягнення швидших та частіших релізів. Безперервна доставка передбачає використання інструментів і процесів для автоматизації тестування, розгортання і моніторингу програмних продуктів.

- Інфраструктура як код. DevOps заохочує використання коду для автоматизації створення та управління інфраструктурою. Розглядаючи інфраструктуру як код, розробники та операційні групи можуть працювати разом, щоб забезпечити узгодженість та відтворюваність інфраструктури.

- Agile та Lean-методології. DevOps часто асоціюється з методологіями розробки програмного забезпечення Agile та Lean, які підкреслюють співпрацю, гнучкість та постійне вдосконалення.

- Культура постійного вдосконалення. DevOps – це більше, ніж просто набір практик. DevOps-команди постійно шукають шляхи вдосконалення своїх процесів та надання програмних продуктів більш ефективно та результативно.

DevOps – це підхід до розробки та підтримки програмного забезпечення, який базується на тісній співпраці між командами розробників та системних адміністраторів. Метою DevOps є забезпечення швидкого, якісного та безперебійного випуску продукту в робоче середовище, а також його моніторинг та оптимізації. Для досягнення цих цілей DevOps методології орієнтуються на дотримання наступних принципів [5]:

- Створення культури співробітництва, довіри та взаємодопомоги між учасниками проекту. Команди повинні мати спільну візію, цінності та цілі, а також дотримуватися принципу "виправляй свої помилки сам".
- Максимальна автоматизація всіх процесів, пов'язаних з розробкою, тестуванням, збиранням, розгортанням та моніторингом продукту. Автоматизація дозволяє зменшити ризик людської помилки, скоротити час випуску продукту, покращити його якість та безпеку.
- Вимірювання успішності DevOps за допомогою конкретних показників, таких як частота релізів, час випуску продукту, час виявлення та вирішення інцидентів, коефіцієнт задоволеності користувачів тощо. Вимірювання дозволяє оцінити ефективність DevOps, виявити проблеми та можливості для покращення.
- Обмін знаннями та навичками між командами та індивідами. Обмін знаннями сприяє навчанню, інноваціям та покращенню якості продукту. Для обміну знаннями можна використовувати регулярні зустрічі, семінари, воркшопи, документацію, код-рев'ю тощо.

Дотримуючись цих принципів, DevOps допомагає створювати ефективну, гнучку та адаптивну систему розробки та підтримки програмного забезпечення.

Загалом, DevOps – це філософія та набір практик, які спрямовані на подолання розриву між розробкою програмного забезпечення та IT-операціями, з кінцевою метою надання високоякісних програмних продуктів більш швидше та надійніше.

Підсумовуючи можна зробити висновок, що розвиток DevOps більше схожий на подорож, а не на пункт призначення. DevOps постійно розвивається і все ще знаходиться в процесі еволюції. З цього моменту ці методології можуть

стати тільки кращими. На даний момент важливість автоматизації в DevOps величезна, і саме інструменти автоматизації підтримують високий рівень ефективності. Незалежно від того, до якої команди ви приєднуєтесь, базові знання DevOps повинні бути обов’язковими перед приєднанням до організації.

1.2 Різниця між DevOps і традиційними методологіями

До появи DevOps розробка програмного забезпечення здійснювалася з використанням різних методологій. Серед них найбільшого поширення набули методології Waterfall і Agile [6].

Традиційні методології при розробці програмного забезпечення, такі як Waterfall, V-model (Модель перевірки та верифікації), Інкрементальна та ітеративна модель, спіральна модель, RUP (англ. The Rational Unified Process) моделі базуються на структурному та прогнозованому підходах до розробки ПЗ [7]. Вони працюють з передбачуваними змінами, використовуючи попередньо визначені процеси та процедури для керування проектами. Кожен етап повинен бути завершений перед початком наступного. Такий підхід може призводити до затримок, помилок та невідповідностей між очікуваннями клієнтів та реальними результатами.

Коротко розглянемо Waterfall модель як представницю всіх традиційних методів в SDLC. Waterfall є однією з найстаріших методологій, які використовувалися. Вона дотримується послідовного підходу, тобто кожен етап залежить від інформації з попереднього етапу. Іншими словами, ця модель дотримується підходу зверху вниз.

Неспроможність змінюватися є основним недоліком моделі Waterfall. Ця модель нежиттєздатна, якщо проект потребує гнучкості або якщо проект довгостроковий і вимагає постійних змін.

Однак з ростом складності та обсягу проектів, з'явилися нові методології, такі як Agile, Scrum, Kanban та Lean [7], які прагнуть до більш гнучкого та ітеративного підходу до розробки ПЗ. Ці методології дозволяють командам

розробників пристосовуватися до змін, швидко реагувати на вимоги клієнта та покращувати якість продукту через постійну зворотну зв'язок зі замовником та використання гнучких процесів та методів управління проектами.

В якості узагальнення гнучких методологій розглянемо Agile, який сприяє постійній ітерації розробки та тестування по всьому SDLC. Проектування та тестування відбуваються одночасно, на відміну від моделі Waterfall. В Agile ми створюємо постійні оновлення, кожне з яких містить невеликі, поступові зміни попередньої версії. На кожній ітерації проект тестується. Це допомагає зрозуміти та визначити незначні проблеми на ранній стадії, щоб уникнути серйозних проблем у міру просування проекту.

Хоча Agile методологія привнесла гнучкість у розвиток, вона не змогла привнести гнучкість в операції [8]. Це було вирішено методологією DevOps. У моделі DevOps команди розробки та IT-операцій тісно співпрацюють як одна команда. DevOps допомагає нам спростити як процеси розробки, так і розгортання.

Автоматизація є ключем до DevOps. Ця методологія налаштовує безперервну інтеграцію, безперервну доставку та безперервне розгортання в циклі випуску. DevOps допомагає у співпраці в команді, зменшенні невдач/відкатів, а також надає можливості для постійного вдосконалення.

DevOps спрямований на підвищення швидкості, якості та надійності доставки програмних продуктів до клієнтів. DevOps використовує автоматизацію, моніторинг, неперервну інтеграцію та неперервну доставку для досягнення цих цілей.

Різниця між DevOps і традиційними методологіями полягає в тому, що DevOps сприяє більш гнучкому, ітеративному та співпрацюючому процесу розробки, де команди Dev і Ops працюють разом впродовж усього SDLC. DevOps також залучає до процесу інші зацікавлені сторони, такі як бізнес-аналітики, користувачі та стейкхолдери. DevOps дозволяє швидше адаптуватися до змін у вимогах, отримувати зворотний зв'язок в режимі реального часу та постійно поліпшувати як продукт, так і процес його розробки.

1.3 Обсяг знань та сфера компетенцій, які необхідні для DevOps-інженера

З появою такої професії як DevOps-інженер збільшилися вимоги до знань людини на цю посаду. Комбінуючи розробку програмного забезпечення та IT-операцій, обсяг інформації для засвоєння подвоївся, також вхідний поріг став набагато вищим, оскільки більшість DevOps-інженерів зробили перехід від розробників чи адміністраторів в цю сферу. Таким чином вони мали вже певну базу на якій розвивали подальші навички з іншої половини Dev/Ops.

Для того, щоб стати DevOps-інженером, необхідно мати базові знання з програмування та системного адміністрування [9]. Для цього можна зайнятися самостійним вивченням відповідних курсів та матеріалів, наприклад, курсів на платформах Udemy, Coursera, або Linux Academy. Також важливо знати принципи роботи інфраструктури, таких як хмарні сервіси, контейнеризація та оркестрація, які є ключовими компонентами DevOps практик. Для вивчення цих технологій можна використовувати платформи, такі як AWS, Azure або Google Cloud, або встановлювати і налаштовувати на власному комп'ютері відповідні програмні засоби, наприклад, Docker та Kubernetes.

Не менш важливо мати навички роботи з системами контролю версій, такими як Git, що є основою для роботи зі змінами в програмному забезпеченні. Також потрібно вивчити принципи тестування та автоматизації процесів, що допоможе забезпечити якість програмного забезпечення та ефективність розробки та доставки.

Згідно з популярним, підтримуваним спільнотою планом [9, 10] для розвитку у сфері DevOps необхідно розпочати з вивчення мови програмування (скриптової) для автоматизації завдань, далі опанувати базові концепції в ОС (операційні системи). Оскільки більшість серверів в мережі Інтернет базуються на лінукс дистрибутивах виникає необхідність також вивчити хоча б один з них, до прикладу, Ubuntu linux. Наступний список містить базові компоненти, необхідні фахівцям на шляху вивчення DevOps [10]:

- Bash Scripting.
- Vim/Nano/PowerShell/Emacs.
- Компіляція з вихідного коду (gcc, make та інші).
- Інструменти маніпулювання текстом.
- Моніторинг процесів.
- Продуктивність систем.
- Мережеві інструменти.
- Комп’ютерна мережа, безпека та протоколи.
- Інфраструктура як код.
- Забезпечення інфраструктури.
- Постачальники хмарних послуг.

Крім перерахованих, для вивчення DevOps також необхідно мати розуміння процесів автоматизації тестування, контролю версій коду, інструментів для розгортання та управління серверами та контейнерами, таких як Docker та Kubernetes, а також розуміння принципів безпеки мереж та систем. Крім того, для ефективного використання DevOps необхідно мати базові знання про Agile та Lean-методології розробки програмного забезпечення. Оскільки DevOps зазвичай є командною роботою, здатність до ефективної комунікації та співпраці з різними членами команди також є важливим фактором успіху в цій галузі.

1.4 Практики та інструменти DevOps

DevOps – це методологія розробки програмного забезпечення, яка підкреслює співпрацю та комунікацію між командами розробників та IT-операцій. Ця методологія спрямована на автоматизацію всього процесу доставки програмного забезпечення, від створення, тестування та розгортання до моніторингу та обслуговування. Існує кілька практик та інструментів, які дозволяють DevOps-командам досягати цих цілей, включаючи безперервну

інтеграцію/безперервне розгортання (CI/CD), моніторинг, контейнеризацію за допомогою таких інструментів, як Kubernetes, та хмарні обчислення.

1.4.1 Практики безперервної інтеграції та розгортання в DevOps

Continuous Integration/Continuous Deployment (CI/CD) – це набір практик, які забезпечують безперервну інтеграцію та розгортання змін коду у системах [11]. Безперервна інтеграція відноситься до процесу безперервної інтеграції змін коду в центральний репозиторій і автоматичного тестування цих змін, щоб переконатися, що вони не порушують існуючий код. Безперервне розгортання, з іншого боку, відноситься до процесу автоматичного розгортання змін коду в системах робочого середовища після успішного тестування.

CD/CD є важливою практикою в DevOps з кількох причин:

- Швидший час виходу на ринок. CD/CD дозволяє організаціям швидше та частіше випускати нові функції та оновлення для клієнтів, що допомагає їм випереджати конкурентів.
- Краща якість. Постійно тестуючи зміни коду, CD/CD допомагає гарантувати, що нові функції та оновлення мають високу якість і не вносять нових помилок.
- Підвищення ефективності. Автоматизація процесу доставки програмного забезпечення економить час і знижує ризик людської помилки, що призводить до підвищення ефективності.
- Покращена співпраця. CD/CD заохочує співпрацю між командами розробників та операцій, що допомагає побудувати культуру спільної відповідальності та власності.

Реалізація CD/CD в середовищі DevOps передбачає використання засобів автоматизації, які забезпечують безперервну інтеграцію, тестування та розгортання змін коду, ці кроки поєднуються в так званий конвеєр, в якому є етапи тестування та деплою, типовий конвеєр зображено на рисунку 1.1. Наведемо необхідні кроки, які пов'язані з впровадженням CD/CD в проект:

Контроль версій. Першим кроком є забезпечення відстеження та управління змінами коду за допомогою системи контролю версій, такої як Git.

Безперервна інтеграція. розробники перевіряють зміни коду в центральному сховищі, що запускає автоматизований процес збірки та тестування. Це гарантує, що зміни коду постійно інтегруються та тестуються.

Автоматизоване тестування. Автоматизоване тестування є важливою частиною CD/CD. Автоматизовані тести гарантують, що зміни коду ретельно перевіряються на функціональність, продуктивність і безпеку.

Безперервне розгортання. Після успішного тестування зміни коду автоматично розгортаються у виробничих системах. Це гарантує, що найновіші функції та оновлення будуть доступні клієнтам якомога швидше.

Моніторинг. Безперервний моніторинг має важливе значення для забезпечення того, щоб розгорнуті зміни коду функціонували належним чином. Моніторинг допомагає виявити та усунути будь-які проблеми, які можуть виникнути.

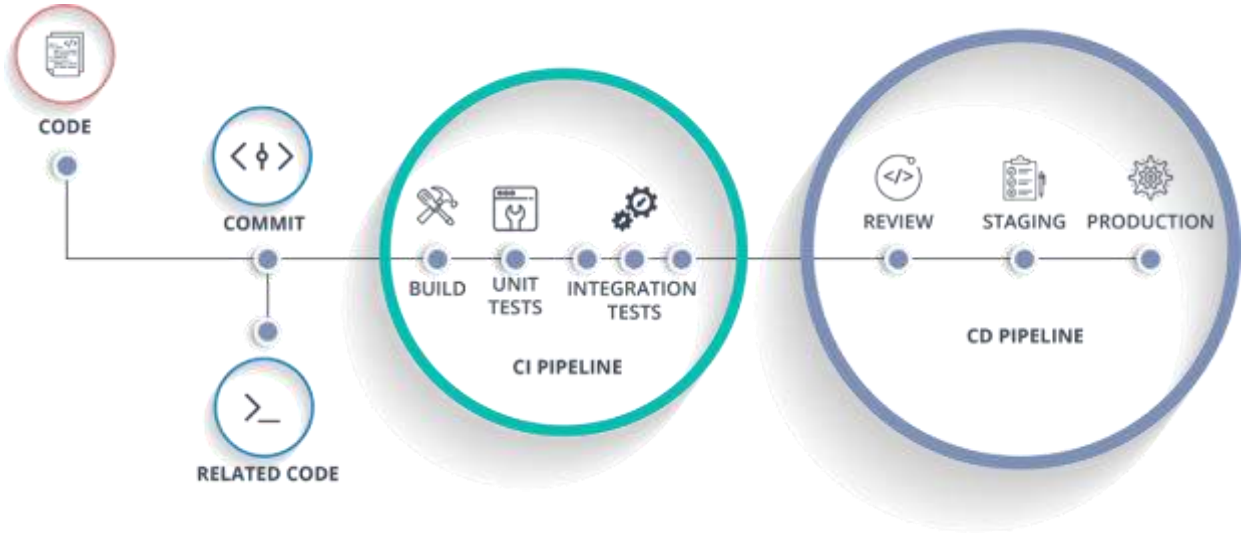

Рисунок 1.1 – Конвеєри CI/CD

Деякі з найпопулярніших і потужних інструментів CI/CD, які можна інтегрувати з системами контролю версій – це Jenkins, Travis CI, CircleCI та GitLab CI/CD [12]. Кожен з них має свої переваги і недоліки, а також різні варіанти настройки і автоматизації.

GitHub Actions – це вбудований інструмент CI/CD, який працює разом із вашим кодом на GitHub [13]. Він дозволяє створювати та запускати автоматизовані робочі процеси для будь-якої події веб-хука на GitHub, включаючи сторонні події веб-хука. Є можливість використовувати понад велику кількість попередньо написаних і протестованих робочих процесів CI/CD і попередньо вбудовані автоматизації в GitHub Marketplace, а також писати власні робочі процеси (або налаштовувати існуючі) у простих у використанні файлах YAML.

Jenkins є одним з найвідоміших і гнучких інструментів CI/CD, який може бути розгорнутий на будь-якій платформі і підтримує безліч плагінів для інтеграції з різними сервісами [14]. Він дозволяє писати власні плагіни для виконання будь-якого завдання. Jenkins можна інтегрувати з GitHub за допомогою веб-хука або спеціального плагіна GitHub.

Travis CI – це хмарний інструмент CI/CD, який тісно інтегрований з GitHub і надає безкоштовний сервіс для відкритих репозиторіїв. Він використовує .travis.yml файл для визначення робочих процесів і підтримує різні мови програмування і фреймворки [15]. Travis CI також підтримує розгортання на різних хмарних платформах.

CircleCI – ще один хмарний інструмент CI/CD, який також підтримує розгортання на ваших власних серверах [15]. Він має простий та інтуїтивно зрозумілий інтерфейс і використовує файл .circleci/config.yml для визначення робочих процесів. CircleCI також підтримує паралельне виконання завдань і кешування залежностей.

GitLab CI/CD є частиною платформи GitLab, яка забезпечує повний стек DevOps для управління SDLC [16]. Він використовує .gitlab-ci.yml для визначення робочих процесів та підтримує контейнеризацію та оркестровку за допомогою Docker та Kubernetes. GitLab CI/CD також підтримує моніторинг, безпеку та аналітику.

Отже, безперервна інтеграція/безперервне розгортання (CI/CD) є важливою практикою в DevOps, яка дозволяє організаціям надавати програмне

забезпечення швидше, з кращою якістю та підвищеною ефективністю. Реалізація CD/CD передбачає використання засобів автоматизації, які забезпечують безперервну інтеграцію, тестування та розгортання змін коду. Дотримуючись практики CD/CD, організації можуть покращити співпрацю між командами розробників та операційними командами, що призводить до більш ефективного та результативного процесу доставки програмного забезпечення.

1.4.2 Моніторинг інфраструктури середовища розгортання та продуктивності додатків

Моніторинг – це процес збору та аналізу даних про продуктивність, доступність і справність системи або програми [17, 18]. Існують різні типи моніторингу, які служать різним цілям і дають різну інформацію, виділимо два види моніторингу – моніторинг інфраструктури та моніторинг продуктивності додатків.

Моніторинг інфраструктури фокусується на фізичних і віртуальних компонентах системи, таких як сервери, мережі, сховища, контейнери і т.д. Це допомагає виявляти та усувати проблеми, пов'язані з обладнанням, програмним забезпеченням, конфігурацією або ємністю [19]. Моніторинг інфраструктури також може допомогти оптимізувати використання ресурсів і спланувати майбутні потреби.

Моніторинг продуктивності додатків (APM) фокусується на поведінці та продуктивності програми та її компонентів, таких як транзакції, запити, помилки, залежності тощо [20]. Це допомагає виміряти та покращити користувальницький досвід, якість та надійність програми. APM також може допомогти виявити та усунути вузькі місця, помилки або аномалії в коді програми або архітектурі.

Моніторинг є важливим аспектом DevOps практик, оскільки він дозволяє командам вимірювати продуктивність, доступність та надійність своїх додатків та інфраструктури [21]. Моніторинг також допомагає командам виявляти та

усувати проблеми, оптимізувати ресурси та покращувати взаємодію з користувачами. Далі розглянемо чому моніторинг важливий у DevOps-середовищі та як він може принести користь як розробникам, так і адміністраторам.

Однією з головних переваг моніторингу в DevOps є те, що він сприяє постійному зворотному зв'язку та вдосконаленню [22]. Збираючи та аналізуючи дані з різних джерел, таких як журнали, показники, трасування та сповіщення, команди можуть отримати уявлення про те, як їхній код поводиться у робочому середовищі, як він впливає на інші компоненти та як він відповідає бізнес-цілям. Моніторинг також дозволяє командам швидко виявляти та усувати помилки, перш ніж вони вплинуть на користувачів або спричинять простої. Таким чином, команди можуть гарантувати, що їхні програми завжди приносять користь і відповідають стандартам якості.

Ще однією перевагою моніторингу в DevOps є те, що він підтримує співпрацю та комунікацію між розробниками та адміністраторами [23]. Моніторинг забезпечує спільну мову та спільне уявлення про стан та продуктивність системи, що може допомогти командам узгодити свої цілі та очікування. Моніторинг також сприяє розвитку культури підзвітності та прозорості, оскільки команди можуть легко відстежувати та звітувати про свій прогрес та досягнення. Крім того, моніторинг може допомогти командам автоматизувати завдання та робочі процеси, такі як тестування, розгортання, масштабування та відновлення, що може зменшити людські помилки та підвищити ефективність.

Третя перевага моніторингу в DevOps полягає в тому, що він дозволяє командам оптимізувати свої ресурси та витрати [23]. Моніторинг може допомогти командам зрозуміти, як їхні програми споживають ресурси, такі як процесор, пам'ять, дисковий простір і пропускна здатність мережі, і як вони реагують на зміни попиту та навантаження. Моніторинг також може допомогти командам виявити та усунути вузькі місця, надмірності та витрати у своїй

інфраструктурі та процесах. Оптимізуючи свої ресурси та витрати, команди можуть підвищити свою операційну ефективність та прибутковість.

Четверта перевага моніторингу в DevOps полягає в тому, що він покращує користувальницький досвід і задоволеність [23]. Моніторинг може допомогти групам вимірювати та покращувати ключові показники ефективності (KPI), такі як час відгуку, доступність, пропускна здатність, рівень помилок і утримання користувачів. Моніторинг також може допомогти групам зрозуміти поведінку та вподобання користувачів, наприклад, які функції вони використовують найчастіше, з якими проблемами стикаються та який зворотний зв'язок надають. Покращуючи користувацький досвід і задоволеність, команди можуть підвищити лояльність і утримання клієнтів.

П'ята перевага моніторингу в DevOps полягає в тому, що він готує команди до майбутніх викликів і можливостей [23]. Моніторинг може допомогти командам передбачити та адаптуватися до змін на ринку, технологій або потреб клієнтів. Моніторинг також може допомогти командам впроваджувати інновації та експериментувати з новими ідеями та рішеннями, не ризикуючи стабільністю або безпекою своїх програм. Готуючись до майбутніх викликів і можливостей, команди можуть випереджати конкурентів і швидше досягати цінності.

1.4.3 Інструменти Docker та Kubernetes

Docker – це платформа контейнеризації з відкритим вихідним кодом, яка надає можливості розробникам розробляти, розгортати та керувати додатками всередині контейнерів [24]. Контейнери – це легкі, автономні та портативні виконувані пакети, які містять усі необхідні програмні компоненти, бібліотеки та файли конфігурації, необхідні для запуску програми.

Однією з основних переваг Docker є те, що він забезпечує ізольоване та однакове середовище для запуску додатків на різних платформах та середовищах [24]. Docker використовує багаторівневу архітектуру для побудови

та управління контейнерами, що дозволяє розробникам легко обмінюватися та розповсюджувати образи додатків у різних системах.

Ключові поняття і компоненти Docker включають в себе [25]:

1. Контейнери. Контейнер – це запущений екземпляр образу. Він забезпечує ізольоване та автономне середовище для запуску програми разом із її залежностями.

2. Образи – це шаблон лише для читання, який містить усі інструкції та залежності, необхідні для створення контейнера. Образи Docker можуть бути створені з нуля або побудовані поверх існуючих образів.

3. Dockerfile – це спеціальний файл з інструкціями, в якому онаписані кроки створення Docker-образу. Він визначає базовий образ, команди для встановлення залежностей та інші параметри конфігурації.

4. Docker Hub – це публічний реєстр для зберігання та обміну Docker-образами. Це дозволяє розробникам легко знаходити та завантажувати образи для популярних програм та сервісів.

5. Docker Compose – це інструмент, який виконує роль оркестрації програм Docker з багатьма контейнерами. Він використовує файл YAML для визначення служб, мереж та сховищ, необхідних для програми.

В цілому, Docker надає потужну платформу для створення і управління контейнеризованими додатками. Це дозволяє розробникам легко створювати стабільні та масштабовані середовища для запуску своїх програм, а також полегшує обмін та розповсюдження образів програм у різних системах.

Kubernetes – платформа для управління контейнеризованими додатками та сервісами, яка розповсюджується з відкритим вихідним кодом та спрощує як декларативну конфігурацію, так і автоматизацію [26]. Kubernetes та його послуги з інструментами є широко доступними.

Назва Kubernetes (з грецької) – означає рульовий або пілот. K8s як абревіатура є результатом підрахунку восьми букв між "K" і "s" [27]. Вихідний код Kubernetes Google відкрив у 2014 році. Kubernetes поєднує в собі понад 15

років досвіду Google у масштабному управлінні виробничими навантаженнями з найкращими у своєму роді ідеями та практиками спільноти.

Дізнатися чим Kubernetes корисний, можна повернувшись у минуле та розглянувши попередні практики розгортання ПЗ [27], що також зображено на рисунку 1.2.

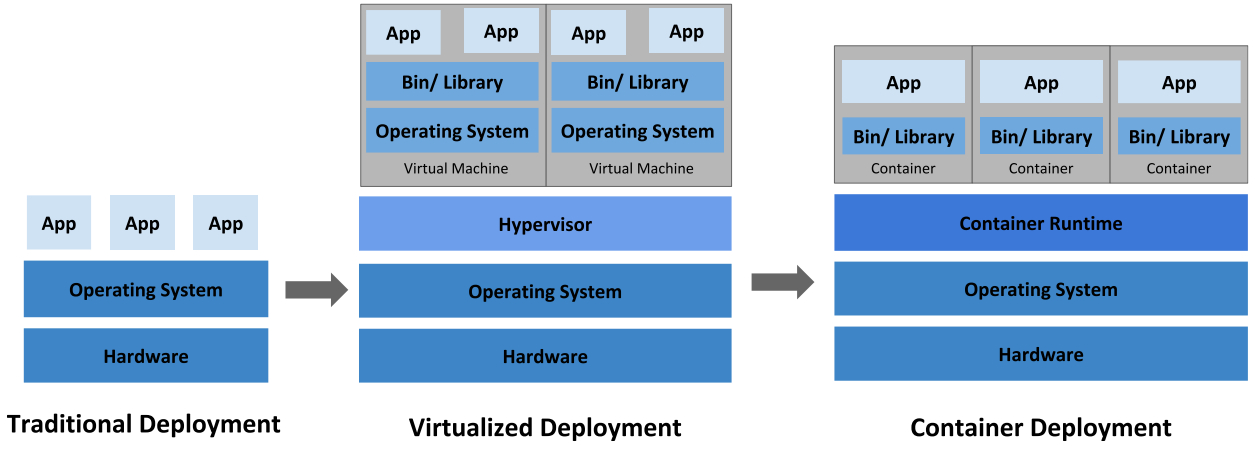

Рисунок 1.2 – Типи розгортання додатків за видом середовища

Традиційне розгортання. Раніше компанії використовували фізичні сервери для запуску програм. Проблемою такого підходу була неможливість ефективно розподілити ресурси між програмами на ньому. Наприклад, якщо декілька програм працюють на одному сервері, одна з них може займати надмірну кількість ресурсів, що призводить до недостатньої продуктивності інших програм. Розумним рішенням стало запускати кожну програму на окремому фізичному сервері. Проте, такий підхід був немасштабованим і коштував багато організаціям, оскільки ресурси були недостатньо використані, а підтримка багатьох фізичних серверів була витратною.

Віртуалізованане розгортання. Шляхом впровадження віртуалізації, було знайдено рішення, що дозволяє запускати декілька віртуальних машин на одному процесорі фізичного сервера. Цей підхід забезпечує ізоляцію додатків між віртуальними машинами та підвищує безпеку, оскільки інформація, що належить одній програмі, не може бути доступна іншим програмам.

Віртуалізація дозволяє краще використовувати фізичні ресурси сервера та забезпечує кращу масштабованість, оскільки програми можна легко додавати або оновлювати, витрати на апаратне забезпечення зменшуються тощо. За допомогою віртуалізації ви можете представити набір фізичних ресурсів як одноразовий кластер віртуальних машин.

Кожна віртуальна машина представляє собою повноцінну систему, яка має свою операційну систему та взаємодіє з усіма компонентами через віртуалізоване обладнання.

Контейнерне розгортання. Контейнери та віртуальні машини мають подібність, але володіють м’якшими властивостями ізолювання при спільному використанні ресурсів операційної системи між програмами. Тому вони рахуються легкими. Подібно до віртуальної машини, контейнер має свою окрему файлову систему, частину процесора, пам'яті, простору тощо. Оскільки вони відокремлені від базової інфраструктури, вони портативні через хмари та дистрибутиви ОС.

Контейнери надають засоби щоб об'єднувати та запускати програми. У виробничому середовищі важливо мати здатність ефективно керувати контейнерами, які виконують програми, і гарантувати безперервну роботу системи без простоїв. Наприклад, при падінні контейнера, слід запустити новий контейнер.

Kubernetes є потужним фреймворком, який забезпечує стійкий запуск розподілених систем. Він забезпечує масштабованість та надійність програм, надаючи широкі можливості для керування та контролю. Завдяки Kubernetes ви можете легко керувати розгортанням тестових релізів.

Kubernetes пропонує:

- Виявлення служб і балансування навантаження. Kubernetes надає можливість експонувати контейнери за допомогою імен DNS або із застосуванням IP-адреси. При великому трафіку на контейнер, Kubernetes може балансувати і розподілити мережевий трафік таким чином, щоб навантаження було стабільним.

• Оркестрування сховища. Kubernetes монтує систему зберігання на ваш вибір, таку як локальні сховища, постачальники загальнодоступних хмар, тощо.

• За допомогою Kubernetes ви можете визначити бажаний стан для розгорнутих контейнерів і система буде забезпечувати зміну фактичного стану контейнерів до потрібного стану. Наприклад, ви можете налаштувати автоматичне розгортання нових контейнерів, видалення існуючих контейнерів та перенесення всіх їхніх ресурсів на нові контейнери.

• Автоматична упаковка контейнерів. Kubernetes використовується на декількох машинах – вузлах, на яких забезпечується виконання контейнеризованих завдань. Ви повідомляєте Kubernetes, скільки ресурсів процесора та оперативної пам'яті (ОЗУ) буде доступно для використання кожному контейнеру. Kubernetes може встановити контейнери на ваших вузлах, щоб найкращим чином використовувати ваші ресурси.

• Самовідновлювальний. Kubernetes здійснює перезапуск контейнерів, що вийшли з ладу, замінює неробочі контейнери, знищує контейнери, що не відповідають встановленим користувачем перевіркам, і приховує їх від клієнтів до моменту, готовності обслуговування.

• Управління конфіденційними даними та конфігурацією. Kubernetes дозволяє зберігати та управляти приватною інформацією, такою як паролі та API ключі . Є можливість оновлювати змінні та конфігурацію програми, не перебудовуючи контейнери та не розкриваючи конфіденційну інформацію у конфігураційному стеку.

Kubernetes не є типовою "все в одному" платформою як послуга (PaaS). Оскільки Kubernetes працює на контейнерному рівні, а не на рівні апаратного забезпечення, він надає загальновизнані функції, які зазвичай притаманні PaaS, такі як розгортання, масштабування та балансування навантаження. Крім того, він дозволяє користувачам інтегрувати свої рішення для реєстрації, моніторингу та сповіщень. Проте Kubernetes не є монолітною системою, і ці функції є додатковими та можуть бути підключені за потреби. Kubernetes надає

можливість для створення платформи розробників, але зберігає свободу вибору та гнучкість, коли це необхідно.

1.4.4 Застосування хмарних технологій в методологіях DevOps

Хмарні обчислення стали частиною сучасного цифрового світу, і популярність швидко зростає в останні роки. Далі розглянемо типи хмарних технологій, висвітлимо їх переваги, проблеми, які вони представляють, дослідимо безпеку та відповідність вимогам у хмарних обчисленнях, а також надамо найкращі практики для хмарних обчислень.

Хмарні обчислення – це модель обчислень, яка забезпечує доступ на вимогу із загального набору ресурсів, таких як сервери, сховища, програми та послуги [28]. Доступ до ресурсів здійснюється через Інтернет, що дозволяє користувачам отримувати доступ до них з будь-якого місця, в будь-який час і на будь-якому пристрої. Хмарні технології надають можливість організаціям швидко збільшувати або зменшувати свої обчислювальні ресурси за потреби, зменшуючи капітальні витрати та збільшуючи гнучкість.

Хмарні обчислення класифікуються такими типами: публічні хмарні рішення, приватні хмари та гібридні хмарні середовища. [29]. Загальнодоступні хмарні сервіси пропонуються сторонніми постачальниками, такими як Amazon Web Services, Microsoft Azure або Google Cloud Platform. Приватні хмари створюються та підтримуються організацією для їх ексклюзивного використання. Гібридна хмара – це комбінація як загальнодоступних, так і приватних хмар, що дозволяє організаціям використовувати переваги обох.

Найбільшими хмарними провайдерами є Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP) [30]. Ці компанії домінують на ринку хмарних обчислень і пропонують широкий спектр послуг, включаючи обчислення, зберігання, мережі, безпеку, бази даних, аналітику тощо.

AWS є найбільшим постачальником хмарних послуг з часткою ринку близько 33% [30]. Він пропонує понад 200 хмарних сервісів і має міцну

репутацію завдяки своїй надійності, масштабованості та безпеці. AWS використовується мільйонами клієнтів по всьому світу, включаючи стартапи, підприємства, уряди та некомерційні організації.

Microsoft Azure є другим за величиною постачальником хмарних послуг з часткою ринку близько 18% [30]. Він пропонує широкий спектр послуг, включаючи віртуальні машини, бази даних, сховища, мережі, штучний інтелект тощо. Azure відома своїми гібридними хмарними можливостями, які дозволяють клієнтам запускати свої програми як у хмарі, так і локально.

Google Cloud Platform (GCP) [30] володіє третьою найбільшою часткою ринку серед постачальників хмарних послуг – близько 10%. Він пропонує широкий спектр послуг, включаючи обчислення, зберігання, мережу, аналіз даних, машинне навчання тощо. GCP відома своїм досвідом у сфері великих даних та аналітики і використовується багатьма клієнтами в науковому, дослідницькому та фінансовому секторах.

Інші відомі хмарні провайдери включають IBM Cloud, Oracle Cloud, Alibaba Cloud і DigitalOcean. Ці компанії пропонують різноманітні хмарні сервіси та орієнтуються на різні ринки та сегменти клієнтів.

Вибір правильного постачальника хмарних послуг може бути складним рішенням, яке залежить від багатьох факторів, таких як тип програми, необхідний рівень безпеки та відповідності, бюджет та технічні знання команди. Важливо ретельно оцінити функції, ціни та продуктивність кожного постачальника та вибрати такий що задовольняє потреби бізнесу.

Хмарні обчислення пропонують організаціям кілька переваг, зниження витрат, масштабованість і гнучкість. Використовуючи хмарні сервіси, підприємства можуть скоротити капітальні витрати, усунувши необхідність мануального обслуговування обладнання та ПЗ. Хмарні технології дозволяють організаціям швидко масштабувати свої ресурси вгору або вниз, забезпечуючи швидкість, необхідну для реагування на мінливі ринкові умови.

Незважаючи на переваги хмарних обчислень, є також кілька проблем, з якими стикаються організації. Однією з основних проблем є складність міграції

в хмару. Підприємства повинні ретельно планувати свою міграційну стратегію, щоб забезпечити плавний перехід, який може зайняти багато часу та ресурсів. Крім того, хмарні обчислення створюють проблеми щодо конфіденційності, безпеки та відповідності даних.

Безпека є серйозною проблемою в хмарних обчисленнях, оскільки дані зберігаються та доступні віддалено. Хмарні провайдери використовують різні заходи безпеки для захисту своєї інфраструктури, такі як шифрування, брандмауери та багатофакторна аутентифікація. Однак організація несе відповідальність за забезпечення безпеки своїх даних у хмарі. Найкращі методи хмарної безпеки включають використання надійних паролів, обмеження доступу до конфіденційних даних, а також регулярний моніторинг і аудит хмарних ресурсів.

Відповідність вимогам є ще однією критичною проблемою для компаній, які використовують хмарні обчислення. Організації повинні забезпечити дотримання різних норм, таких як GDPR, HIPAA та PCI-DSS, під час використання хмарних сервісів [31]. Постачальники хмарних послуг пропонують сертифікати відповідності вимогам, наприклад SOC 2, ISO 27001 і FedRAMP, які демонструють, що їхня інфраструктура відповідає певним вимогам безпеки та відповідності. Підприємства також повинні розробити власну політику та процедури відповідності вимогам, щоб забезпечити виконання своїх регуляторних зобов'язань.

Щоб максимізувати переваги хмарних обчислень при мінімізації їх проблем, організації повинні дотримуватися найкращих практик. До них відносяться розробка хмарної стратегії, вибір правильного постачальника хмарних послуг, ретельне планування міграції, впровадження надійних заходів безпеки, а також регулярний моніторинг та оптимізація хмарних ресурсів.

Постачальники хмарних обчислень – це компанії, які пропонують своїм клієнтам хмарні послуги. Ці провайдери керують величезними центрами обробки даних з потужними серверами, системами зберігання даних та

мережевим обладнанням, яке клієнти можуть використовувати для розміщення своїх додатків, зберігання даних та ведення бізнесу.

Хмарні обчислення зробили революцію в тому, як працюють підприємства, забезпечивши масштабований, гнучкий і економічно ефективний спосіб управління своїми обчислювальними ресурсами. Однак, щоб скористатися перевагами хмарних обчислень, організації повинні ретельно планувати свою хмарну стратегію, впроваджувати надійні заходи безпеки та забезпечувати дотримання відповідних правил. Роблячи це, підприємства можуть розкрити весь потенціал хмарних обчислень і отримати конкурентну перевагу в сучасному цифровому ландшафті.

Впровадження хмарних обчислень значно вплинуло на практики та методології DevOps [32]. Хмарні обчислення дозволили організаціям швидко масштабувати свою інфраструктуру вгору або вниз залежно від попиту, що призвело до збільшення потреби в автоматизації та безперервній доставці. Це добре узгоджується з філософією DevOps автоматизації конвеєра доставки програмного забезпечення, від інтеграції коду до розгортання у робоче середовище, з метою підвищення швидкості та надійності доставки програмного забезпечення.

Хмарні обчислення також дозволили командам DevOps використовувати нові інструменти та технології для своєї роботи. Хмарні провайдери пропонують широкий спектр послуг, таких як управління базами даних, оркестрування контейнерів та безсерверні обчислення, які можна легко інтегрувати в конвеєри DevOps. Це дозволяє DevOps-командам зосередитися на досягненні цінності бізнесу, а не на управлінні інфраструктурою. Крім того, хмарні провайдери часто пропонують специфічні для DevOps функції, такі як інструменти безперервної інтеграції та розгортання, які спрощують процес доставки програмного забезпечення. Як результат, команди DevOps можуть скористатися цими хмарними інструментами та сервісами для автоматизації своїх робочих процесів та підвищення загальної ефективності.

1.5 Переваги застосування методологій DevOps

Велика конкуренція на ринку вимагає здатності компанії швидко постачати послуги та програми. Щоб бути ефективними, управлінські процедури та інструменти потребують швидкої та надійної моделі. Через це ми повинні автоматизувати DevOps-процеси з використанням хмарних і нехмарних технологій автоматизації DevOps при розробці хмарних додатків.

DevOps методології та практики розглядатимемо у контексті хмарних технологій, згідно з дослідженням [32] вони вважаються нерозривними та доповнюють одне одного. У ньому автор досліджує DevOps та його застосування у хмарній розробці та тестуванні. У ній обговорюється, як перенести DevOps у хмару та покращити розробку програмного забезпечення та оперативність роботи, також розглядаються шляхи поширення таких DevOps-процесів та автоматизації на публічні та/або приватні хмари. Кінцева мета полягає в тому, щоб зрозуміти, як DevOps і хмара працюють разом, щоб допомогти організаціям у трансформації. У дослідженні робиться висновок, що DevOps і хмара йдуть пліч-о-пліч, і переваги використання DevOps з хмарними додатками стають все більш очевидними.

1.5.1 Переваги практик CI/CD

Найбільш вагомим аспектом DevOps є гнучкість, яку він забезпечує, коли справа доходить до автоматизації та доставки додатків та програмних систем. Раніше розробники використовували нетрадиційні методи розробки, але з впровадженням DevOps ситуація кардинально змінилася [33]. Основна мета полягає в тому, щоб дозволити розробникам задовольнити вимоги компанії, а також усунути затримку, яка властива розробці протягом багатьох років

Деякі переваги DevOps для організацій та команд включають [34]:

- Зменшення часу виведення продукту на ринок завдяки більш ефективному циклу розробки-тестування-запуску.

- Поліпшення якості продукту завдяки ранньому виявленню та усуненню помилок, а також застосуванню кращих практик написання коду та безпеки.
- Збільшення задоволеності клієнтів та користувачів завдяки швидкому впровадженню нових функцій та виправленню проблем.
- Посилення співпраці та комунікації між розробниками, тестувальниками, адміністраторами систем та іншими зацікавленими сторонами.
- Зниження витрат на інфраструктуру та експлуатацію завдяки оптимізації ресурсів, масштабуванню за запитом та використанню хмарних сервісів.
- Згідно з опитуванням DevOps Trends 2020 року, майже всі (99%) респондентів заявили, що DevOps позитивно вплинув на їхню організацію. Команди, які практикують DevOps, краще працюють, швидше працюють, спрощують реагування на інциденти та покращують співпрацю та комунікацію між командами [35, 36, 37].

Інші статистичні дані про переваги DevOps:

- Традиційний Ops в цілому займає на 41% більше часу.
- Якщо ви використовуєте традиційні операції, ви витратите на 21% більше часу на усунення помилок.
- В середньому, традиційні методи витрачають в середньому більше 7 годин на тиждень на спілкування.
- З DevOps на 60% менше часу витрачається на обробку випадків підтримки.
- З DevOps на покращення інфраструктури витрачається на 33% більше часу.

В реальному прикладі для компанії Tix було реалізовано CI/CD конвеєр та використано AWS DevOps інструменти і сервіси для автоматизації розгортання нових версій ПЗ та виправлення багів, а також для менеджменту інфраструктури [38].

Переваги включають в себе копіювання всієї архітектури і її запуск в найкоротші терміни за допомогою CloudFormation, повне усунення ризику

помилок ручної схильності і прискорення циклу розробки. За допомогою повністю автоматизованого рішення CI/CD, було досягнуто скорочення часу, витраченого на ручне розгортання і усунення помилок на 30%, завершення всього процесу за 10 хвилин замість години і дня для ручних реалізацій, і зроблено процес розробки та розгортання ефективним за допомогою сценарію CloudFormation (інфраструктура як код), що забезпечує швидший вихід на ринок.

1.5.2 Переваги застосування контейнерів

Контейнери мають малий розмір, оскільки їхні образи вимірюються в мегабайтах, а не в гігабайтах, якщо порівняти з VM. Контейнери вимагають менше ресурсів для розгортання, запуску та керування. Контейнери запускаються за секунди. Оскільки їхній розмір менший, одна система може містити набагато більше контейнерів у порівнянні з VM [39].

Переваги контейнерів на противагу віртуальним машинам:

- Зменшення використання ресурсів. Контейнери використовують менше накладних витрат порівняно з віртуальними машинами, оскільки вони поділяють операційну систему хоста та її ресурси, замість того, щоб запускати окрему операційну систему для кожної віртуальної машини. Це призводить до кращого використання ресурсів і дозволяє більш ефективно використовувати обладнання.
- Швидший час запуску. Контейнери можуть запускатися за лічені секунди, порівняно з віртуальними машинами, завантаження яких зазвичай займає хвилини. Це робить контейнери ідеальними для швидкого масштабування додатків і обробки мінливих робочих навантажень.
- Більш портативні. Контейнери більш портативні, ніж віртуальні машини, оскільки для їх запуску потрібен лише код програми та необхідні бібліотеки, а не ціла операційна система. Це полегшує переміщення контейнерів між різними середовищами та платформами.
- Простіші в управлінні. Контейнерами легше керувати, ніж віртуальними машинами, оскільки ними можна керувати за допомогою єдиного інтерфейсу або

інструменту командного рядка, і їх можна легко автоматизувати за допомогою таких інструментів, як Docker Compose і Kubernetes.

Відмінна дослідницька робота IBM 2014 року “An Updated Performance Comparison of Virtual Machines and Linux Containers” від Felter et al. містить порівняння між фізичними машинами, KVM і контейнерами Docker [40]. Загальний результат – Docker майже ідентичний по продуктивності до основної ОС і швидший, ніж KVM у кожній категорії.

Винятком є тільки NAT Docker – якщо ви використовуєте перекидання портів (наприклад, командою docker run -p 8080:8080), тоді можна очікувати незначного попадання в затримку, як показано нижче на рисунку 1.3. Однак тепер є можливість використовувати стек хост-мережі (наприклад, docker run --net=host) під час запуску контейнера Docker, який буде працювати ідентично стовпцю Native, як показано в результатах затримки Redis нижче на рисунку 1.4.

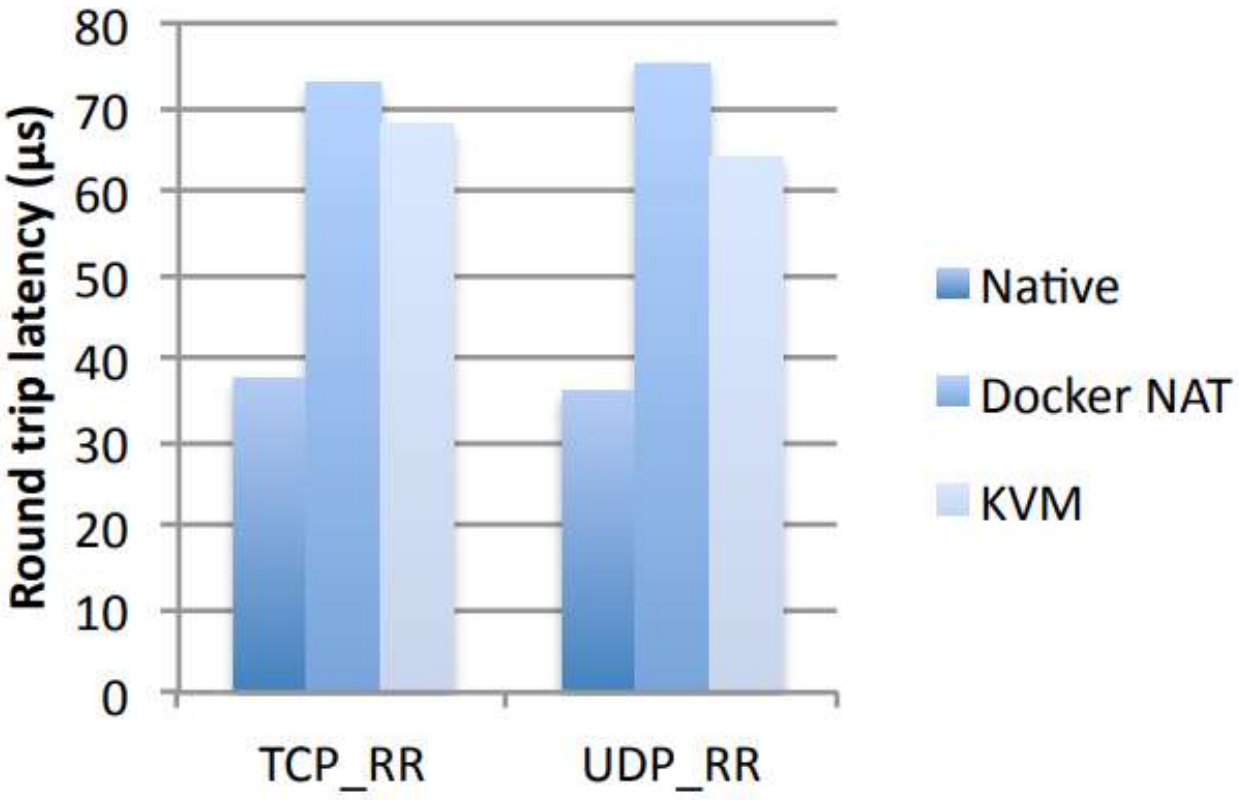

Рисунок 1.3 – Порівняння мережевих затримок TCP та UPD

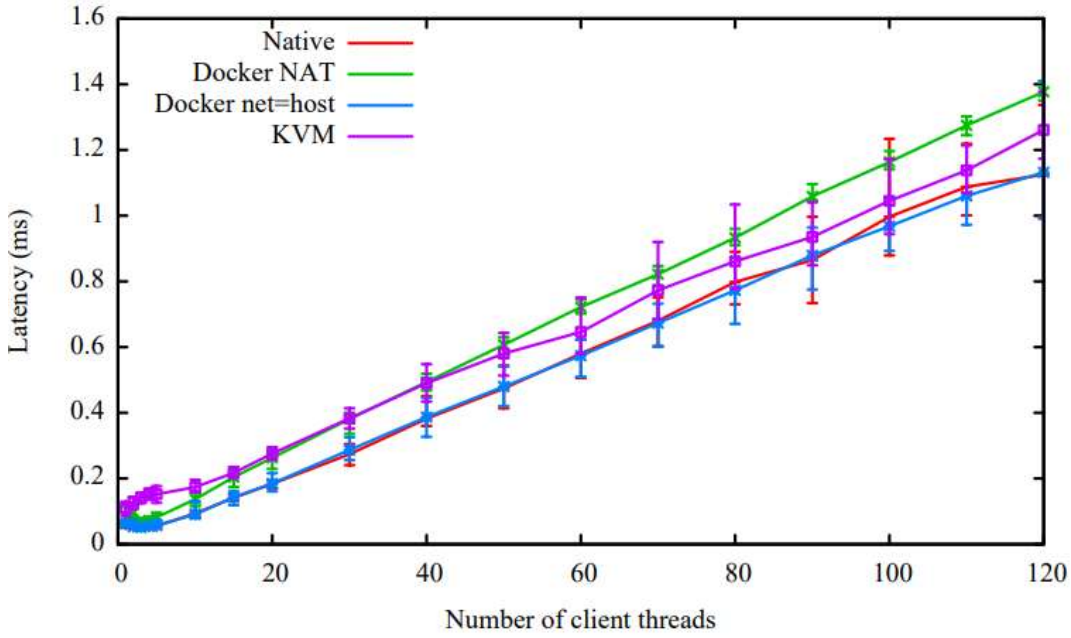

Рисунок 1.4 – Порівняння затримок для Redis

Також проведено тести затримки на кількох конкретних службах, таких як Redis. Можна побачити, що понад 20 клієнтських потоків, накладні витрати з найвищою затримкою йдуть Docker NAT, після нього KVM, і далі майже ідентичні лінії між хостом та Docker.

В цілому, контейнери пропонують ряд переваг в порівнянні як з віртуальними машинами, так і з фізичними машинами, включаючи скорочення використання ресурсів, більш швидкий час розгортання, більшу гнучкість і поліпшену узгодженість.

1.6 Проблеми, які методології DevOps не вирішують на практиці

DevOps методології створені з метою поліпшення процесу розробки, тестування та впровадження програмного забезпечення. Однак, вони не вирішують всіх проблем, що можуть виникати в процесі розробки. Нижче описані деякі з таких проблем:

- Недостатня якість коду. DevOps методології зосереджуються на автоматизації процесу розробки, тестування та впровадження, але вони не забезпечують якість коду, який розробляється. Прикладом може бути недостатня

перевірка якості коду при злитті гілок в репозиторії, що може призводити до помилок та проблем в продакшн середовищі.

- Проблеми з безпекою. DevOps методології можуть забезпечувати автоматизоване випуск програмного забезпечення, але вони не забезпечують безпеку програми. Наприклад, DevOps не може гарантувати безпеку від підкупу або фішингових атак на співробітників, що може призвести до витоку конфіденційної інформації або компрометації системи.

- Проблеми з комунікацією та координацією. DevOps зосереджується на автоматизації процесу, але не забезпечує ефективну комунікацію та координацію між розробниками та іншими відділами компанії. Це може призвести до непорозумінь та затримок у проектах.

- Проблеми з управлінням проектом. DevOps методології зосереджуються на автоматизації процесу, але не забезпечують ефективне управління проектом. Це може призвести до проблем з плануванням, координацією ресурсів та оцінкою ризиків, що може призвести до затримок проекту або до втрати грошей та часу.

- Проблеми з тестуванням. DevOps методології можуть автоматизувати процес тестування, але не гарантують, що тести виявлять всі проблеми. Наприклад, тести можуть бути недостатніми або не покривати всі можливі варіанти використання, що може призвести до випуску програмного забезпечення з помилками.

- Проблеми зі зберіганням інформації. DevOps можуть забезпечити автоматизовану доставку програмного забезпечення, але не забезпечують безпечне зберігання даних. Наприклад, недостатня захист від кібератак або випадкового видалення може призвести до втрати важливих даних або витоку конфіденційної інформації.

Загалом, DevOps методології можуть допомогти покращити процес розробки та впровадження програмного забезпечення, але вони не вирішують всіх проблем, які можуть виникнути в процесі розробки. Для успішної реалізації

проекту необхідно враховувати різні аспекти розробки, включаючи якість коду, безпеку, комунікацію та управління проектом.

1.7 Висновок до першого розділу

В першому розділі кваліфікаційної роботи описано DevOps методології та інструменти для автоматизації процесів при розробці веб-додатків. DevOps – це культура співпраці між розробниками та адміністраторами, яка спрямована на швидку доставку та надійну експлуатацію програмного забезпечення. DevOps виник як реакція на проблеми, що виникають при традиційних методологіях, таких як водоспад або Agile, де існує розрив між розробкою та експлуатацією.

Для успішної реалізації DevOps необхідно мати обсяг знань та сферу компетенцій, які охоплюють програмування, тестування, конфігурацію, безпеку, мережеве адміністрування та інше. Існують рекомендації та мапи розвитку для DevOps-інженерів, які допомагають визначити необхідні навички та шляхи їх набуття.

Одним з ключових аспектів DevOps є застосування практик та інструментів, які сприяють автоматизації процесів розробки, тестування, доставки та моніторингу веб-додатків. Серед них можна виділити CI/CD, яке дозволяє забезпечити постійне оновлення програмного забезпечення з максимальною якістю та мінімальними помилками.

Іншим важливим на сьогодні інструментом DevOps є Docker – платформа для створення, запуску та управління контейнерами, які ізолюють веб-додатки від середовища виконання. Контейнери дозволяють зробити веб-додатки більш портативними, швидшими та безпечними. Для оркестрації контейнеризованих веб-додатків використовуються системи керування кластерами, такі як Kubernetes, який дозволяє автоматично масштабувати, розподіляти навантаження та вирішувати проблеми.

Також, хмарні технології – це ще один фактор, який сприяє успіху DevOps. Хмарні сервіси надають гнучкість, доступність та економну ефективність для

розгортання веб-додатків. Існують різні моделі хмарних сервісів, такі як IaaS (Infrastructure as a Service), PaaS (Platform as a Service) та SaaS (Software as a Service), які відповідають різним потребам розробників та користувачів. За допомогою хмарних сервісів DevOps-інженери можуть швидко створювати, тестувати та випускати високоякісні продукти з мінімальною затратою ресурсів.

Визначено перелік проблем, які DevOps методології не дозволяють вирішити на практиці.

В кінці розділу досліджено та наведено переваги застосування DevOps методологій для покращення якості, швидкості та надійності розробки веб-додатків.

2 ОБГРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ ДЛЯ МОНІТОРИНГУ СЕРЕДОВИЩА РОЗГОРТАННЯ ТА АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ПРИ РОЗРОБЦІ ВЕБ-ДОДАТКУ «ФАЙЛОВИЙ МЕНЕДЖЕР»

Веб-додаток «Файловий менеджер» розроблено під час роботи над бакалаврською роботою [41] і на його основі застосовуватимуться DevOps практик CI/CD для автоматизації процесів та моніторингу. Для контролю версій розробка велася засобами GitHub, тому оптимально буде скористатися вбудованим інструментом для безперервної інтеграції та доставки – GitHub Actions. Також для локальної розробки було створено Dockerfile для ізольованого середовища виконання в Docker контейнері, що значно спростить налаштування розгортання в сервісі хмарного провайдера. В якості провайдера для робочого середовища вибрано Google Cloud Platform – він є достатньо популярним та надає 200$ для безкоштовного ознайомлення. Також для покупки доменного імені використано український сервіс ukraine.com.ua, який також надає послуги хостингу, сертифікатів та ін. Для генерації SSL-сертифікатів використано certbot інструмент, який безкоштовно верифікує домен та генерує сертифікати. Для моніторингу архітектури та продуктивності додатка використано вбудований інструмент Cloud Logging та Monitoring від GCP.

2.1 Реалізація практики CI/CD за допомогою сервісу GitHub Actions

GitHub Actions – це платформа безперервної інтеграції та безперервної доставки (CI/CD), яка дозволяє автоматизувати конвеєри для збирання, тестування та розгортання ПЗ. Робочі процеси (англ. workflow) можна створювати для збирання та тестування кожного PR для в репозиторії або розгортання злитих PR в основну гілку.

В середовищі GitHub, Actions дозволяють запускати робочі процеси на певні події, що відбуваються у репозиторії – створення нової проблеми (англ. issue) чи запустити новий робочий процес після виконання іншого, або

оновлення коду в репозиторії на додавання тегів. Також GitHub надає віртуальні машини Linux, Windows та macOS для запуску робочих процесів. Доступні варіанти із власними віртуальними машинами – локально або в хмарній інфраструктурі.

GitHub Action можна налаштувати створивши файл із розширенням .yml у папці .github/workflows, це буде файл робочого процесу. Загальну схему виконання робочих процесів, а також їхню структуру можна побачити на рисунку 2.1. У файлі потрібно описати при якій умові відбується запуск дій, зазвичай це певна подія як відкриття PR чи додавання коміта. Дії які виконуються в цьому робочому процесі називаються завданнями – jobs і відповідно позначаються у конфігураційному файлі, як і кроки – steps, які позначають що потрібно зробити в завданні. Кроки можуть бути як простими скриптами так і готовими написаними наперед програмами – actions, які згуртовують часто повторювані команди. Також є так званий ранер (англ. runner) – це сервер де виконується весь робочий процес з описаними в ньому кроками.

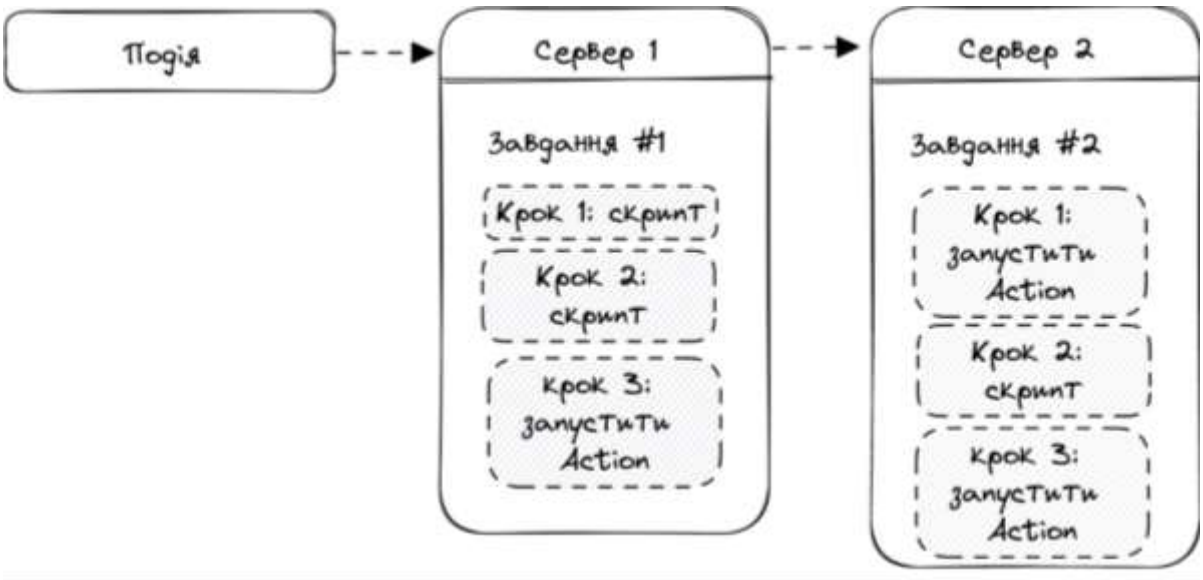

Рисунок 2.1 – Схема компонентів GitHub Actions

Загалом виділено такі компоненти:

- Події.
- Завдання.

- Кроки.
- Програми Actions.
- Ранери – сервери.

Скористаймося прикладом робочого процесу з документації GitHub Actions. В лістингу В.1 дадатка В наведено приклад базового файлу для конфігурації робочого процесу.

Цей файл є конфігураційним файлом для GitHub Actions workflow, який називається learn-github-actions.yml. Він запускається при кожному push-і в репозиторій.

Це одна робота (job) з назвою «check-bats-version», яка виконується в операційній системі Ubuntu останньої версії.

У роботі є кілька кроків (steps):

- Виконується дія (action) «checkout» з останньою версією (v3) для клонування репозиторію в актуальну гілку.
- Виконується дія «setup-node» з останньою версією (v3) для встановлення відповідної версії Node.js (14).
- Виконується команда «npm install -g bats» для глобальної інсталяції тестового фреймворку Bats.
- Виконується команда «bats -v» для виведення версії Bats у консоль.

Підсумовуючи зауважимо, що цей GitHub Actions робочий процес виконує перевірку версії тестового фреймворку Bats при кожному push-і в репозиторій в середовищі Ubuntu.

2.2 Підготовка до розгортання програмного продукту шляхом розміщення додатку в контейнері Docker

Docker є платформою з відкритим вихідним кодом, яка дозволяє розробляти, доставляти та запускати додатки. Він надає можливість відокремити програми від інфраструктури, спрощуючи тим самим швидку доставку програмного забезпечення. Docker дозволяє керувати як інфраструктурою, так і

додатками. Використовуючи методології Docker для швидкої доставки, тестування та розгортання коду, можна значно зменшити час між написанням коду і його запуском у робочому середовищі.

Docker надає можливість упаковувати та запускати програми в ізольованому середовищі, відомому як контейнер. Цей контейнер забезпечує високий рівень ізоляції та безпеки, що дозволяє запускати багато контейнерів одночасно на одному хості. Контейнери є легкими та містять усе необхідне для виконання програми, що означає, що вам не потрібно залежати від налаштувань хоста. Завдяки Docker ви можете легко спільно використовувати контейнери під час роботи і мати впевненість, що всі, хто з вами працює, використовують однаковий контейнер, який працює ідентично.

Docker надає інструменти та засоби для управління контейнерами:

- Розробка додатку і його допоміжних компонентів за допомогою контейнерів.
- Контейнер стає одиницею для розповсюдження та тестування додатка.
- Розгортання програми у виробничому середовищі у вигляді контейнера або організованої служби. Це працює однаково, незалежно від того, чи є виробниче середовище локальним центром обробки даних, постачальником хмарних послуг або гібридом цих двох.

Docker спрощує розробку програмного забезпечення, дозволяючи розробникам працювати у стандартизованому середовищі, використовуючи локальні контейнери, в яких розміщуються їх програми та сервіси. Контейнери ідеально підходять для використання у безперервній інтеграції та процесах безперервної доставки (CI/CD). Ось приклад, що демонструє це:

- Розробники створюють код локально і діляться своєю роботою зі своїми колегами за допомогою контейнерів Docker.
- Використовують Docker для розміщення своїх додатків у тестовому середовищі та виконання автоматизованих та ручних тестів.

• Коли розробники зустрічаються з помилками, вони можуть виправити їх у середовищі розробки та повторно розгорнути додаток в тестовому середовищі для подальшого тестування та перевірки.

• Коли тестування завершено, доставити виправлення клієнту так само просто, як перенести оновлений образ у виробниче середовище.

Контейнерна платформа Docker дозволяє виконувати високопортативні навантаження. Контейнери Docker здатні функціонувати на персональних ноутбуках розробників, фізичних або віртуальних серверах у центрах обробки даних, хмарних платформах або в різноманітних гібридних середовищах.

Портативність і легка природа Docker також дозволяють динамічно керувати робочими навантаженнями, збільшуючи або зменшуючи програми та послуги, як диктують бізнес-потреби, майже в режимі реального часу.

Docker легкий і швидкий. Він пропонує життєздатну, економічно ефективний аналог VM на основі віртуалізаційної платформи. Це дозволяє максимально використовувати власні серверні ресурси з метою досягнення бізнес-цілей. Docker є оптимальним варіантом для розгортання в обмежених середовищах і підходить для малих та середніх розгортань, де досягнення більшої продуктивності за рахунок ефективного використання обмежених ресурсів є важливим.

Docker використовує клієнт-серверну архітектуру. Клієнт Docker розмовляє з фоновою службою Docker, яка виконує важку роботу зі збирання, запуску та розповсюдження контейнерів Docker, що зображено на рисунку 2.2.

Клієнтський компонент та сервіс Docker можуть функціонувати на одній системі, або ви можете налаштувати з'єднання клієнта Docker з віддаленим сервісом Docker. Клієнт і фонова служба Docker спілкуються за допомогою REST API, через сокет UNIX або мережевий інтерфейс. Ще одним клієнтом Docker є Docker Compose, який дозволяє працювати з додатками, що складаються з набору контейнерів.

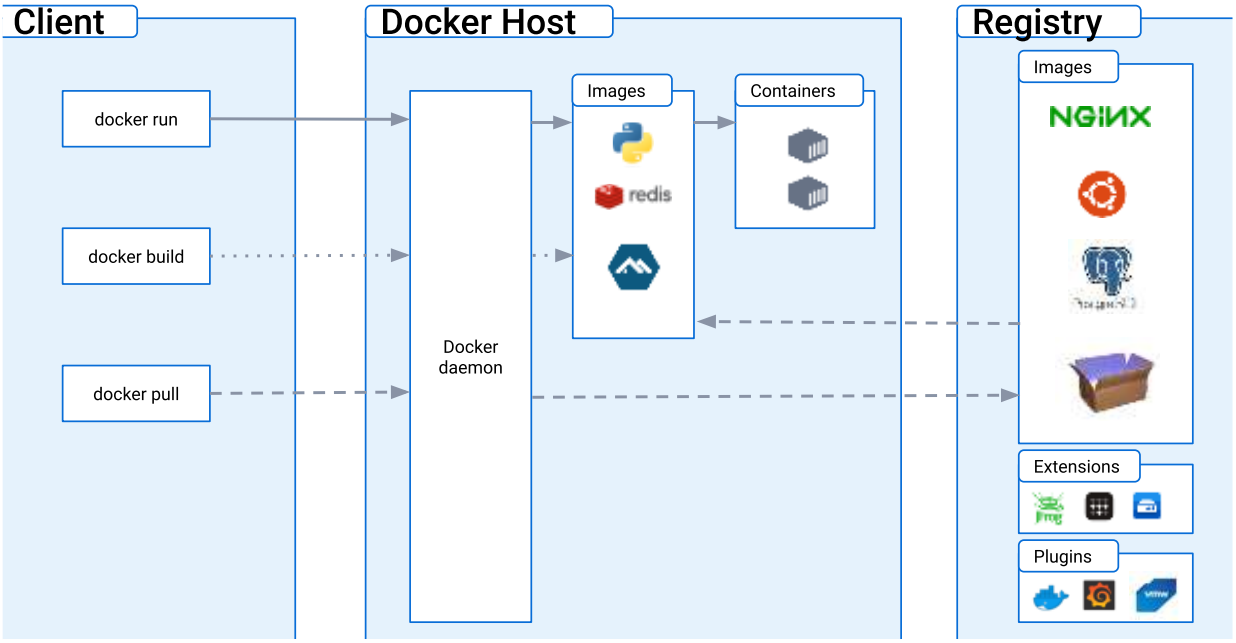

Рисунок 2.2 – Архітектура Docker

Фоновий сервіс Docker (dockerd) приймає запити до Docker API та виконує управління об'єктами Docker, такими як образи, контейнери, мережі та томи. Фонова служба також може зв'язуватися з іншими фоновими службами для керування службами Docker.

Клієнт Docker (docker) є основним інтерфейсом, через який багато користувачів взаємодіють з Docker. При використанні команд, таких як «docker run», клієнт передає ці команди до фонової служби Docker (dockerd), яка їх виконує, а далі dockerd взаємодіє з Docker API. Клієнт Docker має можливість взаємодіяти з однією або кількома одночасно запущеними сервісами Docker.

Docker Desktop – програма для середовища Mac, Windows або Linux, яка дозволяє створювати та обмінюватися контейнеризованими програмами та мікросервісами. Docker Desktop включає в себе фонову службу Docker (dockerd), клієнт Docker (docker), Docker Compose.

Реєстр Docker надає образи Docker. Docker Hub – це репозиторій, який доступний для всіх і зазвичай використовується Docker для пошуку контейнерних образів на Docker Hub за замовчуванням. Крім того, у вас є можливість створити власний приватний репозиторій.

Команди «docker pull» та «docker run» дозволяють отримати потрібні образи з стандартного або вказаного реєстру. Команда «docker push» дозволяє відправити образ до вибраного реєстру.

Docker працює з різними об'єктами, такими як образи, контейнери, мережі, томи, плагіни тощо. Ось стислий опис деяких з них.

Образи Docker – це готові шаблони, які містять інструкції для створення контейнерів. Зазвичай образи створюються на основі інших образів з додаванням певних налаштувань. Наприклад, ви можете побудувати образ, який використовує Debian як основу, але додає веб-сервер Nginx та ваш додаток, а також необхідні параметри конфігурації для роботи додатка.

Для того, щоб зібрати свій власний образ, потрібно написати Dockerfile з простим синтаксисом для опису кроків, які потрібно виконати для побудови образу та його запуску. Кожна команда у файлі Dockerfile додає шар до образу. Коли файл Dockerfile змінюється і образ перебудовується, то перебудовуються лише ті шари, які були змінені. Це одна з причин, чому образи такі легкі, маленькі та швидкі в порівнянні з іншими технологіями віртуалізації.

Контейнер – це образ, який запущено. Ви можете керувати контейнером за допомогою Docker API або CLI: створювати, запускати, зупиняти, переміщувати або видаляти. Також можна налаштувати сховище та мережеві з'єднання контейнера або навіть створити новий образ на основі його поточного стану.

Контейнер має достатній рівень ізоляції від інших контейнерів та машини-хоста за замовчуванням. Можна налаштувати рівень ізоляції мережі, сховища та інших основних підсистем контейнера від інших контейнерів або від хост-машини.

Образ і параметри конфігурації визначають контейнер. Якщо контейнер зупиняється, його стан не зберігається, якщо він не в постійному сховищі.

Щоб запустити контейнер debian та приєднати його інтерактивно до локального сеансу командного рядка, виконуючи «/bin/zsh», можна скористатися такою командою: «Docker run -i -t debian /bin/zsh».

Ця команда виконує наступні дії (за умови використання конфігурації реєстру за замовчуванням):

Якщо образ ubuntu недоступний локально, Docker витягує його з налаштованого реєстру, якби «docker pull debian» був запущений вручну.

Коли ви виконуєте команду «create docker container», Docker створює новий контейнер для вас.

Docker виділяє файлову систему читання-запису в контейнер, як її останній шар. Це дозволяє запущеному контейнеру створювати або змінювати файли та каталоги у своїй файловій системі.

Якщо ви не вказуєте параметри мережі, Docker створює мережевий інтерфейс за замовчуванням для підключення контейнера до мережі. Також він призначає IP-адресу контейнеру. Контейнери можуть зв'язуватися з зовнішніми мережами через мережеве підключення хост-машини за замовчуванням.

Docker створює контейнер та запускає «/bin/zsh» в ньому. Контейнер працює в інтерактивному режимі та підключений до терміналу (завдяки опціям -i та -t), тому можна вводити команди з клавіатури і бачити результати на екрані.

Коли ви вводите exit для закінчення команди «/bin/zsh», контейнер припиняє роботу, але не зникає. Ви можете запустити його знову або вилучити.

Docker створений на мові програмування Go і використовує можливості декількох функцій ядра Linux для надання його функціональності. Docker застосовує технологію під назвою «namespaces» для створення ізольованого робочого середовища, яке називається контейнером. Коли контейнер запускається, Docker створює набір просторів імен для цього контейнера.

Ці простори імен надають шар ізоляції. Кожен елемент контейнера виконується в окремому просторі імен, і його доступ обмежено цим простором імен.

Під час роботи над веб-додатком «Файловий менеджер» було створено Dockerfile, що поданий в лістингу 2.1, для використання разом з Docker-Compose, щоб можна було підняти весь додаток лише однією командою – створиться БД та сам додаток.

Лістинг 2.1 – Вміст Dockerfile для контейнеризації веб-додатку

```
#node dockerfile
FROM node:lts-slim
WORKDIR /usr/src/my-app
COPY package.json .
RUN npm install
EXPOSE 3000
COPY . .
RUN npm run build:frontend
RUN npm run build:backend
RUN apt update && apt install netcat -y
```

Цей Dockerfile створює образ для додатку Node.js. Він використовує базовий образ node:lts-slim, який містить останню стабільну версію Node.js. Він встановлює робочу директорію «/usr/src/my-app» і копіює файл package.json туди. Далі запускає «npm install» для встановлення залежностей проекту,відкриває порт 3000 для мережевого доступу. Копіює усі файли проекту в робочу директорію і запускає «npm run build:frontend» і «npm run build:backend» для створення продакшн-версій фронтенду і бекенду. Він також встановлює утиліту netcat за допомогою «apt update і apt install netcat -y», вона необхідна для запуску скрипта, який перевіряє доступність БД, при використанні з docker-compose.

Для локального та тестового розгортання достатньо docker-compose файлу, вміст якого подано в лістингу В.2, та однієї команди для старту – docker-compose up. Проте для робочого середовища це спричинить складнощі та додаткові дії необхідні із ручним керуванням стану додатку при релізі нових версій. Тому в подальших розділах буде описано інших підхід до розгортання ПЗ, використовуючи інструменти GCP.

Цей docker-compose файл містить опис двох сервісів, «db» та «web».

Сервіс «db» використовує образ «mysql:8.0.21» та має назву контейнера «db». Сервіс завжди перезапускається («restart: always») та має змінні середовища, які передаються з зовнішнього середовища, такі як «MYSQL_DATABASE», «MYSQL_USER», «MYSQL_PASSWORD», та

«MYSQL_ROOT_PASSWORD». Крім того, використовується зовнішній том для зберігання даних бази даних та порти «3306» відкриті для з'єднань.

Сервіс «web» має команду «scripts/run.sh», яка буде запущена при запуску контейнера, та будується з контексту поточної директорії («context: .») та Dockerfile («dockerfile: Dockerfile»). Цей сервіс має назву контейнера «web» та завжди перезапускається. Зовнішній том використовується для зберігання файлів користувача та порти «3000» відкриті для з'єднань. Крім того, він залежить від сервісу «db».

Скрипт при запуску "web» контейнера запускає міграції БД, якщо вона доступна, та сам веб-додаток.

2.3 Застосування хмарного провайдера Google Cloud Platform

2.3.1 Загальна характеристика провайдера

Google Cloud Platform (GCP) – це сукупність хмарних сервісів, які використовують ту саму технологію, що й Google для своїх продуктів, таких як Пошук Google, Gmail, Google Диск і YouTube. Він пропонує різноманітні хмарні рішення для обчислень, зберігання, аналізу і навчання машин. Для використання GCP потрібно мати кредитну картку або банківський рахунок.

Google Cloud Platform надає можливість користуватися інфраструктурою, платформою та безсерверними обчисленнями як послугами.

У 2008 році Google представила App Engine, платформу для створення і розгортання веб-додатків у хмарних центрах даних Google, яка стала першим хмарним продуктом компанії. Сервіс став доступний для всіх у 2011 році. З того часу Google додала до платформи багато інших хмарних сервісів.

Google Cloud Platform є складовою Google Cloud, до якої також входять Google Workspace (G Suite), корпоративні версії Android і ChromeOS, а також API для машинного навчання і картографічних сервісів для бізнесу.

Google має понад 100 продуктів під маркою Google Cloud. У GCP можна скористатися наступними категоріями сервісів:

- Compute.
- Storage and databases.
- Networking.
- Management tools.
- Identity and security.
- Internet of things (IoT).
- API platform.

З Compute категорії для розгортання робочого середовища використовуватиметься Google Compute Engine, який дозволяє користувачам запускати віртуальні машини (VM) на вимогу. Віртуальні машини можна запускати зі стандартних образів або власних образів, створених користувачами.

Із Storage and databases секції використано Cloud SQL на основі MySQL для БД та Persistent Disk для збереження файлів веб-додатку. Також використані сервіси з категорій Networking, Management tools та Identity and security.

Станом на 1 квартал 2023 року Google Cloud Platform доступна в 37 регіонах і 103 зонах світу [42]. Регіон – це певне географічне розташування, де користувачі можуть розгортати хмарні ресурси.

Кожен регіон є самостійною географічною областю, яка складається із зон. Зона – це область розгортання ресурсів Google Cloud Platform у регіоні.

2.3.2 Порівняння ручного управління базою даних та сервісу Cloud SQL

Як і всі хороші речі в інфраструктурі, вибір самостійного розміщення вашої бази даних сповнений компромісів.

З одного боку, у вас є абсолютна свобода робити зі своєю базою даних все, що завгодно – додавання корисного розширення Postgres або експерименти з

новими технологіями. З іншого боку, тепер вам доведеться виділити ресурси для надійного збереження вашої бази даних в Інтернеті.

Коли ми говоримо про самостійний хостинг, то маємо на увазі запуск програмного забезпечення баз даних на віртуальних машинах, над якими маємо повний контроль. Іншими словами, запуск бази даних на AWS EC2, Google Cloud Engine або Azure Virtual Machines або аналогічному постачальнику VPS.

Коли говоримо про готові сервіси – то це сервіси «база даних як послуга», наприклад AWS RDS, Cloud SQL або Azure Database.

Коротко опишемо переваги використання самостійного хостингу БД.

Ціна. Одним з найбільших факторів, що спонукають людей самостійно розміщувати свою базу даних, є ціна.

На низькому рівні найдешевший рівень для AWS RDS (служба керованих баз даних AWS) становить близько 15 доларів США на місяць. Для порівняння, встановлення PostgreSQL або MySQL на сервері додатків є безкоштовним. При цьому ви втрачаєте можливість самостійно масштабувати частини вашої програми, але для невеликих проектів це нормально.

Після того, як вашій програмі знадобиться трохи більше процесора та оперативної пам'яті, і ви почнете шукати, скажімо, сервер оперативної пам'яті 4 vCPU 16 ГБ (db.m4.xlarge на AWS RDS), припускаючи, що у БД є 1 ТБ даних, виходить приблизно 762 доларів США на місяць в AWS RDS. На AWS EC2 ця конфігурація (t4g.xlarge) коштуватиме лише близько 200 доларів США на місяць.

Портативність. Самостійно розміщуючи, ви також мінімізуєте блокування постачальника. Скажімо, одного разу ви помітите, що ваша конфігурація 4 vCPU / 16 ГБ оперативної пам'яті / 1 ТБ пам'яті коштує всього 80 USD на місяць на хостах VPS, таких як Hetzner; переміщення налаштувань на власному хостингу так само просто, як повторний запуск сценаріїв налаштування та відновлення з резервної копії.

Звичайно, використання служби керованої бази даних не заважає вам використовувати іншу службу, просто набагато простіше, коли налаштування

вашої бази даних можна швидко розгорнути на будь-якій віртуальній машині Linux.

Контроль. Самостійно розміщуючи, ви можете вирішити:

- Яку ОС запускати.
- Як часто оновлювати/виправляти ОС.
- Яке інше програмне забезпечення працює на тому самому сервері, що й ваша база даних.
- розширення бази даних для запуску (наприклад, підтримка TimescaleDB великими постачальниками керованих послуг зайняла роки).
- Як часто оновлювати/виправляти програмне забезпечення бази даних.
- Як часто запускати резервні копії.
- Яку конфігурацію диска використовувати.

Переваги БД як сервісу наступні:

- Фокус. За допомогою керованих служб ви вибираєте версію бази даних, яку хочете запустити, бажаний розмір екземпляра, натискаєте «Створити», і це все.
- Резервне копіювання, незначні оновлення, завдання технічного обслуговування тощо, запускаються автоматично, залишаючи вас зосередитися на створенні програми. Звичайно, ви платите за зручність автоматизації цих завдань, але це звільняє інженерів для роботи над функціями продукту, а не налаштування рутин.
- Масштабованість. Однією з найбільших переваг керованих сервісів є простота масштабування бази даних вгору або вниз в будь-який момент. Натисканням кнопки (і пару хвилин простою, залежно від того, як ви налаштували базу даних) ви можете перейти від найменшого рівня бази даних до будь-якого розміру, необхідного для вашого робочого навантаження, і знову відступити, як тільки ви зрозумієте потреби в ресурсах вашого робочого навантаження. Плата відбувається лише за використані ресурси, тому, якщо

вирішено, що потрібна значно покращена машина протягом 12 годин, ви платите лише за ці 12 годин.

- Підтримка. Коли ви вирішите самостійно керувати БД, найкраща безкоштовна підтримка, яку, ймовірно, отримаєте – це списки розсилки та форуми. Тоді як для БД як сервіс, частина витрат покриває базову підтримку від спеціалістів, які спеціалізуються на вашій базі даних.

Хоча вони не зможуть дати вам безкоштовні індивідуальні поради щодо того, як створити вашу програму, вони можуть допомогти з аналізом першопричин, коли трапляються інциденти, і надати загальну пораду.

Отже, беручи до уваги що найкраща підтримка та стабільність забезпечуються хмарним провайдером вирішено використовувати сервіс Cloud SQL для створення MySQL БД для Файлового менеджера.

2.3.3 Виділений сервер Compute Engine для веб-додатку

Оскільки «Файловий менеджер» вже має готовий Dockerfile, який перевірений локальною розробкою, то є сенс скористатися цією перевагою та використати можливості GCP для контейнеризованих додатків.

Google Cloud Platform (GCP) пропонує різноманітні обчислювальні сервіси, які дозволяють запускати ваші програми в хмарі. Однією з таких служб є Compute Engine, яка дозволяє створювати та керувати віртуальними машинами (VM), які можуть запускати будь-яку операційну систему та програмне забезпечення, яке ви хочете.

Compute Engine надає вам високопродуктивні, масштабовані та надійні віртуальні машини, які можуть впоратися з будь-яким робочим навантаженням. Ви можете вибрати один із попередньо визначених типів машин або налаштувати власний відповідно до ваших потреб. Ви також можете використовувати такі функції, як спотові віртуальні машини та графічні процесори, щоб оптимізувати свої витрати та продуктивність.

За допомогою Compute Engine надається повний контроль над віртуальними машинами та їх конфігурацією. Ви можете отримати доступ до них через SSH, використовувати Cloud Console або Cloud SDK для управління ними, а також використовувати хмарний моніторинг і хмарний журнал для моніторингу їх здоров'я та продуктивності. Ви також можете приєднати постійні диски, зовнішні IP-адреси та мережеві інтерфейси до своїх віртуальних машин.

Compute Engine інтегрований з іншими службами GCP, такими як Cloud Storage, Cloud SQL, Cloud Load Balancing і Cloud DNS. Ці служби можна використовувати для зберігання даних, підключення до баз даних, розподілу трафіку та розпізнавання доменних імен. Ви також можете використовувати Compute Engine зі сторонніми інструментами та фреймворками, такими як Kubernetes, TensorFlow та Apache Hadoop.

Compute Engine призначений для безпеки та відповідності безпековим вимогам. Ви можете використовувати шифрування, брандмауери, керування ідентифікацією та доступом (IAM), а також службові облікові записи для захисту своїх віртуальних машин і даних. Ви також можете скористатися глобальною інфраструктурою та мережею GCP, які пропонують низьку затримку, високу доступність та надійність. Compute Engine також підтримує різні стандарти та сертифікати відповідності вимог, такі як ISO 27001, PCI DSS, HIPAA та GDPR.

Оптимізована для контейнерів ОС від Google – це образ операційної системи для віртуальних машин Compute Engine, оптимізований для запуску контейнерів. Оптимізована для контейнерів ОС підтримується Google і базується на проекті Chromium OS з відкритим кодом. За допомогою ОС, оптимізованої для контейнерів, ви можете швидко, ефективно та безпечно відкривати свої контейнери на хмарній платформі Google.

Оптимізована для контейнерів ОС надає наступні переваги:

- Запуск контейнерів з коробки. Оптимізовані для контейнерів ОС поставляються з попередньо встановленими Docker і контейнерними середовищами виконання та «cloud-init». За допомогою ОС оптимізованої для

контейнера, ви можете відкрити свій контейнер одночасно зі створенням віртуальної машини, без налаштування хоста.

• Менша поверхня атаки. оптимізована для контейнерів ОС має менший простір взаємодії, зменшуючи потенційні методи для атаки вашого екземпляра.

• Заблоковано за замовчуванням. ОС оптимізовані для контейнерів, за замовчуванням включають заблокований брандмауер та інші налаштування безпеки.

• Автоматичні оновлення. ОС оптимізовані для контейнерів, налаштовані на автоматичне завантаження щотижневих оновлень у фоновому режимі. Для використання останніх оновлень необхідне тільки перезавантаження.

Перейдемо до задачі виділення ресурсів для збереження даних додатком.

2.3.4 Виділений диск для файлів веб-додатку

Веб-додаток «Файловий менеджер» передбачає збереження користувацьких файлів, тому необхідно забезпечити VM додатковим сховищем даних.

За замовчуванням кожен екземпляр віртуальної машини Compute Engine (VM) має один том завантажувального диска, який містить операційну систему. Коли вашим програмам потрібен додатковий простір для зберігання, одним із можливих рішень є приєднання додаткових томів диска до віртуальної машини.

Постійні томи дисків – це довговічні мережеві пристрої зберігання даних, до яких ваші віртуальні машини можуть отримати доступ, як фізичні диски на робочому столі або сервері. Дані на кожному постійному диску розподіляються між кількома фізичними дисками. Compute Engine керує фізичними дисками та розподілом даних для користувача, щоб забезпечити резервування та оптимальну продуктивність.

Томи з постійними дисками розташовані незалежно від віртуальних машин, тому можна від'єднати або перемістити томи, щоб зберегти дані навіть після видалення екземплярів VM. Продуктивність постійного диска автоматично

масштабується залежно від розміру, тому можна змінити розмір наявних томів постійного диска або додати більше томів постійного диска до віртуальної машини, щоб задовольнити вимоги до продуктивності та дискового простору.

Згідно з документацією для Compute Engine та Disk [43, 44] після або під час створення віртуальної машини можна прикріпити диск до неї. Під час створення конвеєра в 3 розділі буде підключено цей диск, проте його необхідно створити спочатку, що наведено в додатку В у лістингу В.3.

Необхідно зауважити що всі ресурси повинні бути в одній зоні, при створенні диска вказано таку ж зону що і у віртуальної машини. Після виконання цієї команди було створено диск з назвою «file-manager-disk» місткістю 10GB та для архітектури VM X86_64.

2.4 Налаштування доменного імені та SSL-сертифікатів

Важливою складовою роботи DevOps-інженера є налаштування DNS для доменного імені та забезпечення безпечного зв’язку SSL-сертифікатами для веб-сервісів. Для «Файлового менеджера» було обрано доменне ім’я з таким URL: mod-man.online. А для генерації сертифікатів вибрано безкоштовний інструмент certbot від організації Letsencrypt (рис. 2.3).

В пошуковику було знайдено провайдера доменних імен в Україні – ukraine.com.ua, і вибрано .online домен першого рівня через його низьку ціну за перший рік. При створенні віртуальної машини буде виділятися зовнішня IP-адреса, яку необхідно буде ввести в поле «Дані» в панелі керування доменом, що зображено на рисунку 2.4. Оновлення кешу DNS може зайняти до 24 годин.

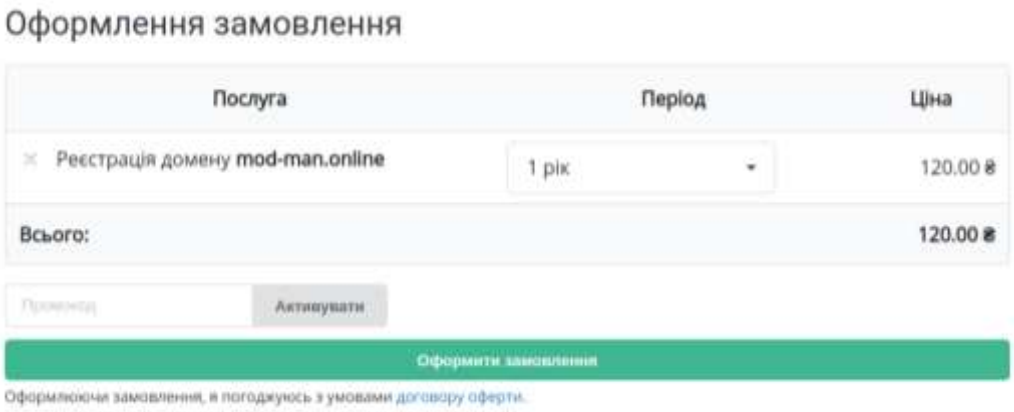

Рисунок 2.3 – Вибір та реєстрація доменного імені

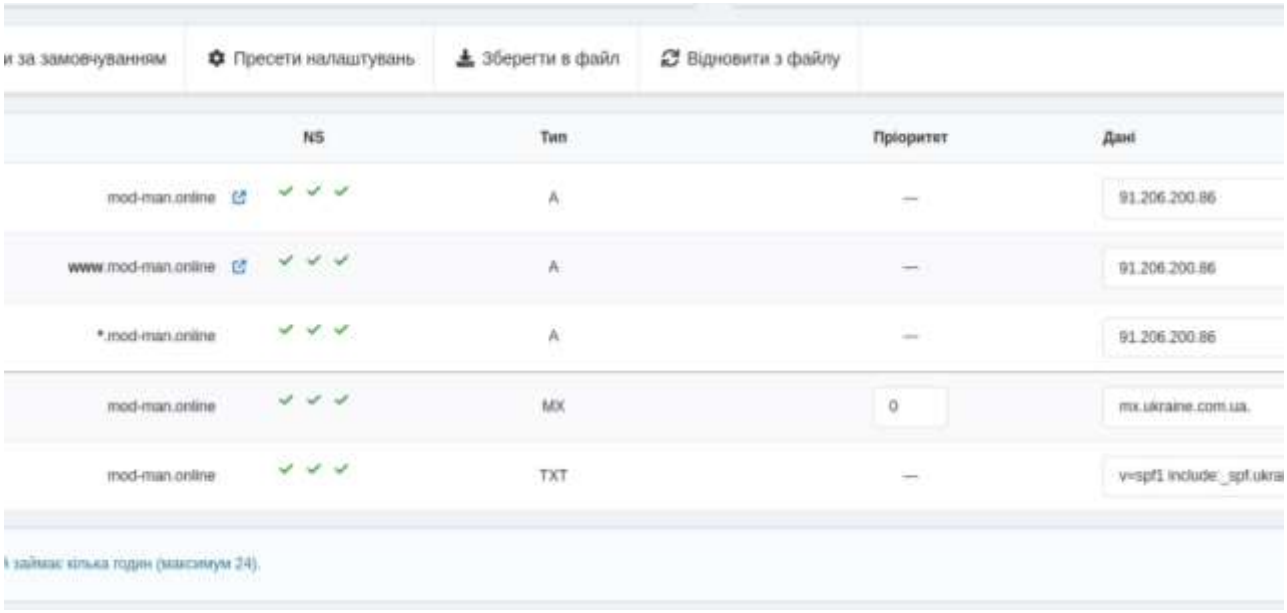

Рисунок 2.4 – Панель керування доменом

Система доменних імен (DNS) – це ієрархічна та розподілена система іменування комп'ютерів, служб та інших ресурсів в Інтернеті або інших мережах Інтернет-протоколу (IP). Він пов'язує різну інформацію з доменними іменами, призначеними кожному з пов'язаних об'єктів [45]. Найбільш помітним є те, що він перетворює легко запам'ятовувані доменні імена на числові IP-адреси, необхідні для пошуку та ідентифікації комп'ютерних служб та пристроїв за допомогою базових мережевих протоколів. Система доменних імен є важливим компонентом функціональності Інтернету з 1985 року.

Публічний ключ або цифровий сертифікат (раніше називався сертифікатом SSL) використовує публічний ключ і приватний ключ для забезпечення безпечного зв'язку між клієнтською програмою (веб-браузером, поштовим клієнтом тощо) і сервером через зашифрований SSL (рівень захищених сокетів) або TLS (безпека транспортного рівня). Сертифікат використовується як для шифрування початкового етапу зв'язку (обмін захищеними ключами), так і для ідентифікації сервера. Сертифікат містить відомості про ключ, ідентифікатор сервера та цифровий підпис видавця сертифіката. Якщо програмне забезпечення, яке ініціює обмін даними, довіряє видавцю, а підпис дійсний, ключ можна використовувати для безпечного зв'язку із сервером, визначеним сертифікатом. Використання сертифіката є хорошим способом запобігання атакам «людина посередині», коли хтось, хто знаходиться між вами та сервером, з яким, на вашу думку, ви розмовляєте, може вставити свій власний (шкідливий) вміст.

Certbot – це безкоштовний програмний інструмент з відкритим вихідним кодом [46], який автоматизує процес отримання цифрового сертифіката від Let's Encrypt [47] та налаштування шифрування HTTPS на веб-сервері. Він також може виступати в якості клієнта для будь-якого іншого центру сертифікації (CA), який використовує протокол ACME.

Certbot використовує ряд різних команд (також званих «підкомандами») для запиту конкретних дій, таких як отримання, оновлення або відкликання сертифікатів.

Certbot помагає досягти двох завдань:

1. Отримання сертифіката: автоматичне виконання необхідних кроків автентифікації, щоб довести, що ви контролюєте домен(и).

2. Збереження сертифіката в «/etc/letsencrypt/live/» та регулярне поновлення його.

За бажанням можна встановити цей сертифікат на підтримувані веб-сервери (наприклад, Apache або nginx) та інші типи серверів. Це робиться шляхом автоматичної зміни конфігурації вашого сервера для використання сертифіката.

Для виконання цих завдань Certbot попросить вас вибрати один із ряду плагінів автентифікатора та інсталятора (табл. 2.1). Відповідний вибір плагінів буде залежати від того, яке серверне програмне забезпечення ви використовуєте і де плануєте використовувати свої сертифікати.

Таблиця 2.1 – Плагіни для роботи Certbot

Плагін	**Аутенти-фікатор**	**Інста-лятор**	**Опис**	**Типи викликів (і порт)**
1	**2**	**3**	**4**	**5**
apache	Так	Так	Автоматизація отримання та встановлення сертифіката за допомогою Apache.	http-01 (80)
nginx	Так	Так	Автоматизує отримання та встановлення сертифіката за допомогою Nginx.	http-01 (80)
webroot	Так	Ні	Отримання сертифіката шляхом запису до каталогу webroot у вже запущений веб-сервер.	http-01 (80)
standalone	Так	Ні	Використовує «автономний» веб-сервер для отримання сертифіката. Потрібен порт 80, щоб бути доступним.	http-01 (80)
DNS plugins	Так	Ні	Дана категорія плагінів автоматизує отримання сертифікату шляхом змінення записів DNS для підтвердження контролю над доменом. Виконання перевірки домену таким способом – це єдиний спосіб отримати wildcard сертифікати від Let's Encrypt.	dns-01 (53)
manual	Так	Ні	Отримання сертифікатів шляхом виконання інструкцій вручну.	http-01 (80) або dns-01 (53)

Аутентифікатори – це плагіни, які автоматично виконують необхідні кроки, щоб довести, що ви керуєте доменними іменами, для яких ви намагаєтеся

згенерувати сертифікат. Для отримання сертифіката завжди потрібен автентифікатор.

Інсталятори – це плагіни, які можуть автоматично змінювати конфігурацію вашого веб-сервера для обслуговування вашого веб-сайту через HTTPS, використовуючи сертифікати, отримані Certbot. Інсталятор потрібен лише в тому випадку, якщо ви хочете, щоб Certbot встановив сертифікат на ваш веб-сервер.

Деякі плагіни є автентифікаторами та інсталяторами, і можна вказати окрему комбінацію автентифікатора та плагіна.

Плагіни використовують однин з декількох викликів протоколу ACME, щоб довести, що ви контролюєте домен. Варіанти http-01 (який використовує порт 80) і dns-01 (вимагає конфігурації DNS-сервера на порту 53, хоча це часто не той самий комп'ютер, що ваш веб-сервер). Кілька плагінів підтримують більше одного типу викликів, і в цьому випадку ви можете вибрати необхідний з параметром «--preferred-challenges».

2.5 Моніторинг інфраструктури та продуктивності додатків з Cloud Logging та Monitoring

Моніторинг інфраструктури та продуктивності додатків (APM) – це процес вимірювання та керування роботою програмних систем, що використовуються в організаціях. Метою APM є забезпечення високої якості та доступності програм для кінцевих користувачів, а також виявлення та усунення проблем з продуктивністю, що можуть негативно впливати на бізнес-процеси.

Cloud Logging та Monitoring – це процеси зберігання логів у всіх продуктах Cloud з можливістю пошуку, моніторингу та оповіщення на основі різних показників [48]. Операційний пакет Google Cloud (раніше Stackdriver) забезпечує інтегровані служби моніторингу, реєстрації та керування відстеженням для програм і систем, що працюють у Google Cloud та за його межами [49].

Cloud Logging – це сервіс, який збирає та зберігає журнали аудиту, подій та метрик від різних джерел в GCP. Ви можете переглядати, фільтрувати та

пошуковувати журнали в консолі GCP або за допомогою API. Ви також можете експортувати журнали в інші сервіси GCP, такі як Cloud Storage, BigQuery або Pub/Sub для подальшої обробки та аналітики.

Cloud Monitoring – це сервіс, який збирає та вимірює метрики про роботу та наявність вашої хмарної інфраструктури та додатків. Ви можете створювати спостереження за метриками, налаштовувати сповіщення про порушення порогових значень або аномалій, створювати дашборди для візуалізації метрик у реальному часі або історично. Ви також можете інтегрувати Cloud Monitoring з іншими сервісами GCP або сторонніми інструментами для розширення функціональності моніторингу.

Основними перевагами використання APM та Cloud Logging та Monitoring в GCP є:

- Можливість отримувати централізований доступ до всієї інформації про продуктивність програм і інфраструктури в одному місці.
- Можливість налаштовувати сповіщення та правила ескалації для реагування на проблеми з продуктивністю або порушеннями SLA (Service Level Agreement).
- Можливість створювати інформаційні панелі та звіти для визначення тенденцій, проблемних областей та можливостей для поліпшення продуктивності.
- Можливість інтегруватися з іншими сервісами GCP, такими як Cloud Trace, Cloud Debugger, Cloud Profiler тощо, для отримання більш детальної інформації про продуктивність програм на різних рівнях.

Моніторинг інфраструктури та продуктивності є важливими аспектами управління хмарними сервісами. Cloud Logging та Monitoring в gcp дозволяють збирати, аналізувати та візуалізувати дані про стан та поведінку вашої хмарної інфраструктури та додатків. Вони також допомагають виявляти та усувати проблеми, оптимізувати ресурси та покращувати якість обслуговування.

2.6 Висновок до другого розділу

В другому розділі кваліфікаційної роботи досліджено та обґрунтовано вибір інструментів для моніторингу та автоматизації процесів при розробці веб-додатку «Файловий менеджер». Розглянуто переваги використання CI/CD засобами GitHub Actions для забезпечення якості коду та швидкості доставки.

Описано підготовку до розгортання додатку за допомогою контейнеризації в Docker, що спрощує процес установки та запуску програмного забезпечення. Обґрунтовано вибір хмарного провайдера Google Cloud Platform для розміщення додатку та його компонентів.

Порівняно переваги та недоліки ручного управління базою даних та використання сервісу Cloud SQL. Вибрано виділений сервер Compute Engine для запуску контейнеризованого додатку та описано способи збереження файлів на виділеному диску.

Розглянуто процес реєстрації доменного імені та отримання SSL-сертифікатів для захисту передачі даних та безпечної роботи з веб-додатком. Проаналізовано можливості моніторингу інфраструктури та продуктивності додатків з використанням Cloud Logging та Monitoring.

3 ВПРОВАДЖЕННЯ МЕТОДОЛОГІЇ DEVOPS ДЛЯ АВТОМАТИЗАЦІЇ ТА МОНІТОРИНГУ ПРОЕКТУ «ФАЙЛОВИЙ МЕНЕДЖЕР» І АНАЛІЗ РЕЗУЛЬТАТІВ ЇЇ ЗАСТОСУВАННЯ

В цьому розділі автоматизовано процеси тестування та випуску нових версій програмного забезпечення а також його розгортання на виділений хостинг у хмарі. Додано засоби моніторингу стану веб-додатку та термінів оновлення SSL-сертифікатів і доменного імені. В якості проекту буде використовуватися програмний продукт «Файловий менеджер» [41].

3.1 Налаштування етапу постійної інтеграції

Для початку реалізуємо конвеєр для інтеграції нових змін у репозиторій. В лістингу В.4 наведено робочий процес для GitHub Action, який виконає необхідні перевірки – перевірка коду на стилістику та тестування коду модульними тестами.

GitHub Actions це платформа це виконуються завдання на віртуальних машинах, ці завдання запускаються на певну подію, таку як злиття гілки чи відкриття PR, вони описані спеціальним конфігураційним файлом в каталозі .github/workflows. Кожне завдання має певні кроки – скрипти або готові програми з бібліотеки спільноти.

На рисунку 3.1 зображено процес перевірки нових змін коду в репозиторії проекту «Файловий менеджер» на GitHub згідно описаних раніше плавил. Коли конфігураційний файл вперше попадає в репозиторій та на кожну подію описану в ньому, автоматично запускається робочий процес з написаними кроками.

Даний робочий процес описує кроки для виконання лінтування (перевірки коду на відповідність встановленим правилам форматування) у проекті за допомогою GitHub Actions. Результат виконання цього робочого процесу можна побачити на рисунку 3.2.

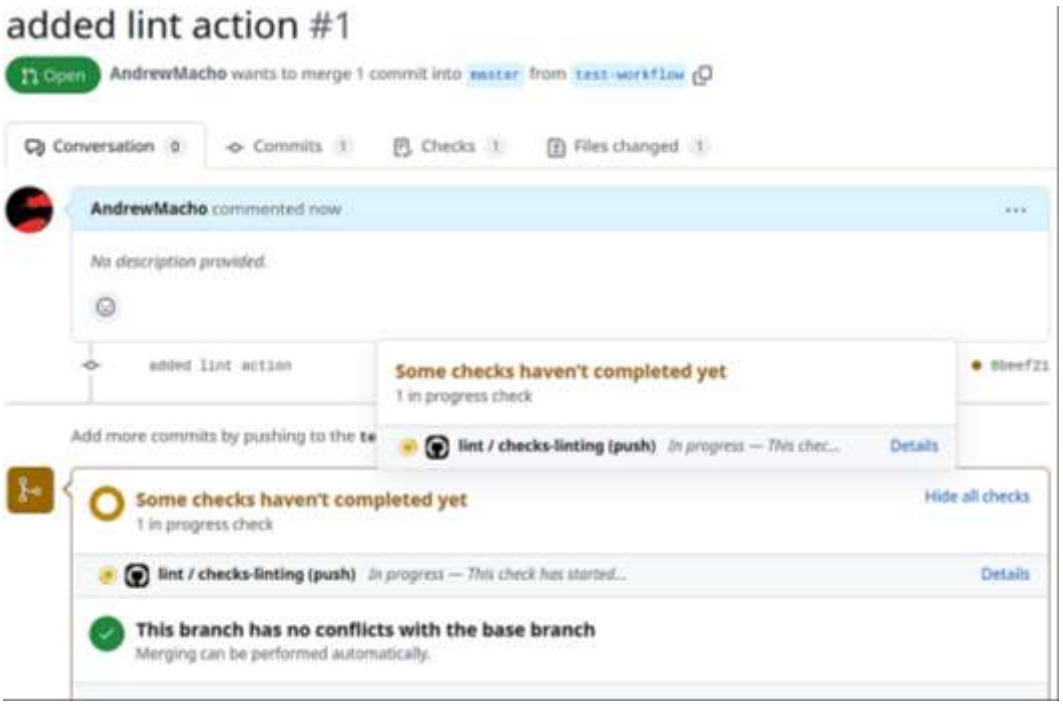

Рисунок 3.1 – Запуск процесу перевірки програмного коду

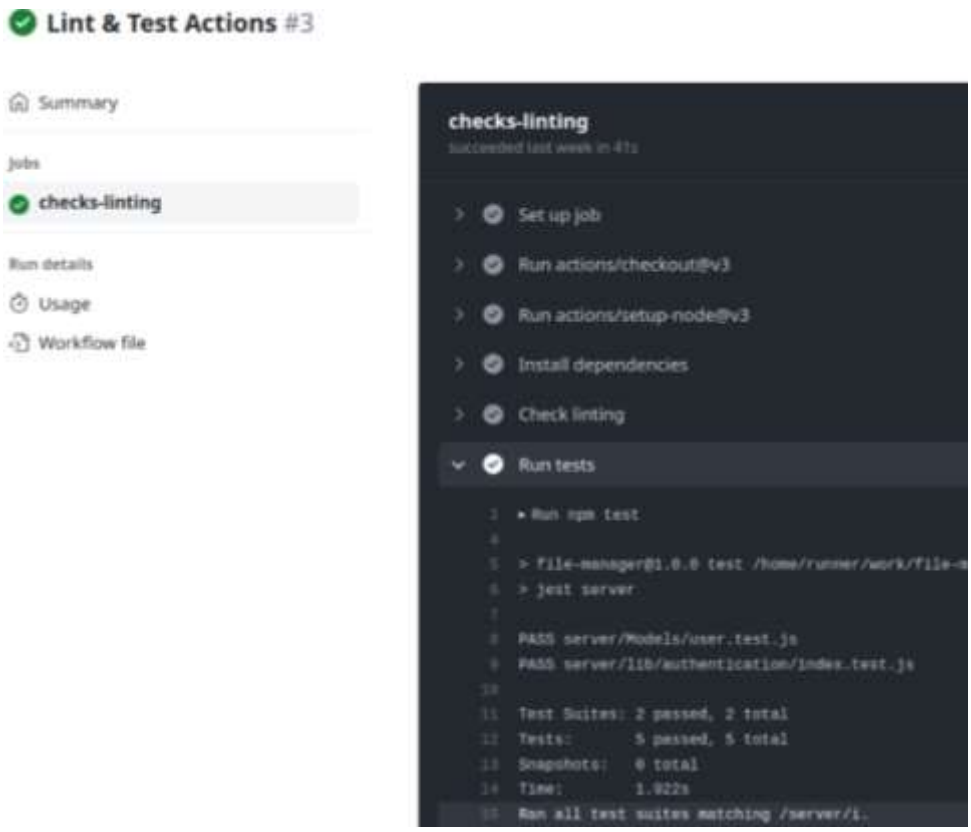

Рисунок 3.2 – Результат виконання перевірки коду перед інтеграцією

Завантаження коду проекту. Робочий процес починається з завантаження коду проекту з репозиторію GitHub за допомогою «actions/checkout@v3».

Налаштування середовища виконання. На цьому етапі налаштовується середовище виконання на Ubuntu з встановленою версією Node.js 14 за допомогою «actions/setup-node@v3».

Встановлення залежностей проекту. Виконується команда «npm ci», яка встановлює всі необхідні залежності проекту відповідно до «package-lock.json».

Перевірка лінтування. На передостанньому етапі виконується команда npm run check, яка запускає скрипт для перевірки відповідності коду проекту встановленим правилам форматування.

Запуск тестів: На останньому етапі запускається виконання тестів командою npm test, яка запускає перевірку коду фреймворком «Jest».

3.2 Налаштування випуску нових версій програмного продукту відповідно до методології CI/CD за допомогою GitHub Actions та Google Cloud Platform

Перед тим як розпочати створювати конвеєр для автоматизації розгортання необхідно створити робоче середовище у хмарному провайдері. Для продовження кроку розгортання має бути створена БД та віртуальна машина для веб-додатку, також треба налаштувати комунікацію між БД та VM.

3.2.1 Створення бази даних в хмарі

У пункті 2.3.1 проаналізовано використання БД в ручному керуванні БД як сервіс. Так як ця БД буде використовуватися для робочого середовища, то вибрано сервіс Cloud SQL на основі MySQL, на рисунку 3.3 показано процес створення БД.

Рисунок 3.3 – Створення БД в Cloud SQL

Однією з переваг БД як сервісу у Cloud SQL в GCP є те, що ви можете легко налаштовувати, підтримувати, керувати та адмініструвати свої реляційні бази даних на платформі Google Cloud. Ви можете використовувати Cloud SQL з MySQL, PostgreSQL або SQL Server.

Ще однією перевагою є те, що Cloud SQL автоматично забезпечує захист даних та резервне копіювання (рис.3.4). Cloud SQL автоматизує створення резервних копій, відновлення даних, шифрування даних у режимі очікування та реплікацію даних. Ви можете налаштувати резервне копіювання за своїм графіком та вибрати місцезнаходження для зберігання резервних копій.

Використовуючи «cloud-sql-proxy» можна підключитися до БД та зі спеціальним ключем, який можна згенерувати на сторінці дашборда БД віртуальної машини, та назвою екземпляра створеної БД, ввівши команду наведену в лістингу 3.1, яка створює проксі:

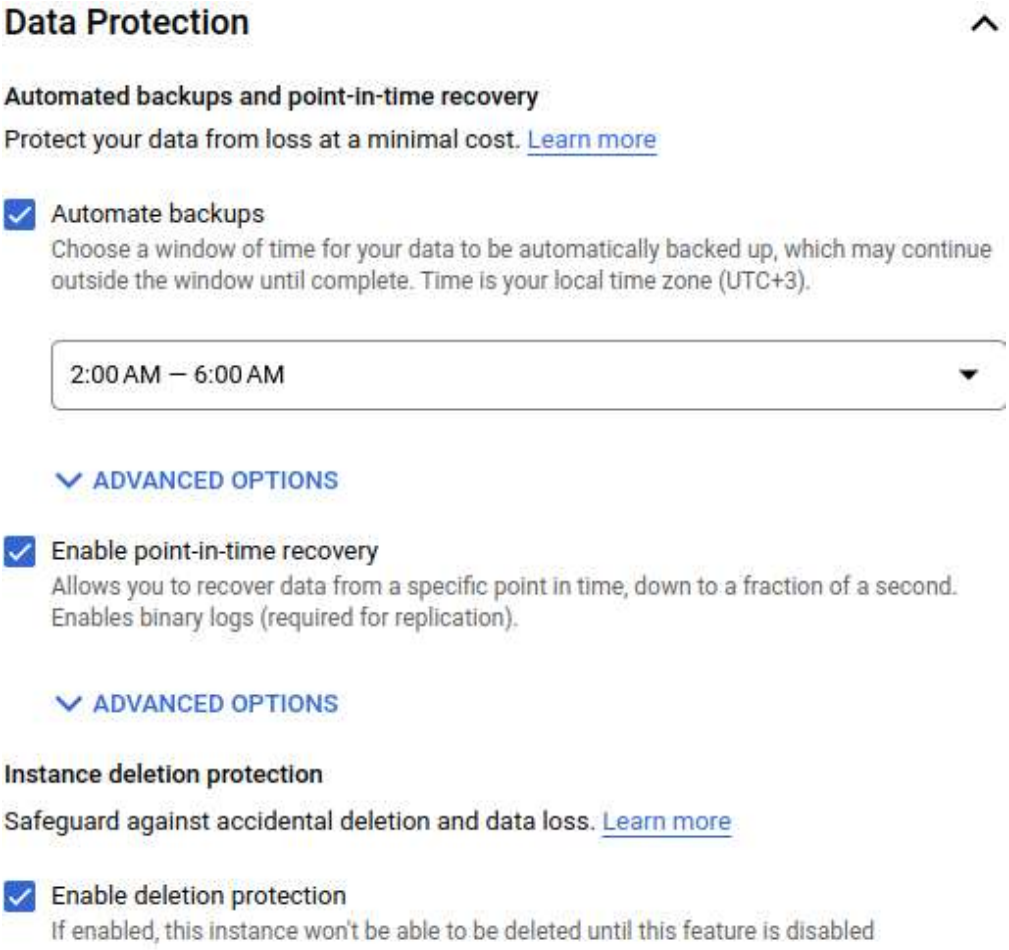

Рисунок 3.4 – Налаштування резервний копій та захисту даних

Лістинг 3.1 – Cloud SQL – підключення до БД через проксі

```
./cloud-sql-proxy -credentials-file ~/Documents/file-manager-application-a8b67b5360f4.json file-manager-application:us-central1:file-manager-db
⬡ system
2023/04/24 15:07:10 Authorizing with the credentials file at "/home/71ndrew/Documents/file-manager-application-a8b67b5360f4.json"
2023/04/24 15:07:11 [file-manager-application:us-central1:file-manager-db] Listening on 127.0.0.1:3306
2023/04/24 15:07:11 The proxy has started successfully and is ready for new connections!
2023/04/24 15:07:26 [file-manager-application:us-central1:file-manager-db] accepted connection from 127.0.0.1:44898
2023/04/24 15:07:36 [file-manager-application:us-central1:file-manager-db] client closed the connection
;^C2023/04/24 15:27:49 SIGINT signal received. Shutting down…
```

Після повідомлення «The proxy has started successfully and is ready for new connections!» можна підключатися до сервера БД локально командою «mysql -u

user_name», що означає що хост знаходиться адресою 127.0.0.1, підключення та перевірка чи база даних створена показано на рисунку 3.5

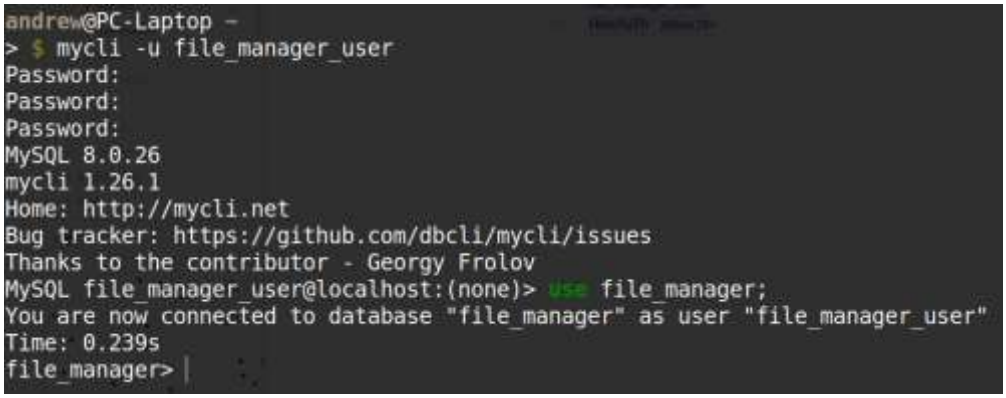

Рисунок 3.5 – Успішна автентифікація через проксі

Як видно на рис. 3.5 підключення до БД успішно встановлено та вона готова до роботи. Після створення віртуальної машини в документації рекомендується налаштувати дозволи до БД. При роботі з віртуальними машинами в GCP цей метод з проксі має спрацювати, проте в документації рекомендують використовувати внутрішню IP-адресу на противагу зовнішній [50], так як буде швидший час відповіді, та більша швидкість передачі даних. Для цього необхідно буде зайти в налаштування БД у вкладку з’єднання, та отримати цю IP-адресу (рис. 3.6).

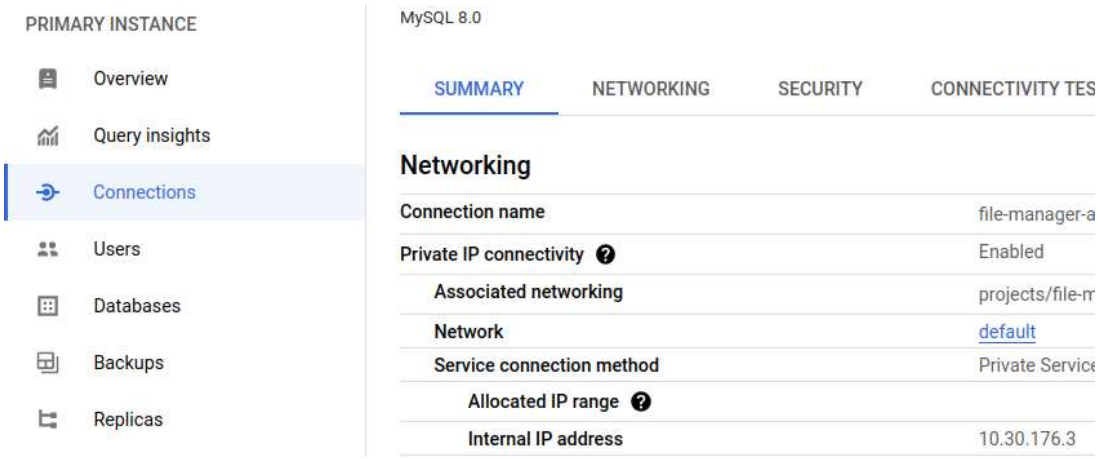

Рисунок 3.6 – Інформація про підключення бази даних

Далі цю інформацію потрібно додати до налаштувань GitHub Action у секцію Secrets and variables, звідки якраз отримає доступ робочий процес для релізів нових версій веб-додатку. Це буде зображено далі в наступних пунктах.

3.2.2 Налаштування провайдера ідентифікації для GitHub Actions

Workload Identity Provider – це сервіс в GCP, який дозволяє автентифікувати та авторизувати робочі навантаження в різних хмарних середовищах. За допомогою Workload Identity Provider ви можете інтегрувати ваші робочі процеси в GitHub Actions з GCP, використовуючи сервісні облікові записи GCP. Це полегшує розгортання та управління вашими додатками в GCP з GitHub Actions [51].

Під час запуску робочого процесу в GitHub Actions для виконання дій з хмарою необхідно автентифікувати запити цього процесу. Для цього використаємо «auth» Action для конфігураційного файлу робочого процесу. Щоб користуватися цим плагіном треба надати два параметри: «workload_identity_provider» та «service_account». В лістингу В.5 наведено команди які створять необхідні сервіси для автентифікації запитів з GitHub Actions.

Представлені команди створюють сервіси, які надають тимчасову автентифікацію для GitHub Actions. Перша команда «gcloud iam workload-identity-pools create» створює пул ідентифікаторів робочого навантаження для проекту, в якому буде запущений веб-додаток. Друга команда «gcloud iam workload-identity-pools providers create-oidc» створює провайдера, який забезпечує взаємодію між веб-додатком та сервісами GCP. Третя команда «gcloud iam service-accounts add-iam-policy-binding» дозволяє додати роль «roles/iam.workloadIdentityUser» для сервісного облікового запису, що дозволить йому отримати доступ до ресурсів GCP, які визначені в пулі ідентифікаторів робочого навантаження.

Ці команди дозволяють створити сервіси, які можуть взаємодіяти з іншими сервісами Google Cloud Platform, використовуючи тимчасові ідентифікатори робочого навантаження. Це забезпечує безпеку та захист даних, оскільки сервіси мають доступ лише до обмеженого набору ресурсів, визначених у пулі ідентифікаторів робочого навантаження. Дані команди є важливим інструментом для розробників, які працюють з веб-додатками та забезпечують безпеку та надійність своїх сервісів.

Для автентифікації було налаштовано сервісний акант «filemanager-service-account@file-manager-application.iam.gserviceaccount.com» та за допомогою пула ідентифікаторів робочого навантаження отримано дані для надання індентифікації «projects/1006422617706/locations/global /workloadIdentityPools/ file-manager-pool/*»», які передамо у змінні середовища для GitHub Actions – «service_account» та «workload_identity_provider».

3.2.3 Створення віртуальної машини для веб-додатку та репозиторію для Docker-контейнерів

Останнім кроком перед створенням робочого процесу в GitHub Actions для постійного розгортання буде створення віртуальної машини та репозиторію для Docker-контейнерів. Після звернення до документації обрано найоптимальніший варіант створення віртуальної машини зважаючи на те що «Файловий менеджер» вже має Dockerfile – створення екземпляра віртуальної машини за допомогою команди «create-with-container», яка використовує ОС оптимізовану для контейнерів (див. лістинг В.6).

В лістингу В.6 наведена команда створення нового віртуального сервера з контейнером на GCP з використанням сервісу Compute Engine. Команда починається зі зазначення назви створюваного інстансу «file-manager-app-instance».

Наступний параметр команди «--container-image» вказує Docker-образ, який буде використовуватися для запуску контейнера на новому інстансі. Це

відбувається за допомогою пакетного репозиторію, який знаходиться на Google Cloud Platform.

Параметр «--disk» дозволяє створити диск, який буде доданий до створюваного інстансу та буде монтуватися на контейнер. Диск названий «file-manager-disk» та має режим «read-write» (mode=rw), його необхідно використовувати в ручному режимі, авто-видалення не ввімкнуто (auto-delete=no).

Нарешті, параметр «--zone» вказує зону, в якій необхідно створити новий інстанс. У цьому випадку екземпляр буде створений в зоні us-central1-c.

Таким чином, ця команда створює новий віртуальний сервер з контейнером, який зберігається в репозиторії на Google Cloud Platform та монтує диск, що дозволяє зберігати файли користувачів.

Перевіримо доступність сервера перейшовши за його зовнішньою IP-адресою, яка видається віртуальним машинам (рис. 3.7).

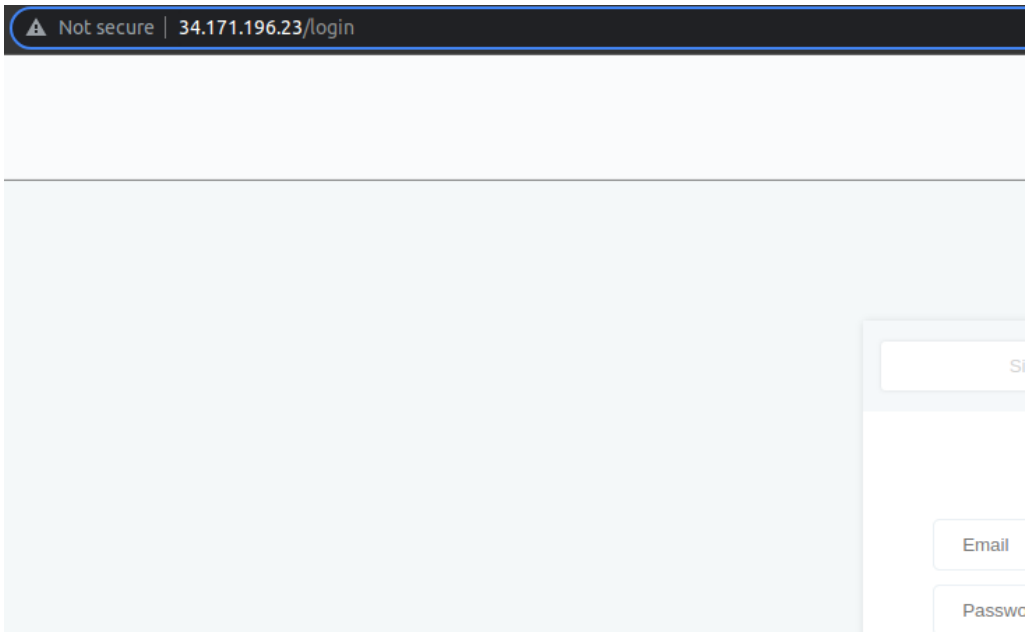

Рисунок 3.7 – Веб-додаток «Файловий менеджер» успішно запущено

Ця IP-адреса не є статичною, і може змінюватися при перезапуску VM, тому в подальному при додаванні домену, необхідно буде зарезервувати цю IP-адресу.

Створення репозиторію дозволяє зберігати та управляти Docker-образами в зручний спосіб, що спрощує розгортання та управління веб-додатками на основі Docker-контейнерів.

Щоб мати змогу створити та оновлювати віртуальну машину новими версіями докеризованого веб-додатку необхідно створити docker-репозиторій в сервісі Artifact Registry, команда створення репозиторію наведена в лістингу В7. У цьому лістингу ми маємо команду для створення репозиторію з назвою «file-manager-repository» в сервісі Google Artifact Registry. Опції команди передають необхідну інформацію про цей репозиторій, таку як формат репозиторію (docker) та розташування (us-central1).

Але після тестового збирання образу та спроби його відправити у docker-репозиторій виникла помилка доступу з повідомленням «Permission denied on on resource “projects/file-manager…”». Щоб усунути цю помилку треба впевнитися що було правильно вказано шлях до репозиторію, а також надати необхідні права для сервіс-акаунта, команда в хмарному терміналі подана в лістингу В.8 в ній оновлено доступ акаунта та надано роль для роботи з Artifact Repository. В майбутньому необхідно буде повторювати цю процедуру для інших сервісів, наприклад, для створення екземплярів віртуальних машин.

У лістингу В.8 показана команда для оновлення політик безпеки доступу до Artifact Registry. Команда виконує додавання нового правила доступу для сервісного аккаунта з іменем «filemanager-service-account@file-manager-application.iam.gserviceaccount.com». При виконанні цієї команди, роль «roles/artifactregistry.repoAdmin» надається сервісному аккаунту. Це дозволяє сервісному аккаунту отримувати права на адміністрування репозиторіїв у Artifact Registry.

Команда використовує ідентифікатор проекту «file-manager-application» для знаходження проекту, до якого буде додано нове правило доступу. Це забезпечує можливість керування доступом до ресурсів з одного місця та спрощує процес управління політиками безпеки.

Ця команда допомагає забезпечити безпеку даних та виконання належних стандартів безпеки в процесі розробки та розгортання веб-додатків, які використовують Artifact Registry.

Останнім кроком перед створенням робочого процесу в GitHub Actions буде налаштування змінних оточеня, є два види змінних для робочих процесів в сервісі – variables та secrets [52], що зображено на рисунку 3.8.

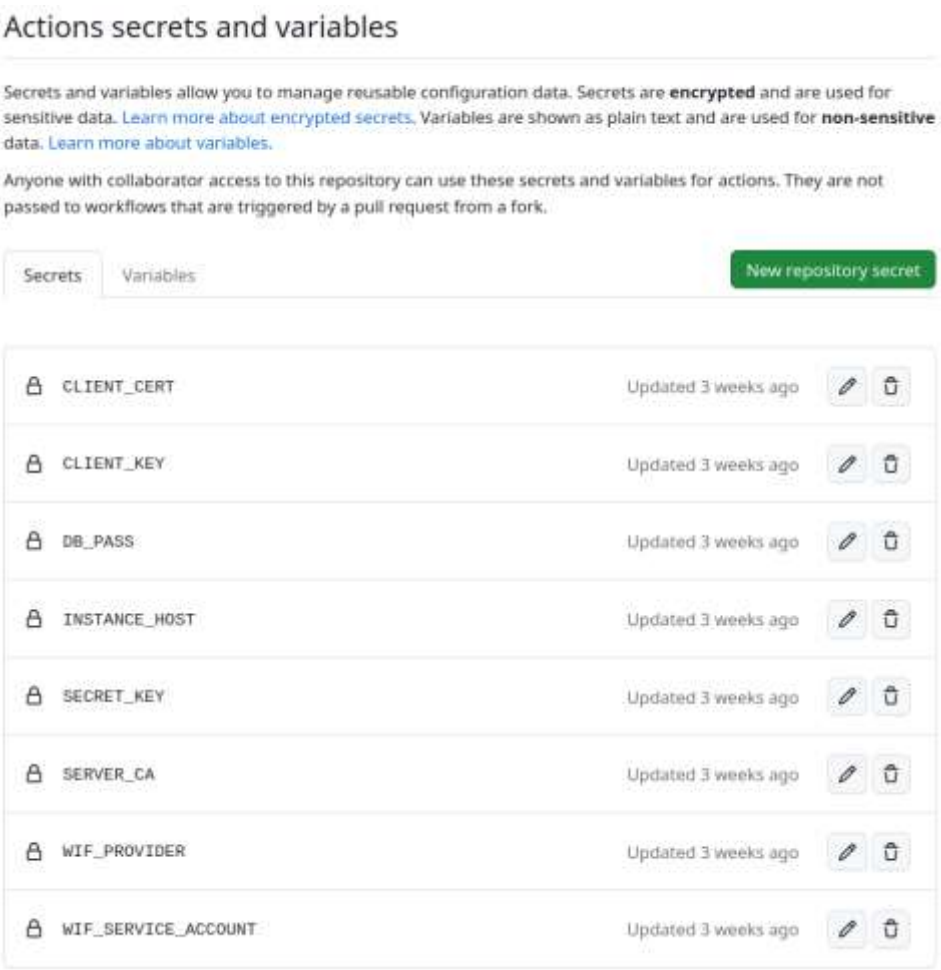

Рисунок 3.8 – Шифровані та звичайні змінні середовища в GitHub Actions

Variables визначають змінні оточення, які можуть бути використані в робочому процесі. В даному випадку вказані змінні оточення для підключення до бази даних та хостів

Secrets представляють конфіденційні дані, які використовуються в робочому процесі. Вони зберігаються в зашифрованому вигляді і можуть бути використані лише в межах робочого процесу. У даному випадку вказані

конфіденційні дані для підключення до бази даних, а також сертифікати для безпечного зв'язку

Отже, основна відмінність між variables та secrets полягає в тому, що variables представляють загальнодоступні змінні оточення, які можуть бути використані в робочому процесі, тоді як secrets представляють конфіденційні дані, які не можуть бути загальнодоступні і використовуються для безпечного зв'язку з іншими сервісами або системами.

3.2.4 Створення конвеєра в GitHub Actions

Створення конвеєра в GitHub Actions передбачає створення робочих процесів, які складаються з окремих кроків, необхідних для виконання певної задачі. Пошук доступних робочих процесів здійснюється в розділі Actions репозиторію на GitHub (рис 3.9). Для створення нового робочого процесу, можна вибрати один з наявних шаблонів або створити власний процес з нуля.

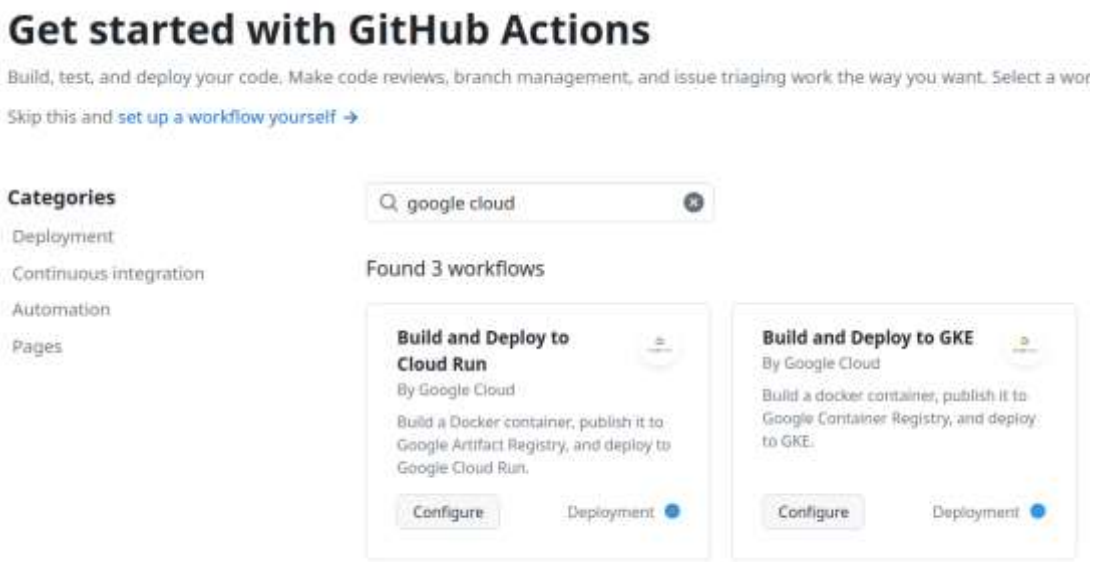

Рисунок 3.9 – Пошук шаблонів робочих процесів

В пошуку було знайдено та вибрано шаблон для розгортання контейнеризованих додатків для сервісу Cloud Run. У випадку з веб-додатком

«Файловий менеджер» на Compute Engine необхідно замінити відповідні команди і скористатися командою «update-container» для оновлення образу контейнера з останньою версією веб-додатку. Це дозволяє забезпечити, що веб-додаток запущений на віртуальній машині, завжди містить останні зміни.

Таким чином створено конфігураційний файл для розгортання нових версій ПЗ з використанням шаблону в бібліотеці (лістинг В.9)

Цей файл містить конфігурацію робочого процесу в GitHub Actions з назвою «deploy». Процес буде запускатися при додаванні нових змін (push) в гілки «master» і «deploy-workflow».

Спочатку відбувається авторизація користувача в сервісі Google Cloud і Docker Hub за допомогою ключів доступу, які зберігаються у вигляді секретів в налаштуваннях репозиторію. Далі виконується збірка Docker-образу і його публікація в Docker-реєстрі на Google Cloud.

Останнім кроком є розгортання оновленої версії додатку на віртуальній машині Google Compute Engine, яке виконується за допомогою команди «gcloud compute instances update-container». В якості параметрів вказуються назва віртуальної машини, зона розташування та ідентифікатор оновленого Docker-образу, який був завантажений до Docker-реєстру на попередньому кроці.

Згодом, після декількох запусків цього робочого процесу було виявлено що він закінчився з помилкою, проаналізувавши журнал виконання цього процесу було визначено що віртуальна машина не має вільного дискового простору. Для вирішення цієї проблемо потрібно очищувати старі Docker-образи та контейнери, які вже не є актуальними, тому було додано спеціальний скрипт при завантаження віртуальної машини (це відбувається при кожному релізі нової версії), який видаляє ці старі артефакти, команду наведено в лістингу 3.2.

.

Лістинг 3.2 – Додавання стартового скрипта для звільненння місця

```
macho0863@cloudshell:~ (file-manager-application)$ gcloud compute instances add-metadata file-manager-app-instance --zone=us-central1-c --metadata=startup-script='#! /bin/bash
  docker system prune -af
EOF'
```

У приведеному прикладі виконується команда для додавання стартового скрипта на віртуальну машину з назвою «file-manager-app-instance». Стартовий скрипт виконує команду «docker system prune -af», яка звільнює місце на віртуальній машині, що використовується для розгортання контейнерів. Команда «gcloud compute instances add-metadata» використовується для додавання метаданих до віртуальної машини. В параметрі «--zone» вказується зона, в якій розташована віртуальна машина. Параметр «–metadata» містить стартовий скрипт, який виконується при запуску віртуальної машини. Результат виконання цього робочого процесу наведено на рисунку 3.10.

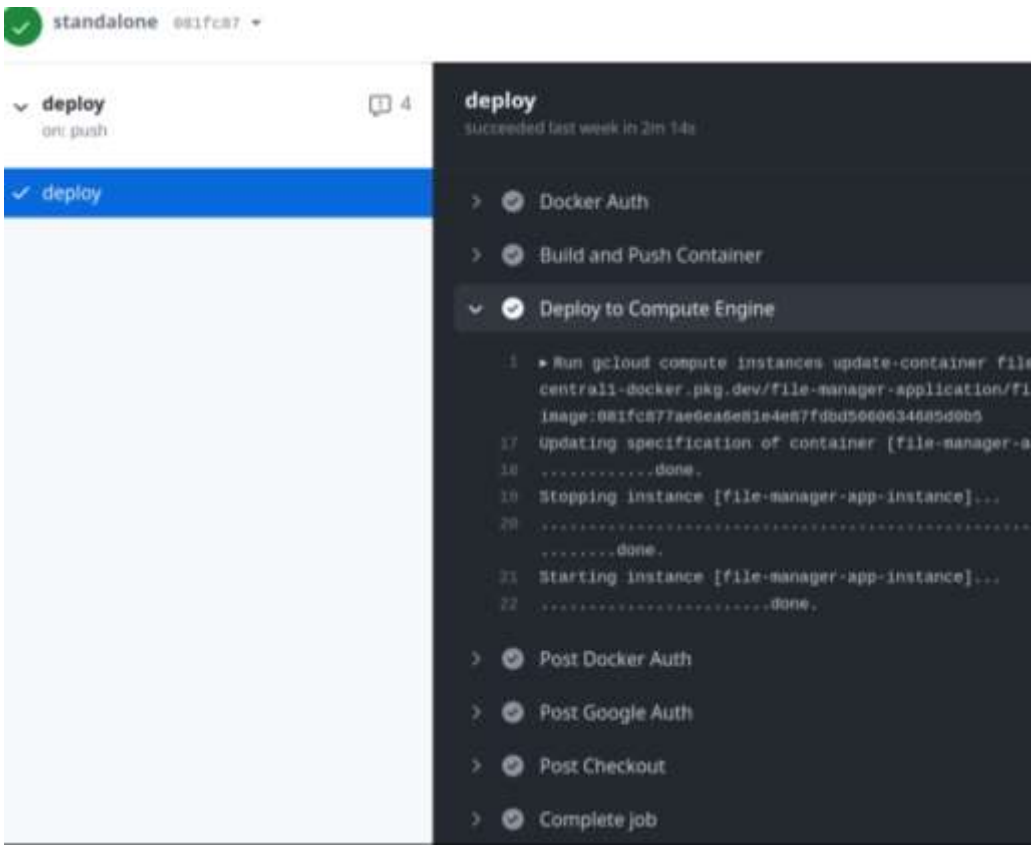

Рисунок 3.10 – Результат виконання робочого процесу в GitHub Actions

Після додавання змін до репозиторію в гілку «deploy-workflow» запустився робочий процес «deploy» на платформі GitHub Actions. Він автоматично зібрав та опублікував нову версію Docker-образу в Docker-реєстрі на Google Cloud, та розгорнув оновлену версію додатку на віртуальній машині Google Compute Engine. Цей робочий процес дозволяє внести зміни в додаток та оновлювати

його, не потребуючи вручну здійснювати процеси збірки, публікації та розгортання.

3.3 Налаштування доменного імені та SSL сертифікатів

Для налаштування доменного імені веб-додатку «Файловий менеджер» потрібно оновити DNS записи типу «A» у провайдера доменів та вказати в них IP-адресу сервера на який спрямовувати при запиті на домен mod-man.online. Проте спочатку необхідно зарезервувати статичну IP-адресу (рис. 3.11), так як адреса віртуальної машини динамічна, а система доменних імен оновлює записи рідко, необхідно мати незмінну IP-адресу.

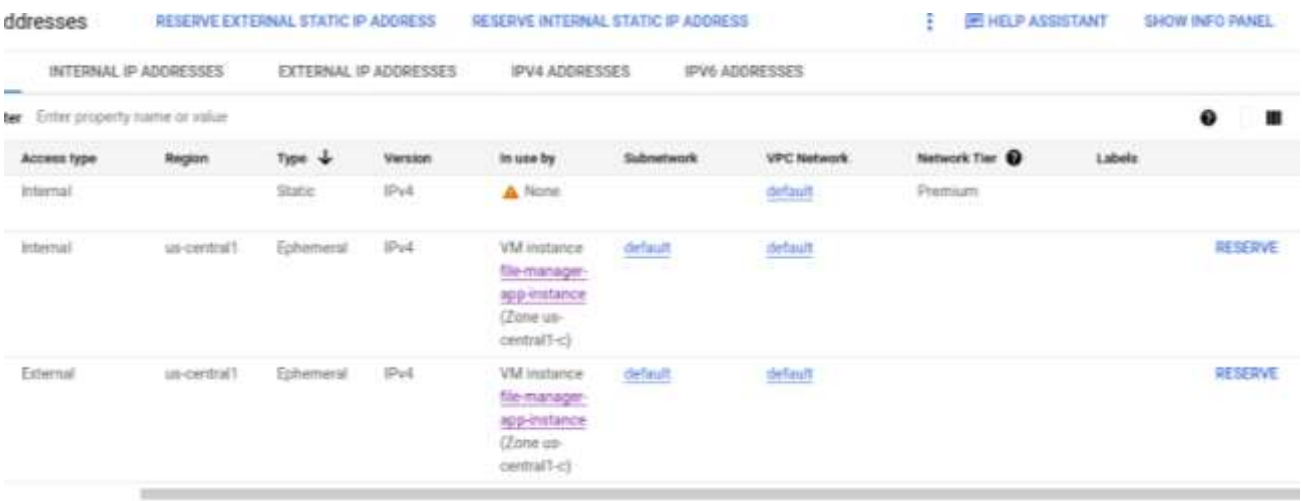

Рисунок 3.11 – Резервування IP адреси для віртуальної машини

Щоб зарезервувати необхідно натиснути відповідну кнопку – RESERVE. Після цього перевіримо статус IP-адреси (рис. 3.12).

Оновити записи DNS типу «A» можна зробити в налаштуваннях дашборда на сайті провайдера, на рисунку 3.13, зображено оновлені записи домена.

Рисунок 3.12 – IP-адреса отримала статус статичної

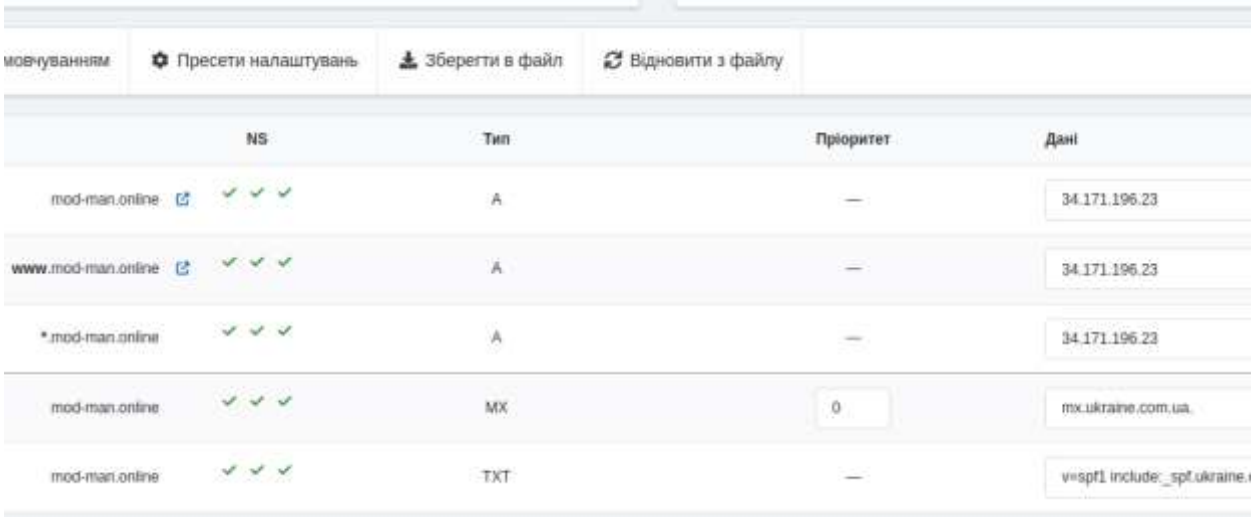

Рисунок 3.13 – Доменне ім’я спрямовує запити на сервер із веб-додатком

Щоб забезпечити безпеку сайту, рекомендується встановити SSL-сертифікати. SSL (Secure Sockets Layer) – це протокол шифрування, який захищає дані, що передаються між сервером і браузером. Це забезпечує безпеку передачі даних, запобігає крадіжці даних, включаючи особисту інформацію користувачів, паролі, платіжну інформацію тощо.

SSL-сертифікати встановлюються на веб-сервері і дозволяють встановити захищене з'єднання між сервером і браузером за допомогою протоколу HTTPS (HTTP over SSL). HTTPS забезпечує захист персональної інформації користувачів і знижує ризик крадіжки даних під час їх передачі по мережі.

Для використання сертифікатів у проекті необхідно створити https сервер в кореневому файлі сервера node.js. Сертифікати підключаються у вигляді файлів, що подано в лістингу В.10.

Модифікуємо логіку створення сервера, додамо також https сервер з переданими до нього параметрами шляхів до сертифікату

Самі сертифікати генеруємо за допомогою команди наведеної в лістингу 3.3. Certbot автоматично згенерує сертифікат і пройде необхідні перевірки щодо власності домена. Увагу слід звернути на параметри «-standalone» та «--post-hook ./scripts/start.sh», перший запускає certbot’a в режиму сервера, а другий параметр дозволяє запустити скрипт по успішному закінченні генерації сертифіката – запуск сервера «Файлового менеджера».

Лістинг 3.3 – Запуск certbot для підтвердження власності домена та генерації SSL-сертифікатів

```
certbot certonly --standalone -d mod-man.online --non-interactive --agree-tos --email macho0863@gmail.com --post-hook ./scripts/start.sh
```

Цю команду додано до Dockerfile як останню в списку, таким чином при кожному запуску веб-сервісу буде спроба генерувати нові сертифікати, якщо термін придатності сплине – створяться нові сертифікати.

Після успішної генерації сертифікатів запускається веб-сервер. На рисунках 3.14 та 3.15 показано що було використано безпечне з’єднання HTTPS.

Коли на веб-сайті встановлений SSL-сертифікат і з'єднання з ним відбувається за допомогою HTTPS, то в браузері з'являється індикатор безпечного з'єднання. Це може бути зображення замка або зеленої іконки з надписом "Secure" чи "Secure Connection". Крім того, адреса сайту починається з «https://» замість звичайного «http://».

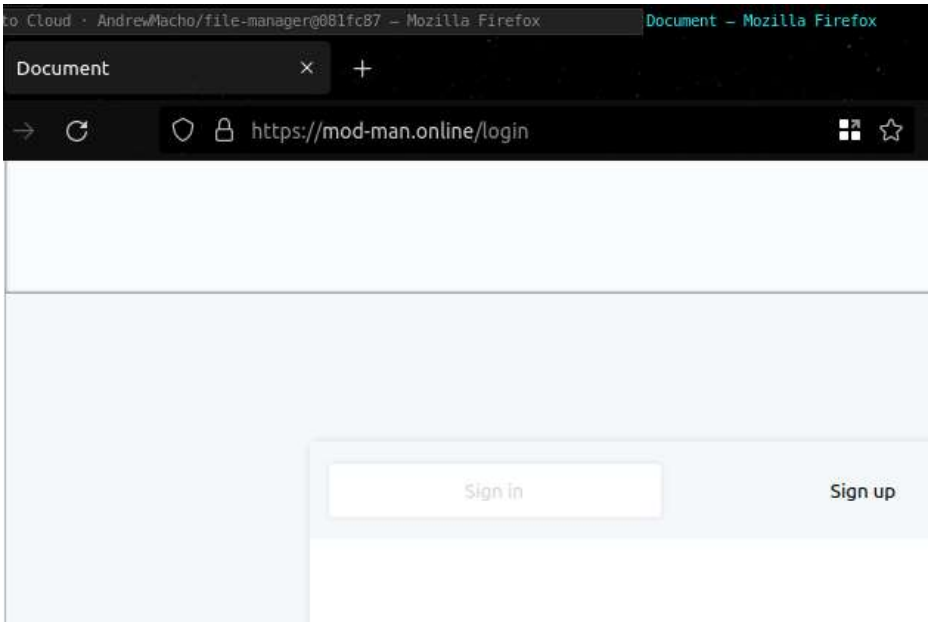

Рисунок 3.14 – Знак замка та протокол HTTPS в URL

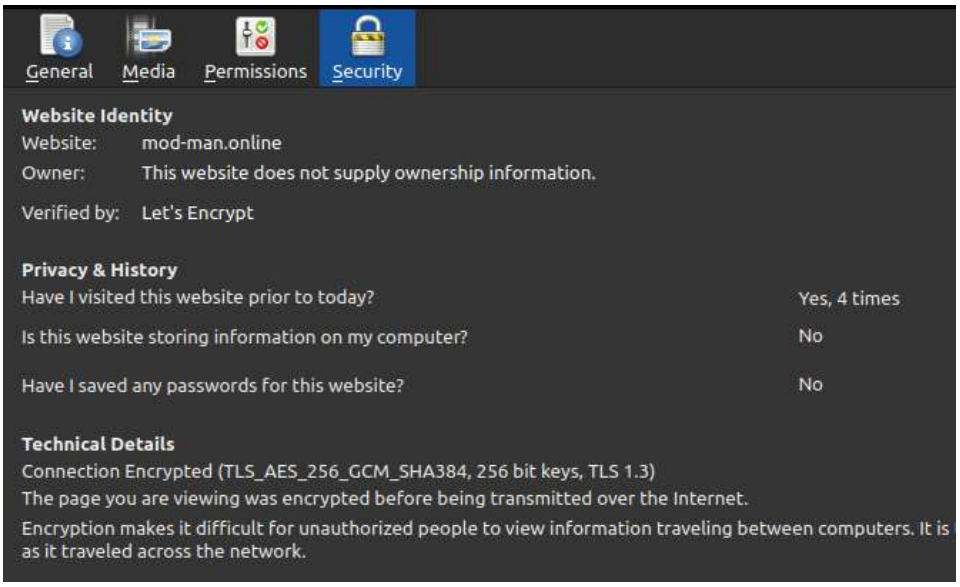

Рисунок 3.15 – Детальна інфомація про безпеку на веб-сайті mod-man.online

При кліку на індикатор безпечного з'єднання в браузері відображається додаткова інформація про сертифікат, включаючи інформацію про власника, видавця та термін дії. Це дає користувачеві можливість перевірити, що він з'єднується з правильним веб-сайтом і має дійсний сертифікат безпеки.

3.4 Налаштування моніторингу інфраструктури та продуктивності веб-додатку

Агент журналювання передає журнали з віртуальних машин та вибраних сторонніх пакетів програмного забезпечення до Cloud Logging. Найкраще запускати агент журналювання на всіх екземплярах віртуальної машини.

Образи віртуальних машин для Compute Engine і Amazon Elastic Compute Cloud (EC2) не містять агента журналювання, тому необхідно виконати ці кроки, щоб встановити його в таких екземплярах.

Якщо віртуальні машини працюють в Google Kubernetes Engine або App Engine, агент вже включений в образ віртуальної машини. Ви можете встановити агент на окрему віртуальну машину Linux за допомогою консолі Google Cloud зі сторінок Cloud Monitoring або Compute Engine.

- У консолі Google Cloud виберіть свій проект Google Cloud.
- В області переходів виберіть елемент «Моніторинг».
- В області переходів Моніторинг виберіть пункт «Дашборд».
- У таблиці дашбордів знайдіть запис «Екземпляри віртуальної машини», а тоді клікніть на назві.

В списку на вкладці «Інвентаризація» на інформаційній панелі перелено всі віртуальні машини та містить стовпець стану для вашого агента (рис. 3.16).

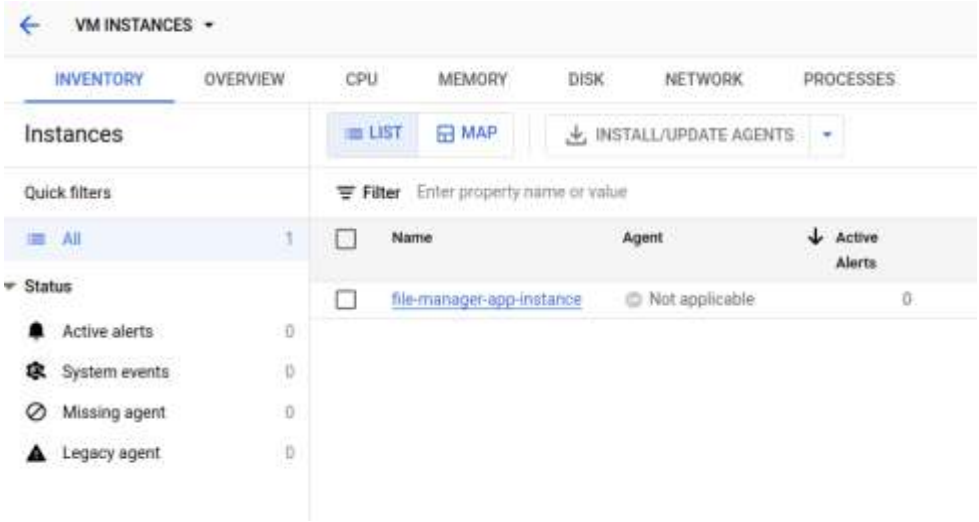

Рисунок 3.16 – Список доступних віртуальних машин

Стовпець Агент повідомляє про такі значення:

- Не виявлено: агент не встановлено. Якщо Cloud Monitoring не виявляє жодних агентів, встановлених у екземплярі Compute Engine, ви можете встановити агент.
- Ops Agent: Ви запускаєте Ops Agent. Якщо ви не бачите зеленої галочки поруч із записом, це означає, що доступне оновлення агента на основі виявленої операційної системи вашої віртуальної машини. При наведенні курсору на індикатор Ops Agent в таблиці ви бачите інформацію про версію Ops Agent. Якщо ви використовуєте старішу версію, ви також побачите рекомендацію оновити свого агента.
- Очікує на розгляд: Ops Agent встановлюється або оновлюється.
- Застарілий агент: Ви запускаєте застарілий моніторинг або агент журналювання.
- Не застосовується: Ця віртуальна машина не є підтримуваною платформою для запуску агента.
- Невідомо: віртуальна машина не працює, тому статус агента невідомий.

У випадку веб-сервісу «Файловий менеджер» в стовпці «Агент» вказано «Not applicable», проте якщо навести на цей елемент то з’явиться повідомлення: «Віртуальні машини Anthos/GKE та Dataproc керуються Google і включають вбудовані агенти».

Це означає, що веб-сервіс "Файловий менеджер" не потребує окремого агента для моніторингу інфраструктури або додатку. Замість цього, Google надає вбудовані агенти для віртуальних машин, які працюють в Anthos/GKE та Dataproc. Ці агенти дозволяють збирати різноманітні метрики та логи з віртуальних машин та інших складових інфраструктури, і передавати їх до сервісів моніторингу та логування Google, таких як Cloud Monitoring та Cloud Logging.

Отже, веб-сервіс «Файловий менеджер» автоматично підключається до вбудованих агентів для моніторингу та логування, що дозволяє забезпечити ефективний моніторинг та аналіз проблем в інфраструктурі та додатку. Щоб

впевнитися що він інстальований на VM, підключимося до цієї машини через SSH, та подивимося які docker-контейнери запущені (лістинг В.11).

Цей лістинг показує результат виконання команди «docker ps» на Docker-оптимізованій операційній системі, яка працює на віртуальній машині з назвою "file-manager-app-instance". Рядок CONTAINER ID IMAGE STATUS PORTS NAMES показує заголовки стовпців у виводі.

У виводі відображаються два контейнери Docker, що запущені на цій операційній системі:

Контейнер з ідентифікатором «8f7b014bbf8b», який містить файловий менеджер.

Контейнер з ідентифікатором «df216a413cde», який містить агента журналювання Stackdriver Logging Agent для Docker та запущений з образу «gcr.io/stackdriver-agents/stackdriver-logging-agent:1.9.9».

Рисунок 3.17 відображає список метрик, які доступні для моніторингу веб-додатків у Google Cloud Platform (GCP). На ньому можна побачити різні метрики, такі як CPU використання, мережевий трафік, навантаження на диск, кількість запитів на секунду, а також інші метрики, які дозволяють зрозуміти, як працює веб-додаток в GCP.

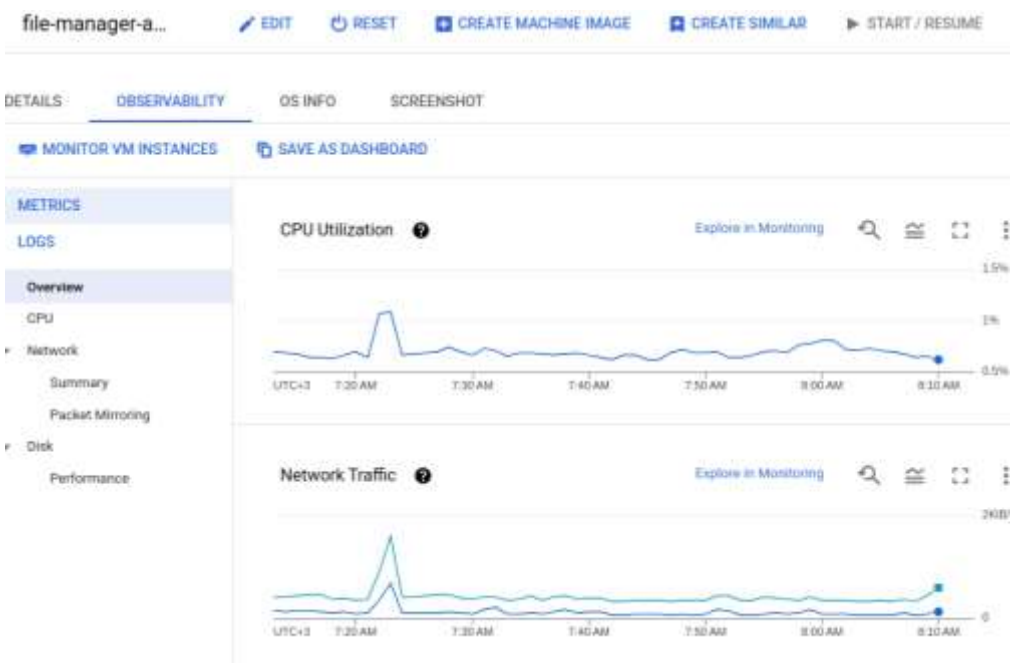

Рисунок 3.17 – Метрики для веб-додатку в GCP

CPU використання моніторить кількість процесорного часу, який використовується веб-застосунком. Мережевий трафік відображає обсяг даних, які передаються між веб-додатком і клієнтами. Навантаження на диск відображає кількість дискової активності, яка здійснюється веб-додатком. Кількість запитів на секунду моніторить кількість запитів, які обробляються веб-додатком за певний період часу. Середній час відповіді сервера відображає час, який потрібно веб-застосунку для обробки запиту і надсилання відповіді.

Ці метрики є корисним інструментом для моніторингу веб-додатків в GCP, оскільки дозволяють виявляти проблеми та вдосконалювати продуктивність веб-додатків, забезпечуючи кращий досвід для користувачів.

3.5 Аналіз результатів застосування DevOps методологій для веб-додатку «Файловий менеджер»

3.5.1 Аналіз результатів застосування методології CI/CD

Після реалізації практики постійної інтеграції та розгортання було виміряно основні ключові метрики – кількість релізів новий версій за проміжок часу, кількість помилок при злитті в основну гілку, а також проаналізовано журнали

В рамках експерименту було проведено порівняння ефективності DevOps CI/CD практик зі звичайним деплоєм. Для цього було взято два ідентичні веб-додатки на Node.js з базою даних MySQL, які були запущені на окремих віртуальних машинах в хмарному сервісі.

Вимірювання проводилося на основі часу проведеного для релізу нової версії в робоче середовище, а також кількістю дій необхідних для виконання процедури. Загалом, кількість дій, необхідних для звичайного деплою, може відрізнятися в залежності від проекту та конфігурації. Для «Файлового менеджера» було виконано наступні дії:

1. Вхід в систему за допомогою SSH.

2. Отримання нових змін з репозиторію.

3. Налаштування змінних середовища (опційно, при оновленнях змінних).

4. Запуск перевірок та тестів, які необхідні для переконання, що нова версія працює належним чином.

5. Зупинка старої версії.

6. Запуск нової версії, яку було отримано з репозиторію.

7. Очищення залишкових файлів, які більше не використовуються.

8. Вихід з системи.

Детальний огляд процедури ручного розгортання наведено в лістингу В.12 Після входу на сервер через SSH фіксувався час початку процедури розгортання командою «date» (рис 3.18) і після виконання усіх процедур також фіксувався час завершення :

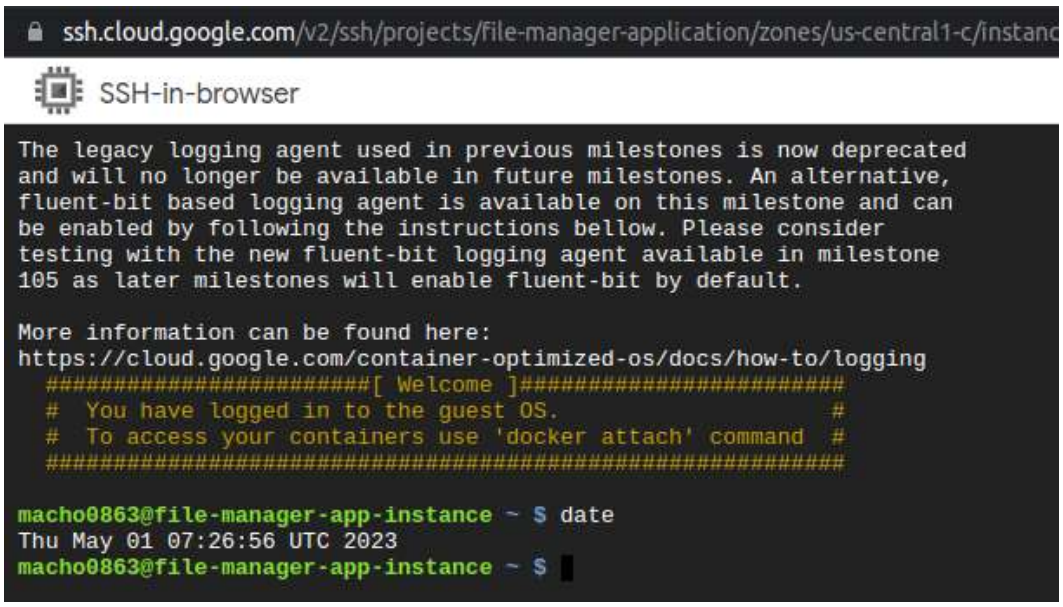

Рисунок 3.18 – Консоль входу на SSH-сервер та фіксування часу початку процедури розгортання

Для розгортання із застосуванням DevOps CI/CD практик було налаштовано автоматизований процес Continuous Integration та Continuous Deployment з використанням хмарної CI/CD платформи GitHub Actions. Цей процес включав наступні кроки:

1. Кожний коміт до репозиторію на GitHub запускає процес Continuous Integration, який виконує наступні дії:

- Клонує останню версію коду з репозиторію.
- Запускає перевірки та модульні тести на відповідність стандартам та вимогам.

2. Кожне оновлення основної гілки запускає процес Continuous Deployment, який виконує наступні дії:

- Якщо всі тести успішні, то тоді збирається новий Docker-образ з додатком та завантажується в Docker-реєстр на хмарному сервісі.
- Завантажує оновлений Docker-образ з Docker-реєстру для веб-додатку.
- Зупиняє старий контейнер з додатком.
- Запускає новий контейнер з додатком на базі оновленого Docker-образу.

Оновлення веб-додатку було виконано за 5 хвилин (3 хв розгортання та 2хв запуск процесів на віртуальній машині) з використанням DevOps CI/CD практик (рис. 3.19), тоді як звичайний деплой зайняв 30 хвилин. Крім того, DevOps CI/CD практики забезпечують більшу стабільність веб-додатку та зменшують його вразливість до атак.

Рисунок 3.19 – Тривалість релізу нової версії в робочому процесі

В результаті експериментів було виміряно час, який потрібен для оновлення веб-додатку та введення його в експлуатацію в обох випадках. Вимірювання показали, що DevOps CI/CD практики дають значно швидший та безпечніший процес розгортання веб-додатку в порівнянні зі звичайним деплоєм.

Нижче наведена таблиця 3.1, в якій порівняно звичайний деплой (ручний) та CI/CD (автоматичний):

Таблиця 3.1 – Порівняльна характеристика ручного та автоматичного розгортання

Характеристика	**Звичайне розгортання**	**CI/CD**
Час виконання	10-20 хв	3-5 хв
Кількість необхідних дій	Залежить від складності проекту та конфігурації (8 кроків у випадку файлового менеджера	Залежить від складності проекту та конфігурації, але зазвичай менше, ніж у звичайного деплою
Автоматизований процес	Ні	Так
Потенційні помилки	Часто трапляються	Менш імовірні
Стабільність	Може бути менш стабільним через ручні помилки	Зазвичай більш стабільний через автоматичний процес

Також було визначено, що DevOps CI/CD практики дозволяють підвищувати швидкість відповіді на запити щодо розробки функціоналу та забезпечують більшу доступність веб-додатка для користувачів. В цілому, результати експерименту показали, що використання DevOps CI/CD практик є важливою складовою для підвищення ефективності розгортання веб-додатків та поліпшення їхньої якості.

3.5.2 Виявлення наявних та потенційних проблем функціонування веб-додатку за допомогою інструментів моніторингу використання ресурсів

Також практики моніторингу надають важливу статистичну інформацію, яка може виявити слабкі місця у веб-додатку.

Для аналізу проблем веб-додатку, пов'язаних з використанням ресурсів, було використано Cloud Monitoring. Cloud Monitoring, який надає можливість моніторити та відстежувати різноманітні метрики та показники продуктивності в середовищі хмарних обчислень.

Для аналізу проблем веб-додатку використовувалися наступні функціональності та можливості Cloud Monitoring:

- Моніторинг використання ресурсів. Cloud Monitoring надає засоби для моніторингу використання ресурсів веб-додатку, таких як CPU, пам'ять, мережа та сховище даних.
- Створення метрик та показників продуктивності. За допомогою Cloud Monitoring можна налаштовувати метрики та показники продуктивності, які відображають стан веб-додатку.
- Створення сповіщень та сповіщення про проблеми. Cloud Monitoring дозволяє налаштовувати сповіщення та отримувати повідомлення про виникнення проблем або досягнення заданих порогів.
- Аналіз та візуалізація даних: Cloud Monitoring надає можливість аналізувати та візуалізувати дані про метрики та показники продуктивності за допомогою графіків, діаграм та інших візуальних засобів.

Загалом, за допомогою Cloud Monitoring можна виконувати постійний моніторинг використання ресурсів веб-додатку та вчасно виявляти проблеми, що дозволяє ефективно управляти та оптимізувати продуктивність системи.

У даному випадку, за допомогою моніторингу диску для файлів було виявлено, що операції над файлами навантажують диск і це є слабким місцем. Всього лиш для декількох користувачів використання диску на запис досягало 70-80 Мб/с (рис. 3.20), при збільшенні користувачів цей показник зросте до

максимально доступних показників та швидкість запису буде сповільнюватися через паралельні записи, тому необхідно буде удосконалити архітектуру.

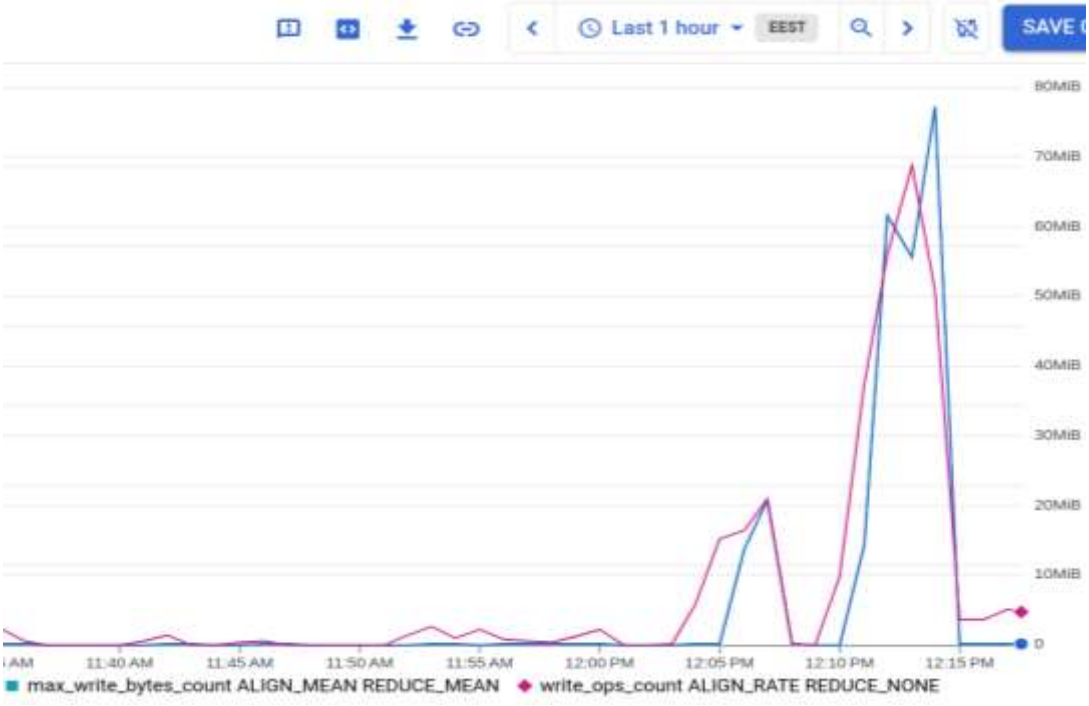

Рисунок 3.20 – Використання диску при записі файлів

Також під час аналізу інших показників, таких як використання CPU бази даних (рис. 3.21), кількість запитів до MySQL (рис. 3.22) та читання/запису (рис. 3.23), було виявлено, що ці показники знаходяться в нормальних межах і не вказують на проблеми з архітектурою або ресурсами віртуальної машини та бази даних. Таким чином, під час аналізу було виявлено, що тільки диск потребує уваги в покращенні архітектури чи потужностей і він може бути обмежуючим фактором або вузьким місцем.

У випадку зі збільшенням використання диску на запис, моніторинг допоміг виявити цю проблему та дав розробникам чітку інформацію щодо використання ресурсів та можливих причин проблеми. З цієї інформації, команда розробників прийняла рішення щодо необхідних змін в архітектурі, щоб оптимізувати роботу з файловою системою та додати необхідну масштабованість для подальшого зростання обсягів даних.

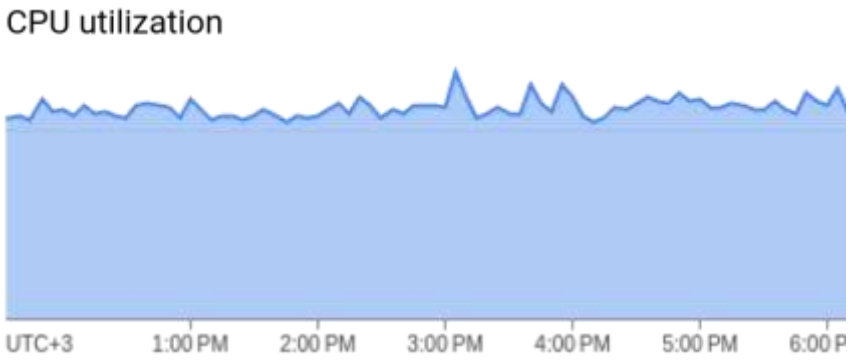

Рисунок 3.21 – Навантаження процесора при зверненні до БД

Рисунок 3.22 – Кількість запитів до БД

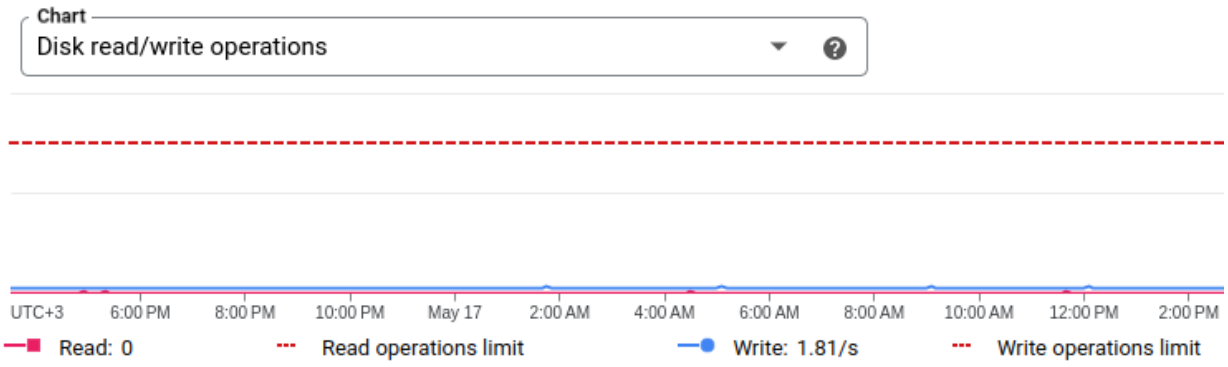

Рисунок 3.23 – Операції читання та запису в БД

Без використання практик моніторингу ці проблеми могли залишитися непоміченими і в результаті можуть виникнути серйозні проблеми з продуктивністю та надійністю веб-додатку. Практики моніторингу дозволяють швидко виявляти проблеми та вчасно вживати заходів для їх вирішення, що дозволяє забезпечити якість та стабільність роботи веб-додатку.

3.6 Висновок до третього розділу

У третьому розділі розглянуто впровадження методів DevOps для автоматизації та моніторингу проекту «Файловий менеджер» та їхній аналіз. Та спроєктовано архітектуру веб-додатку.

Було налаштовано етап постійної інтеграції за допомогою GitHub Actions та використано Google Cloud Platform для розгортання веб-додатку в хмарі.

Створено базу даних у хмарі, налаштовано провайдера ідентифікації для GitHub Actions, створено віртуальну машину для веб-додатку та репозиторій для Docker-контейнерів.

Також створено конвеєр в GitHub Actions для релізу нових версій методологією CI/CD. Було налаштовано доменне ім'я та SSL сертифікати для веб-додатку. Налаштовано моніторинг інфраструктури та продуктивності веб-додатку за допомогою Google Cloud Monitoring.

Проаналізовано реалізації DevOps методологій та практик для проекту «Файловий менеджер». Здійснено оцінку результатів застосування методологій CI/CD та проаналізовано проблеми веб-додатку за допомогою інструментів моніторингу.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Законодавча основа Євросоюзу з питань охорони праці. Рамкова директива 89/391/ЄС «Про введення заходів, що сприяють поліпшенню безпеки та гігієни праці працівників».

Тема кваліфікаційної роботи освітнього рівня «Магістр» присвячена дослідженню методології DevOps для розробки та підтримки веб-застосунків. Хоча методології DevOps спрямовані на оптимізацію процесу розробки, інженери DevOps працюють у командах і часто тісно співпрацюють з іншими відділами, включаючи IT-операції та розробку, що робить командну роботу важливою частиною процесу DevOps, тому дуже важливо забезпечити безпеку робочого середовища для всіх залучених співробітників

Оскільки все більше і більше компаній впроваджують методології DevOps у розробці веб-додатків, важливо враховувати правила безпеки на робочому місці та закони, які регулюють безпеку працівників. Європейський Союз створив законодавчу базу з безпеки та гігієни праці за допомогою директиви 89/391/ЄС 1989 року, яка спрямована на поліпшення умов праці працівників у всьому ЄС.

Одним із способів забезпечення безпеки на робочому місці є включення стандартів і практик безпеки в робочий процес DevOps. Команди DevOps повинні переконатися, що заходи безпеки інтегровані в процес розробки програмного забезпечення з самого початку, в тому числі на етапах проектування та тестування. Інженери DevOps також повинні пройти належну підготовку та освіту щодо стандартів безпеки, щоб переконатися, що вони оснащені знаннями та навичками для виявлення та усунення небезпек на робочому місці.

Крім того, оскільки DevOps-команди часто працюють віддалено або в розподілених командах, комунікація та співпраця є життєво важливими. Інженери DevOps повинні обов'язково повідомляти про будь-які проблеми безпеки та співпрацювати з іншими членами команди для впровадження заходів безпеки. Також повинні проводитися регулярні перевірки та оцінки безпеки, щоб

гарантувати, що робоче місце залишається безпечним і здоровим для всіх залучених працівників.

4.1.1 Мета директиви, визначення та принципи

Метою цієї директиви 89/391/ЄС 1989 року є впровадження заходів для заохочення поліпшення безпеки та здоров'я працівників на роботі. Це стосується всіх секторів діяльності, як державних, так і приватних, за винятком конкретних видів державної служби, таких як збройні сили, поліція або певні служби цивільного захисту [53].

Вона має принципове значення, оскільки є основним нормативно-правовим актом з безпеки та гігієни праці, який встановлює загальні принципи запобігання та захисту працівників від нещасних випадків на виробництві та професійних захворювань. Вона містить принципи, що стосуються запобігання ризикам, захисту безпеки та гігієни праці, оцінки ризиків, усунення ризиків і факторів аварії, інформування, консультацій та збалансованої участі та навчання працівників та їх представників.

На основі цієї Основної директиви була прийнята серія індивідуальних директив. Основна директива з її загальними принципами продовжує застосовуватися в повномоу обсязі до всіх сфер, охоплених окремими директивами, але там, де окремі директиви містять більш жорсткі та/або конкретні положення, ці спеціальні положення окремих директив мають переважну силу.

Визначення термінів які використовуються в директиві: «працівник», «роботодавець», «представник працівника з особливою відповідальністю за безпеку та здоров'я працівників» та «профілактика».

Основна директива містить базові зобов'язання для роботодавців і працівників. Проте обов'язки працівників не повинні впливати на принцип відповідальності роботодавця.

Роботодавець зобов'язаний забезпечити безпеку та здоров'я працівників у всіх аспектах, пов'язаних з роботою, і він не може покладати фінансові витрати на працівників для досягнення цієї мети. Так само, якщо роботодавець залучає компетентні зовнішні служби або осіб, це не звільняє його від обов'язків у цій сфері.

Загальні профілактичні принципи, перераховані в директиві, такі:

1. Уникнення ризиків.
2. Оцінка ризиків.
3. Боротьба з ризиками в корені.
4. Адаптація роботи до особистості.
5. Адаптація до технічного прогресу.
6. Заміна небезпечного безпечним або менш небезпечним.
7. Розробка послідовної загальної політики профілактики.
8. Надання пріоритету колективним захисним заходам (над індивідуальними захисними заходами).
9. Надання відповідних інструкцій робітникам.

Необхідно забезпечити нагляд за станом здоров'я працівників відповідно до національних систем. Особливо чутливі групи ризику повинні бути захищені від небезпек, які конкретно впливають на них.

4.1.2 Обов'язки роботодавців і працівників

Ця директива спрямована на впровадження заходів для заохочення поліпшення безпеки та здоров'я працівників на роботі. Це стосується всіх видів діяльності, як державних, так і приватних, за винятком конкретних видів державної служби, таких як збройні сили, поліція або певні служби цивільного захисту [54]. Директива містить основні зобов'язання для роботодавців і працівників. Роботодавець зобов'язаний забезпечити безпеку та здоров'я працівників у всіх аспектах, пов'язаних з роботою, і він не може покладати фінансові витрати на працівників для досягнення цієї мети. Аналогічно, якщо

роботодавець залучає компетентні зовнішні служби або осіб, це не звільняє його від виконання обов'язків [55].

Роботодавець зобов'язаний:

1. Оцінити всі ризики для безпеки та здоров'я працівників, в тому числі при виборі робочого обладнання, використовуваних хімічних речовин або препаратів, обладнанні робочих місць.

2. Вживати заходів, які забезпечують підвищення рівня захисту, що надається працівникам, та інтегруються в усі види діяльності підприємства та/або установи на всіх ієрархічних рівнях.

3. Враховувати можливості працівника щодо охорони здоров'я та безпеки, коли він доручає завдання працівникам.

4. Консультувати працівників з питань впровадження нових технологій. призначати працівників (працівників) для здійснення діяльності, пов'язаної із захистом та запобіганням професійним ризикам.

5. Вживати необхідних заходів для надання першої медичної допомоги, гасіння пожеж, евакуації працівників і дій, необхідних у разі серйозної і неминучої небезпеки.

6. Вести список нещасних випадків на виробництві та складати і складати, для відповідальних органів звіти про нещасні випадки на виробництві, яких зазнали його працівники.

7. Інформувати і консультувати працівників і дозволяти їм брати участь в обговореннях з усіх питань, що стосуються безпеки та гігієни праці на виробництві.

8. Забезпечити, щоб кожен працівник пройшов належну підготовку з безпеки та гігієни праці.

Згідно з директивою, працівники зобов'язані співпрацювати зі своїм роботодавцем, щоб дати їм можливість виконувати вимоги директиви. Працівники також повинні дбати про свою безпеку та здоров'я, а також про безпеку та здоров'я інших осіб, які постраждали від їхніх дій або бездіяльності на роботі. Вони повинні використовувати робоче обладнання та речовини

відповідно до інструкцій, наданих їхнім роботодавцем, та інформувати свого роботодавця або його представника про будь-яку робочу ситуацію, яка становить серйозну та безпосередню небезпеку для безпеки та здоров'я [56].

Детальніше про зобов’язання працівників, вони повинні:

1. Правильно використовувати машини, апарати, інструменти, небезпечні речовини, транспортне обладнання, інші засоби виробництва та засоби індивідуального захисту.

2. Негайно інформувати роботодавця про будь-яку робочу ситуацію, що становить серйозну і безпосередню небезпеку, і про будь-які недоліки в заходах захисту.

3. Співпрацювати з роботодавцем у виконанні будь-яких вимог, що пред'являються до охорони здоров'я та безпеки, а також у наданні йому можливості забезпечити, щоб робоче середовище та умови праці були безпечними та не становили ризиків.

Необхідно забезпечити нагляд за станом здоров'я працівників відповідно до національних систем. Особливо чутливі групи ризику повинні бути захищені від небезпек, які конкретно впливають на них.

Також відвовідно до 12 статті директиви ради 89/391/ЄЕС [57], роботодавці зобов'язані забезпечити належну підготовку працівників з питань безпеки та гігієни праці. Таке навчання повинно відбуватися в робочий час і не може здійснюватися за рахунок працівників або їх представників.

Оскільки методології DevOps продовжують набирати обертів у розробці веб-додатків, важливо враховувати правила та закони безпеки на робочому місці. Інженери DevOps повинні співпрацювати з іншими членами команди, щоб інтегрувати стандарти та практики безпеки в процес розробки програмного забезпечення з самого початку. Надаючи пріоритет безпеці на робочому місці, компанії можуть гарантувати, що їхні співробітники можуть працювати продуктивно та безпечно, сприяючи позитивному та здоровому робочому середовищу.

4.2 Планування заходів цивільного захисту на об’єкті у випадку надзвичайних ситуацій

Планування цивільного захисту об'єкта – це розроблення сукупності документів, у яких визначені сили і засоби, порядок і послідовність дій з метою забезпечення захисту населення, виробництва, а також виконання завдань вищих органів, пов'язаних із поданням допомоги населенню інших об'єктів і міст [58].

Ці документи, розроблені з урахуванням реальних можливостей і умов об'єкта, є настановою для організованих дій як з метою підготовки об'єкта до захисту в надзвичайних умовах, так із метою ліквідації наслідків надзвичайних ситуацій (стихійних лих, виробничих аварій і вогнищ воєнних конфліктів) [59].

На об'єкті мають бути розроблені два плани: на воєнний та мирний час.

План цивільного захисту на воєнний час – це документи, які визначають організацію і порядок переведення об'єкта з мирного на воєнний час і ведення цивільного захисту в початковий період війни.

План цивільного захисту на мирний час – це документи, які визначають організацію і порядок виконання заходів цивільного захисту з метою запобігання або зменшення можливих втрат від важких виробничих аварій, катастроф і стихійних лих, а також ведення рятувальних та інших невідкладних робіт при їх виникненні.

Планування цивільного захисту об'єкта. Розробка плану відбувається у три етапи в певній послідовності.

Перший етап – підготовчий, протягом якого визначається склад виконавців і затвердження їх, підготовка виконавців до роботи, доведення до них директив, рекомендацій та інших документів, узагальнення й аналіз вихідних даних, необхідних для розробки плану ЦЗ, визначення обсягу робіт і розподіл обов'язків між виконавцями та закріплення відповідальних за розділами плану.

Другий етап – практична розробка, оформлення документів. Заходи, які плануються в документах плану, мають бути спрямовані на виконання завдань ЦЗ у надзвичайних ситуаціях.

Третій етап – узгодження розроблених планів із відділом ЦЗ району, з районним агропромисловим управлінням, адміністрацією населеного пункту, службами ЦЗ району, після цього затвердження документів плану ЦЗ. Документи плану ЦЗ підписує керівник – ЦЗ об'єкта, деякі (план евакуації, прийому і розміщення евакуйованого) підписує і начальник ЦЗ голова адміністрації населеного пункту. Зміст плану ЦЗ об'єкту узгоджується з вимогами плану ЦЗ району, що підтверджує начальник відділу з питань цивільного захисту населення району, після чого план ЦЗ затверджує керівник ЦЗ об'єкта.

План цивільного захисту на особливий період. План на воєнний час складається з текстової частини і додатків. Текстова частина складається з трьох розділів.

Розділ 1. Оцінка обстановки, що може скластися на об'єкті в результаті дій противника**.**

У цьому розділі висвітлюється: коротка характеристика і оцінка обстановки, що може скластися на території об'єкта після несподіваного нападу і при плановому переведенні ЦЗ на воєнний стан:

- Можливий ступінь руйнування виробничих дільниць і житлових будинків.
- Ступінь радіоактивного забруднення тварин, території. можливість виникнення і характер впливу осередків сильнодіючих отруйних речовин (СДОР), лісових, торфових пожеж, зон затоплення. можливе зниження виробництва.
- Можлива радіаційна, пожежна і хімічна обстановка. стан транспортних артерій, систем енерго-, газо-, водо-, теплозабезпечення, матеріально-технічної бази, оповіщення, зв'язку і управління. втрати сил і засобів ЦЗ і людей, об'єкта та населеного пункту. втрати від повторних факторів ураження. обставини, які можуть скластися на території об'єкта і населеного пункту при використанні противником звичайних засобів ураження.

Висновки з оцінки можливої обстановки і стан сил для рятувальних робіт, вплив на вирішення завдань об'єкта при переведенні на воєнний стан і в період проведення рятувальних та інших невідкладних робіт.

Розділ 2. Виконання заходів на об'єкті при планомірному переведенні на особливий період.

Виконання заходів при загрозі нападу противника:

1. Захист працюючих і членів їх сімей.

2. Заходи забезпечення стійкої роботи у воєнний час.

3. Заходи і ведення рятувальних та інших невідкладних робіт.

4. Організація забезпечення заходів ЦЗ.

5. Організація управління.

Розділ 3. Виконання заходів ЦЗ на об'єкті в умовах несподіваного нападу противника.

1. Дії за сигналом "Повітряна тривога" (ПТ).

2. Дії після нападу противника.

Додатки до плану на воєнний час описують плани та заходи, що повинні бути здійснені для захисту працюючих, їхніх сімей та об'єкта у випадку воєнного часу або інших небезпечних ситуацій. Вони містять розрахунки укриття працюючих, прийому та розміщення евакуйованого населення, склад сил і засобів об'єкта, а також схему управління, зв'язку і оповіщення. План-графік нарощування заходів підвищення стійкості роботи об'єкта у воєнний час є також важливим елементом цих додатків, отже вони мають на меті забезпечити безпеку працівників та збереження об'єкта у небезпечних умовах.

План цивільного захисту на мирний час. План складається з текстової частини і додатків. Текстова частина плану складається з двох розділів.

Розділ 1. Висновки з оцінки можливої обстановки на об'єкті при виникненні великих виробничих аварій, катастроф і стихійних лих.

Зміст: перелік можливих великих аварій, катастроф і стихійних лих на даному об'єкті; висновки з оцінки обстановки, яка може скластися на об'єкті при виникненні великих виробничих аварій, катастроф і стихійних лих.

Розділ 2. Здійснення заходів при загрозі і виникненні великих виробничих аварій, катастроф і стихійних лих на об'єкті.

При виникненні небезпечних ситуацій необхідно оперативно діяти із застосуванням відповідних заходів. До таких заходів можна віднести:

1. Заходи при небезпеці великих аварій, катастроф і природних катастроф: оповіщення керівного складу ЦЗ, працівників та населення. повідомлення вищих органів. готовність сил і засобів для рятувальних робіт. заходи для захисту населення та майна.

2. Заходи при великих аваріях, катастрофах та природних катастрофах: повідомлення керівного складу, працівників та вищих органів ЦЗ. розвідка та спостереження на об'єкті аварії. готовність рятувальних сил. організація медичної допомоги та безаварійної зупинки виробництва. заходи для захисту населення.

3. Організація управління: перехід керівництва ЦЗ до пунктів управління. забезпечення зв'язку з підрозділами та вищими органами управління. подання донесень вищим територіальним та галузевим органам.

На випадок аварії на АЕС важливими заходами є організація управління силами і засобами. Крім того, в районі розміщення АЕС необхідно виконати такі заходи: забезпечити високий ступінь готовності захисних споруд (ЗС) у 30-кілометровій зоні, забезпечити фонд ЗС для повного укриття на об'єкті працюючих і членів їхніх сімей; забезпечити виконання комплексу медичних заходів; створити запас засобів розвідки, дозиметричного контролю, захисту органів дихання, шкіри, знезараження. Управління ЦЗ разом з керівництвом АЕС складає план заходів захисту населення: оповіщення населення про можливі наслідки аварії; захист населення; заходи ліквідації наслідків аварії; ведення рятувальних та інших невідкладних робіт.

Із досвіду аварії на ЧАЕС заходи ЦЗ необхідно планувати у три етапи:

1-й – від початку аварії до трьох діб. У цей час необхідно терміново оцінити обстановку і масштаби проведення першочергових заходів, спрямованих на захист населення і запобігання наслідкам аварії; інформація про

аварію; виклик аварійних бригад і формувань ЦЗ; проведення заходів ліквідації наслідків аварії;

2-й – понад 1 добу після аварії; уточнити радіаційну обстановку; вжити додаткові заходи для захисту населення; дозиметричний контроль;

3-й – перехідний – від аварійного до нормального стану (коли вжиті всі заходи захисту): уточнюються дози опромінення, ступінь забрудненості РР урожаю, продуктів, води, сировини та ін.

Додатки для плану цивільного захисту на мирний час:

1. Календарний план основних заходів ЦЗ при загрозі й виникненні великих виробничих аварій, катастроф і стихійних лих.

2. План захисту об'єкта і проведення рятувальних та інших невідкладних робіт із зазначеними потенційно небезпечними місцями.

3. Розрахунок сил і засобів для виконання заходів ЦЗ при загрозі й виникненні аварій, катастроф і стихійних лих.

4. План медичного забезпечення.

5. Розрахунок евакозаходів.

6. Схема організації управління, зв'язку і оповіщення.

Планування цивільного захисту об'єкта – це розроблення сукупності документів, у яких визначені сили і засоби, порядок і послідовність дій з метою забезпечення захисту населення, виробництва, а також виконання завдань вищих органів, пов'язаних із наданням допомоги населенню інших об'єктів і міст.

4.3 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи описано відношення DevOps-інженерів у контексті законодавства Євросоюзу з питань охорони праці. Особливу увагу приділено рамковій директиві 89/391/ЄС, яка встановлює загальні принципи та заходи для поліпшення безпеки та гігієни праці працівників. Розглянуто мету директиви, її визначення та принципи. Директива має на меті забезпечити високий рівень охорони здоров'я та безпеки працівників,

а також запобігти професійним захворюванням та нещасним випадкам на роботі. Директива визначає поняття роботодавця, працівника, ризику, запобіжному заходу тощо. Вона ґрунтується на принципах запобігання ризикам, оцінки ризиків, інформування та консультації працівників та покращення умов праці.

Також аналізуються обов'язки роботодавців і працівників щодо охорони праці. Роботодавець зобов'язаний забезпечити безпечну та здорову робочу середовище для своїх працівників, а також проводити оцінку ризиків, вживати заходи щодо їх усунення або зменшення, надавати необхідне обладнання та засоби захисту, інструктувати та навчати працівників тощо. Працівник зобов'язаний дотримуватися правил охорони праці, користуватися обладнанням та засобами захисту належним чином, повідомляти про небезпечні ситуації та нещасні випадки на роботі.

Для пункту безпеки в надзвичайних ситуаціях розглядаються питання планування заходів цивільного захисту у випадку НС.

На об'єкті мають бути розроблені два плани: на воєнний та мирний час.

План цивільного захисту на воєнний час – це документи, які визначають організацію і порядок переведення об'єкта з мирного на воєнний час і ведення цивільного захисту в початковий період війни.

План цивільного захисту на мирний час – це документи, які визначають організацію і порядок виконання заходів цивільного захисту з метою запобігання або зменшення можливих втрат від важких виробничих аварій, катастроф, і стихійних лих, а також ведення рятувальних та інших невідкладних робіт при їх виникненні

ВИСНОВКИ

В даній кваліфікаційній роботі проаналізовано стан досліджень в області DevOps методологій та відповідних інструментів. Зокрема:

- Проаналізовано найпопулярніші на сьогоднішній день методології та інструментальні засоби DevOps, зокрема, такі як continuous integration та continuous delivery (CI/CD).
- Проаналізовано основні методи тестування та моніторингу програмного забезпечення, що використовуються в складі DevOps.
- Виконано порівняння існуючих DevOps-інструментів для автоматизації процесів розробки, тестування, розгортання та моніторингу програмного забезпечення.
- Застосовано на практиці методи та інструменти DevOps при розробці веб-додатку «Файловий менеджер» та здійснено оцінку їх ефективності, переваг та недоліків.

У першому розділі кваліфікаційної роботи досліджено DevOps методології та інструменти, які допомагають автоматизувати процеси розробки, тестування, впровадження нових змін та моніторингу веб-додатків. Розглянуто DevOps як культуру співпраці між розробниками та адміністраторами, яка спрямована на швидку та надійну доставку програмного забезпечення. Проаналізовано практики та інструменти для автоматизації процесів, таких як CI/CD, яке дозволяє забезпечити постійну інтеграцію та постійну доставку програмного забезпечення.

У другому розділі кваліфікаційної роботи проведено дослідження та обґрунтування вибору інструментів для моніторингу та автоматизації процесів при розробці веб-додатку «Файловий менеджер». Було виявлено, що використання CI/CD засобами GitHub Actions дозволяє покращити якість коду та прискорити доставку програмного продукту. Також було показано, що контейнеризація в Docker спрощує процес установки та запуску на виконання додатку на різних платформах. Було обрано хмарний провайдер GCP для

розміщення додатку та його компонентів, таких як база даних Cloud SQL та сервер Compute Engine та сервісів для моніторингу показників та метрик. Було розглянуто процес реєстрації доменного імені та отримання SSL-сертифікатів для забезпечення безпеки передачі даних та роботи з веб-додатком.

У третьому розділі продемонстровано застосування DevOps методологій та інструментів для розробки та підтримки веб-додатку «Файловий менеджер». Для цього використано GitHub Actions для автоматизації процесу постійної інтеграції та доставки, GCP для розгортання веб-додатку в хмарному середовищі, Google Cloud Monitoring для моніторингу стану інфраструктури та продуктивності веб-додатку.

Проаналізовано переваги та недоліки DevOps практик для проекту, а також виявлено можливі напрямки для подальшого вдосконалення їх застосування.

На основі проведеного аналізу можна зробити висновки, що застосування DevOps методологій та інструментів для проекту «Файловий менеджер» дозволило покращити ефективність розробки, тестування, релізів нових версій та моніторингу ПЗ. Зокрема, було досягнуто таких переваг:

- Скорочення часу на розгортання нових версій веб-додатку.
- Покращення якості коду за рахунок автоматизованих тестів.
- Зменшення витрат на інфраструктуру за рахунок контейнеризації.
- Покращення контролю над станом веб-додатку за рахунок моніторингу.

Застосовуючи DevOps методології та інструменти для проекту «Файловий менеджер», необхідно також брати до уваги проблеми та обмеження, яким цей підхід підлягає. Зокрема, необхідно: забезпечити ефективну комунікацію і спільноту між учасниками DevOps команди; постійно підтримувати актуальний і безпечний інструментарій; балансувати між швидкою доставкою і якістю програмного забезпечення.

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» проаналізовано законодавства Євросоюзу з питань охорони праці та питання планування заходів цивільного захисту у випадку надзвичайних ситуацій.

ПЕРЕЛІК ДЖЕРЕЛ

1. Fowler, Martin, and Jim Highsmith. "The agile manifesto." Software development 9.8 (2001): 28-35.

2. A Brief History of Devops [with Infographic]. KnowledgeHut: Professional Bootcamps and Certification Courses Provider for your Future. URL: https://www.knowledgehut.com/blog/devops/history-of-devops (дата звернення: 12.03.2023).

3. C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, "DevOps," in IEEE Software, vol. 33, no. 3, pp. 94-100, May-June 2016, doi: 10.1109/MS.2016.68.

4. Leite, Leonardo, et al. A survey of DevOps concepts and challenges. ACM Computing Surveys (CSUR), 2019, 52.6: 1-35.

5. Sacks, Matthew. Devops principles for successful web sites. In: Pro Website Development and Operations: Streamlining DevOps for Large-Scale Websites. Berkeley, CA: Apress, 2012. p. 1-14.

6. Awad, M. A. A comparison between agile and traditional software development methodologies. University of Western Australia, 2005, 30: 1-69.

7. 8 Software Development Models Organized in Charts and Explained (scnsoft.com) URL: https://www.scnsoft.com/blog/software-development-models. (дата звернення: 13.03.2023).

8. Rise of DevOps – The Evolution of Software Development Life Cycle (SDLC) URL: https://www.htown-tech.com/blogs/rise-of-devops-the-evolution-of-software-development-life-cycle. (дата звернення: 14.03.2023).

9. DevOps Roadmap – Walkthrough – YouTube URL: https://www.youtube.com/watch?v=hKfU_QNJzo8. (дата звернення: 14.03.2023).

10. DevOps Roadmap Step by step guide for DevOps, SRE or any other Operations Role in 2023 URL: https://roadmap.sh/devops. (дата звернення: 25.03.2023).

11. Мачужак А. Автоматизація задач тестування та розгортання програмного забезпечення / А. Мачужак // Матеріали X науково-технічної

конференції „Інформаційні моделі, системи та технології“, 7–8 грудня 2022 року. — Т. : ТНТУ, 2022. — С. 61. — (Інформаційні системи та технології, кібербезпека).

12. Готович В. А. Застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення / В. А. Готович, А. В. Мачужак // X I Міжнародна науково-практична конференція молодих учених та студентів „Актуальні задачі сучасних технологій“, 7-8 грудня 2022 року. — Т. : ТНТУ, 2022. — С. 131–132. — (Комп’ютерно-інформаційні технології та системи зв’язку).

13. Understanding GitHub Actions – GitHub Docs. GitHub Docs. URL: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions#understanding-the-workflow-file. (дата звернення: 25.03.2023).

14. Leszko, Rafal. Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines. Packt Publishing Ltd, 2022.

15. Belmont, Jean-Marcel. Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI. Packt Publishing Ltd, 2018.

16. Achdian, Asfin; MARWAN, M. Akbar. Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company. CD Application Based on Cloud Computing Services on Fintech Company, 2019, 4.3: 112-114.

17. Schlossnagle, Theo. Monitoring in a DevOps world. Communications of the ACM, 2018, 61.3: 58-61.

18. Chen, Boyuan. Improving the software logging practices in DevOps. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2019. p. 194-197.

19. Why Monitoring and Logging are Important in DevOps. URL: https://www.kovair.com/blog/why-monitoring-and-logging-are-important-in-devops/. (дата звернення: 27.03.2023).

20. What is DevOps Monitoring? Types, Importance and Use Cases. URL: https://www.simform.com/blog/devops-monitoring/. (дата звернення: 01.04.2023).

21. The Importance of Monitoring in DevOps – DevOps Online. URL: https://www.devopsonline.co.uk/the-importance-of-monitoring-in-devops/. (дата звернення: 03.04.2023).

22. The Importance of Constant Monitoring and Feedback – A DevOps Blog. URL: https://www.valewood.org/devops-monitoring. (дата звернення: 03.04.2023).

23. DevOps Monitoring | Atlassian. URL: https://www.atlassian.com/devops/devops-tools/devops-monitoring. (дата звернення: 05.04.2023).

24. Docker, Inc. Docker. lınea.[Junio de 2017]. Disponible en. URL: https://www. docker. com/what-docker, 2020. (дата звернення: 05.04.2023).

25. Miell, Ian; SAYERS, Aidan. Docker in practice. Simon and Schuster, 2019.

26. Luksa, Marko. Kubernetes in action. Simon and Schuster, 2017.

27. Kubernetes Documentation. Concepts, overview URL: https://kubernetes.io/docs/concepts/overview/. (дата звернення: 10.04.2023).

28. Qian, Ling, et al. "Cloud computing: An overview." Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1. Springer Berlin Heidelberg, 2009.

29. Alhamazani, Khalid, et al. "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art." Computing 97 (2015): 357-377.

30. Global cloud infrastructure market share 2022 | Statista URL: https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/. (дата звернення: 10.04.2023).

31. Chaput, Shawn R., and Katarina Ringwood. "Cloud compliance: A framework for using cloud computing in a regulated world." Cloud computing: Principles, systems and applications (2010): 241-255.

32. Battina, Dhaya Sindhu, Devops, A New Approach To Cloud Development & Testing (August 8, 2020). International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, Vol.7, Issue 8, page no.982-

985, August-2020, Available :http://www.jetir.org/papers/JETIR2008432.pdf, Available at SSRN: https://ssrn.com/abstract=4004330

33. Jones, "A Proposal for Integrating DevOps into Software Engineering Curricula", International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, pp. 3347, 2018.

34. 40+ DevOps Statistics You Should Know in 2023 | StrongDM. URL: https://www.strongdm.com/blog/devops-statistics. (дата звернення: 10.04.2023).

35. Benefits of Using Cloud With DevOps Services – Dzone URL: https://dzone.com/articles/benefits-of-using-cloud-with-devops-services. (дата звернення: 15.04.2023).

36. Benefits of DevOps | Atlassian. URL: https://www.atlassian.com/devops/what-is-devops/benefits-of-devops. (дата звернення: 15.04.2023).

37. Riungu-Kalliosaari, Leah, et al. "DevOps adoption benefits and challenges in practice: A case study." Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17. Springer International Publishing, 2016.

38. AWS DevOps Services Case Study – Implemented a CI/CD Pipeline on Tix (rapyder.com) URL: https://www.rapyder.com/case-studies/tix-technologies-devops-case-study/. (дата звернення: 19.04.2023).

39. Containers vs. Virtual Machines (VMs): What’s the Difference? | IBM. URL: https://www.ibm.com/cloud/blog/containers-vs-vms (дата звернення: 19.04.2023).

40. Felter, Wes, et al. "An updated performance comparison of virtual machines and linux containers." 2015 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, 2015. Cloud SQL documentation. Default MySQL configuration https://cloud.google.com/sql/docs/mysql/features

41. Мачужак А. В. Розробка веб-сервісу „Файловий менеджер” : кваліфікаційна робота освітнього рівня „Бакалавр“ „122 — комп’ютерні науки“ / А. В. Мачужак. — Тернопіль : ТНТУ, 2021. — 76 с.

42. Global Locations – Regions & Zones | Google Cloud. Google Cloud. URL: https://cloud.google.com/about/locations/ (дата звернення: 19.04.2023).

43. Gcloud compute instances create-with-container | Google Cloud CLI Documentation URL: https://cloud.google.com/sdk/gcloud/reference/compute /instances/create-with-container (дата звернення: 25.04.2023).

44. Add a persistent disk to your VM | Compute Engine Documentation | Google Cloud. Google Cloud. URL: https://cloud.google.com/compute/docs/disks/add-persistent-disk#create_disk (дата звернення: 25.04.2023).

45. What is DNS? | How DNS works | Cloudflare. URL: https://www.cloudflare.com/learning/dns/what-is-dns/ (дата звернення: 26.04.2023).

46. GitHub – certbot/certbot: Certbot is EFF's tool to obtain certs from Let's Encrypt and (optionally) auto-enable HTTPS on your server. It can also act as a client for any other CA that uses the ACME protocol. GitHub. URL: https://github.com/certbot/certbot (дата звернення: 26.04.2023).

47. Getting Started – Let's Encrypt. Let's Encrypt. URL: https://letsencrypt.org/getting-started/ (дата звернення: 26.04.2023).

48. Cloud Logging and Monitoring – happtiq. happtiq. URL: https://www.happtiq.com/cloud-logging-and-monitoring (дата звернення: 27.04.2023).

49. Operations: Cloud Monitoring & Logging | Google Cloud. Google Cloud. URL: https://cloud.google.com/products/operations (дата звернення: 29.04.2023).

50. Learn about using private IP | Cloud SQL for MySQL | Google Cloud. Google Cloud. URL: https://cloud.google.com/sql/docs/mysql/private-ip (дата звернення: 30.04.2023).

51. Vargo S., Baiju B. Enabling keyless authentication from GitHub Actions | Google Cloud Blog. Google Cloud Blog. URL:

https://cloud.google.com/blog/products/identity-security/enabling-keyless-authentication-from-github-actions (дата звернення: 30.04.2023).

52. Variables – GitHub Docs. GitHub Docs. URL: https://docs.github.com/en/actions/learn-github-actions/variables (дата звернення: 16.05.2023).

53. Жидецький В. Ц. Основи охорони праці / В. Ц. Жидецький. – Л. : Афіша, 2005. – 349 с

54. Main Obligations and Duties of Employers and Employees. Wonder.Legal – Contracts, Letters and Agreements. URL: https://www.wonder.legal/au/guide/main-obligations-and-duties-of-employers-and-employees (дата звернення: 05.05.2023).

55. 26 employees and employers rights and responsibilities – wisestep. Wisestep. URL: https://content.wisestep.com/employees-and-employers-rights-and-responsibilities/ (дата звернення: 07.05.2023).

56. Are you an employee: workers' health and safety. HSE: Information about health and safety at work. URL: https://www.hse.gov.uk/workers/responsibilities.htm (дата звернення: 05.05.2023).

57. Директива Ради від 12 червня 1989 року про запровадження заходів, покликаних заохочувати до покращення безпеки та охорони здоров'я працівників на роботі (89/391/ЄЕС). Офіційний веб-портал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/994_b23#Text (дата звернення: 05.05.2023).

58. Методичні рекомендації щодо розроблення планів з питань цивільного захисту / Державна служба України з надзвичайних ситуацій. Київ: ДСНС, УНДІЦЗ, 2015. 148с.

59. Про затвердження Порядку розроблення планів діяльності єдиної державної системи цивільного захисту. Офіційний веб-портал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/626-2017-п#Text (дата звернення: 09.05.2023).

ДОДАТКИ

Додаток А

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Міжнародний університет цивільної авіації (Марокко)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XI Міжнародної науково-практичної конференції молодих учених та студентів
7-8 грудня 2022 року

УКРАЇНА
ТЕРНОПІЛЬ – 2022

Матеріали XI Міжнародної науково-практичної конференції молодих учених та студентів «АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 7-8 грудня 2022 року

УДК 004.5

В.А. Готович, к.т.н., А.В. Мачужак
Тернопільський національний технічний університет імені Івана Пулюя

ЗАСТОСУВАННЯ МЕТОДОЛОГІЇ CI/CD ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

V.A. Hotovych, Ph.D., A.V. Machuzhak

APPLICATION OF CI/CD METHODOLOGY FOR AUTOMATION OF SOFTWARE TESTING AND DEPLOYMENT PROCESSES

Для успішності технологічного процесу розробки програмного забезпечення (ПЗ) важливим є вирішення завдань підтримки якості як програмного коду зокрема, так і програмного продукту в цілому, а також постачання чергових версій програмного продукту до споживача. На практиці вирішення цих задач відбувається автоматизованим чином шляхом застосування методології CI/CD (англ. Continuous Integration, Continuous Delivery) [1]. Дана методологія складається з двох етапів [2]:

1. Безперервна інтеграція (CI). Це практика інтегрування програмного коду в основну гілку репозиторію при автоматичному тестуванні якості коду. Зокрема, дана практика вирішує проблему наявності занадто великої кількості гілок в системі контролю версій в процесі розробки, які можуть конфліктувати між собою.
2. Безперервне постачання (CD). Це практика безперервного постачання протестованого на попередньому етапі програмного коду (нових версій ПЗ) до кінцевого користувача. Це етап автоматизації, на якому програмне забезпечення розгортається в одному або декількох середовищах виконання. Команди розробників зазвичай використовують декілька окремих середовищ для тестування програмного забезпечення.

В даній роботі наведено результати практичного застосування методології CI/CD для автоматизації процесів тестування та розгортання програмного забезпечення при роботі над конкретним програмним продуктом. Зокрема, реалізовано 2 конвеєри CI/CD [3], що схематично показано на рис. 1.

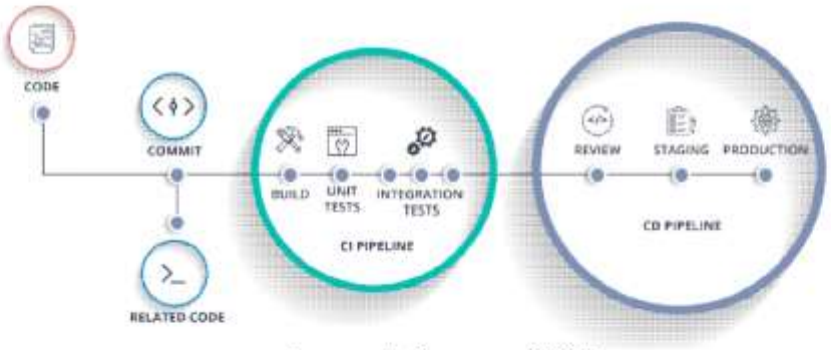

Рисунок 1. Конвеєри CI/CD

На першому етапі (постійної інтеграції), використовувався сервіс GitHub Actions, який надає можливості виконувати різні дії над програмним кодом в репозиторії проекту. Цей сервіс специфічний для GitHub. Аналогічні до GitHub сервіси для репозиторіїв надають відповідні можливості. Зокрема, для роботи з GitHub Actions створено конфігураційний yml-файл, в якому задано налаштування щодо того, при яких умовах слід виконувати дії, в якому середовищі, послідовність кроків тощо. Наприклад,

при відкритті пул реквеста (англ. pull request) чи нового коміта (англ. commit) запуститься програмне середовище з операційною системою Ubuntu та буде виконано необхідні кроки по інсталяції залежностей, збиранні коду, запуску тестів тощо.

На першому етапі застосування методології CI/CD вирішено наступні задачі:

- автоматизація процесів збирання коду;
- тестування ПЗ;
- швидке впровадження змін в спільний репозиторій.

На етапі безперервного постачання використано інструмент Jenkins. В конфігураційному файлі за допомогою спеціальної нотації описано різні етапи, такі як збірка, тестування та розгортання. Також задаються змінні середовища, параметри, секретні ключі, сертифікати та інші налаштування.

Результатом другого етапу є конвеєр безперервної доставки з такими етапами (рис. 2):

- отримання останніх змін в програмному коді з репозиторію;
- інфраструктурні зміни (збільшити чи зменшити потужності хмарної інфраструктури);
- переміщення програмного коду в цільове обчислювальне середовище та управління змінними середовища;
- виконання тестів та згортання середовища, якщо тести не виконалися успішно;
- надання даних з журналів (логів) та сповіщення про стан постачання ПЗ;
- розгортання нових змін в цільовому середовищі;
- сповіщення про завершення етапу.

Рисунок 2. Процес постачання нових версій ПЗ

Серед конкретних переваг, яких було досягнуто в результаті впровадження методології CI/CD, можна назвати:

- Підвищення продуктивності роботи команди розробників;
- Автоматичні збірки на зміни в коді, що дозволяє підготувати та протестувати нову версію програмного забезпечення;
- Автоматичне розгортання нових версій ПЗ.

Перспективами подальшого дослідження є тематика оркестрації контейнерів – керування контейнерами у великих масштабах за допомогою технології Kubernetes.

Література

1. CI/CD. URL: https://en.wikipedia.org/wiki/CI/CD (дата звернення: 28.11.2022).

2. What is CI/CD? Continuous integration and continuous delivery explained. URL: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html (дата звернення: 28.11.2022).

3. ROSSEL, Sander. Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests, and deployment. Packt Publishing Ltd, 2017.

Додаток Б

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

X НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

7–8 грудня 2022 року

ТЕРНОПІЛЬ
2022

УДК 004.5
А. Мачужак
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АВТОМАТИЗАЦІЯ ЗАДАЧ ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.5
A. Machuzhak

AUTOMATION OF SOFTWARE TESTING AND DEPLOYMENT TASKS

В процесі розробки програмного забезпечення (ПЗ) важливими є етапи контролю якості (тестування) та розгортання готових версій ПЗ (реліз) в робочому середовищі. При цьому необхідно вирішити задачі пов'язані з такими проблемами [1]:

- Уніфікована розробка – над проектом можуть працювати декілька людей, тому потрібно забезпечити однакові інструменти та середовище розробки, щоб виключити можливість непередбачуваних помилок;
- Тестування – забезпечувати високу якість ПЗ при швидкому темпі виходу нових версій та з можливістю використання тестового середовища без впливу на робоче середовище;
- Швидке розгортання нових змін – бізнес потребує нові можливості та функції для свого продукту, щоб адаптуватися до змін на ринку та бути конкурентоспроможним.

На сьогоднішній день такі задачі вирішуються за допомогою методологій та практик, які об'єднані спільною назвою DevOps [2] (англ. development operations). До основних інструментів на сьогоднішній день належать:

1. Docker, дозволяє інкапсулювати програми в так звані контейнери, які виконуватиметься однаково в різних середовищах;
2. GitHub actions або аналоги для постійної інтеграції (англ. Continuous integration – CI) та постійної доставки (англ. Continuous delivery – CD) – CI/CD такі як GitLab runners та Jenkins. Робота цих технологій полягає в автоматизації тестування та інтегрування нових змін в різні середовища виконання – тестове чи робоче;
3. New relic – забезпечує моніторинг показників продуктивності та працездатності веб-сайту, а також логування дій та ведення історії виконуваних операцій.

В даній роботі пропонується проект автоматизації процесів тестування та розгортання програмного забезпечення, який полягає у контейнеризації готового додатка в Docker, налаштування етапів CI/CD засобами github actions, в ці етапи включаються автоматичне тестування нових версії ПЗ та їхнє розгортання на сервері, як тестовому так і робочому. Останнім кроком буде інтеграція сервісу для моніторингу стану сервера.

До перспектив подальших досліджень належать теми безпеки в контейнеризованих додатках та налаштування інфраструктури для веб-сайту, методології розробки CI/CD, шаблони в хмарних технологіях, а також знання про бази даних (БД) та резервне копіювання БД, щоб захистити дані та мати змогу їх відновити.

Література

1. Паляниця В. А. Якість програмного забезпечення та тестування: базовий курс. Навчальний посібник. Тернопіль: ФОП Паляниця В. А., 2020. 478с.
2. Learn to become a DevOps Engineer or SRE. URL: https://roadmap.sh/devops (дата звернення: 28.11.2022).

Додаток В

Лістинги команд та конфігураційних файлів

Лістинг В.1 – Приклад робочого процесу, вміст файлу learn-github-actions.yml

```
name: 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@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

Лістинг В.2 – docker-compose файл для запуску всього веб-застосунка

```
version: "3.3"
services:
    db:
        image: mysql:8.0.21
        container_name: db
        restart: always
        environment:
          MYSQL_DATABASE: "${MYSQL_DB}"
          MYSQL_USER: "${MYSQL_USER}"
          MYSQL_PASSWORD: "${DB_PASS}"
          MYSQL_ROOT_PASSWORD: "${DB_PASS}"
        volumes:
            - "${ROOT_DIR}/db:/var/lib/mysql/"
        ports:
            - "3306:3306"
    web:
        command: scripts/run.sh
        build:
            context: .
            dockerfile: Dockerfile
        container_name: web
        restart: always
        volumes:
            - "${ROOT_DIR}:/usr/src/my-app/user_files"
        ports:
            - "3000:3000"
        depends_on:
            - "db"
```

Лістинг В.3 – створення диску

```
gcloud compute disks create file-manager-disk --zone=us-central1-c
--size=10GB --architecture=X86_64
```

Лістинг В.4 – конфігураційний файл lint.yml

```
name: lint
run-name: ${{ github.actor }} is learning GitHub Actions
on: [push]
jobs:
  check-linting:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3
   - uses: actions/setup-node@v3
    with:
     node-version: '14'
   - name: Install dependencies
     run: npm ci
   - name: Check linting
     run: npm run check
   - name: Run tests
     run: npm test
```

Лістинг В.5 Команди створення сервісів що надають тимчасову автентифікацію для GitHub Actions

```
macho0863@cloudshell:~ (file-manager-application)$ gcloud iam
workload-identity-pools create “file-manager-pool” --
project=”${DEVSHELL_PROJECT_ID}” --location=”global” --display-
name=”File manager pool”
Created workload identity pool [file-manager-pool].

Macho0863@cloudshell:~ (file-manager-application)$ gcloud iam
workload-identity-pools providers create-oidc “file-manager-
provider” \
  --project=”${DEVSHELL_PROJECT_ID}” \
  --location=”global” \
  --workload-identity-pool=”file-manager-pool” \
  --display-name=”File manager provider” \
  --attribute-
mapping=”google.subject=assertion.sub,attribute.actor=assertion.ac
tor,attribute.aud=assertion.aud” \
  --issuer-uri=”https://token.actions.githubusercontent.com”
Created workload identity pool provider [file-manager-provider].

Macho0863@cloudshell:~ (file-manager-application)$ gcloud iam
service-accounts add-iam-policy-binding “filemanager-service-
account@${DEVSHELL_PROJECT_ID}.iam.gserviceaccount.com” \
  --project=”${DEVSHELL_PROJECT_ID}” \
```

```
  --role=”roles/iam.workloadIdentityUser” \
  --member=
“principalSet://iam.googleapis.com/projects/1006422617706/location
s/global/workloadIdentityPools/file-manager-pool/*”
Updated IAM policy for serviceAccount [filemanager-service-
account@file-manager-application.iam.gserviceaccount.com].
```

Лістинг В.6 – створення віртуальної машини з докер образу

```
gcloud compute instances create-with-container file-manager-app-
instance \
    --container-image=us-central1-docker.pkg.dev/file-manager-
    application/file-manager-repository/file-manager-
    image:869368fda433 \
    --disk=”name=file-manager-disk,mode=rw,auto-delete=no” \
    --container-mount-disk=”mode=rw,mount-path=/usr/src/my-
    app/user_file,name=file-manager-disk” \
    --zone=us-central1-c

WARNING: Default device-name for disk name [file-manager-disk] will
be [file-manager-disk] because it is being mounted to a container
with [`--container-mount-disk`]
Created      [https://www.googleapis.com/compute/v1/projects/file-
manager-application/zones/us-central1-c/instances/file-manager-
app-instance].
NAME: file-manager-app-instance
ZONE: us-central1-c
MACHINE_TYPE: n1-standard-1
PREEMPTIBLE:
INTERNAL_IP: 10.128.0.2
EXTERNAL_IP: 34.173.168.229
STATUS: RUNNING
```

Лістинг В.7 – створення репозиторію в Artifact Registry

```
macho0863@cloudshell:~ (file-manager-application)$ gcloud artifacts
repositories    create    file-manager-repository    --repository-
format=docker --location=us-central1

Create request issued for: [file-manager-repository]
Waiting       for       operation       [projects/file-manager-
application/locations/us-central1/operations/616d5beb-7bfe-4982-
a1be-c7f6c9cb9ae9] to complete...done.
Created repository [file-manager-repository].
```

Лістинг В.8 – оновлення прав для роботи з Artifact Registry

```
gcloud projects add-iam-policy-binding file-manager-application --
member="serviceAccount:filemanager-service-account@file-manager-
```

```
application.iam.gserviceaccount.com"                                    --
role="roles/artifactregistry.repoAdmin"
Updated IAM policy for project [file-manager-application].
```

Лістинг В.9 – Конфігураційний файл робочого процесу GitHub Actions для розгортання в GCP

```
name: deploy
run-name: Deploy to Cloud
on:
  push:
    branches: ["master", "deploy-workflow"]
env:
  PROJECT_ID: file-manager-application
  GAR_LOCATION: us-central1
  SERVICE: file-manager-repository
jobs:
  deploy:
    permissions:
      contents: "read"
      id-token: "write"
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Google Auth
        id: auth
        uses: "google-github-actions/auth@v0"
        with:
          token_format: "access_token"
          workload_identity_provider: "${{ secrets.WIF_PROVIDER }}
          service_account: "${{ secrets.WIF_SERVICE_ACCOUNT }}"

      - name: Docker Auth
        id: docker-auth
        uses: "docker/login-action@v1"
        with:
          username: "oauth2accesstoken"
          password: "${{ steps.auth.outputs.access_token }}"
          registry: "${{ env.GAR_LOCATION }}-docker.pkg.dev"

      - name: Build and Push Container
        run: |-
          echo                                                          -e
'MYSQL_DB=${{vars.DB_NAME}}\nMYSQL_USER=${{vars.DB_USER}}\nDB_PASS
=${{secrets.DB_PASS}}\nMYSQL_HOST=${{vars.INSTANCE_HOST}}\nSECRET_
KEY=${{secrets.SECRET_KEY}}\nHOST=${{vars.HOST}}' > .env
          docker    build    -t    "${{    env.GAR_LOCATION    }}-
docker.pkg.dev/${{   env.PROJECT_ID   }}/${{   env.SERVICE   }}/file-
manager-image:${{ github.sha }}" ./
```

```
          docker push "${{ env.GAR_LOCATION }}-docker.pkg.dev/${{
env.PROJECT_ID }}/${{ env.SERVICE }}/file-manager-image:${{
github.sha }}"

      - name: Deploy to Compute Engine
        run: |-
          gcloud compute instances update-container file-manager-
app-instance       --zone=us-central1-c       --container-image=${{
env.GAR_LOCATION }}-docker.pkg.dev/${{ env.PROJECT_ID }}/${{
env.SERVICE }}/file-manager-image:${{ github.sha }}
```

Лістинг В.10 – Серверний код для ініціалізації https сервера

```
// Certificate
const privateKey = fs.readFileSync(
  `/etc/letsencrypt/live/${process.env.HOST}/privkey.pem`,
  'utf8'
);
const certificate = fs.readFileSync(
  `/etc/letsencrypt/live/${process.env.HOST}/cert.pem`,
  'utf8'
);
const ca = fs.readFileSync(
  `/etc/letsencrypt/live/${process.env.HOST}/chain.pem`,
  'utf8'
);
const credentials = {
  key: privateKey,
  cert: certificate,
  ca,
};
const httpServer = http.createServer(app);
const httpsServer = https.createServer(credentials, app);
httpServer.listen(80, () => {
  console.log('HTTP Server running on port 80');
});
httpsServer.listen(443, () => {
  console.log('HTTPS Server running on port 443');
});
```

Лістинг В.11 – Агент журналювання в Docker для Docker-оптимізованої ОС

```
macho0863@file-manager-app-instance ~ $ docker ps
CONTAINER ID IMAGE STATUS PORTS NAMES
8f7b014bbf8b    us-central1-docker.pkg.dev/file-manager-application
/file-manager-repository/file-manager-image:
081fc877ae6ea6e81e4e87fdbd5060634685d0b5
```

```
"docker-entrypoint.s…"   20 hours ago   Up 20  klt-file-manager-app-instance-tzxb

df216a413cde        gcr.io/stackdriver-agents/stackdriver-logging-agent:1.9.9 "/entrypoint.sh /usr…"    20 hours ago    Up 20 hours stackdriver-logging-agent
```

Лістинг В.12 – Команди та їхній вивід для процедури ручного розгортання веб-застосунку

```
macho0863@file-manager-app-instance ~ $ date
Thu May 05 07:26:56 UTC 2023

macho0863@file-manager-app-instance ~ $ mkdir app-temp
macho0863@file-manager-app-instance ~ $ cd app-temp/
macho0863@file-manager-app-instance   ~/app-temp   $   git   clone git@github.com:AndrewMacho/file-manager.git
Cloning into 'filemanager'...
remote: Enumerating objects: 7500, done.
remote: Counting objects: 100% (169/169), done.
remote: Compressing objects: 100% (83/83), done.
remote: Total 7500 (delta 48), reused 156 (delta 46), pack-reused 7331
Receiving objects: 100% (7500/7500), 12.55 MiB | 26.28 MiB/s, done.
Resolving deltas: 100% (4208/4208), done.

macho0863@file-manager-app-instance ~/app-temp $  npm test
Now using node v14.17.5 (npm v9.3.1)
> file-manager@1.0.0 test
> jest server
Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.646s
Ran all test suites matching /server/i

macho0863@file-manager-app-instance ~/app-temp $ docker build -t file .  Building 0.7s (13/13) FINISHED
=> [internal] load .dockerignore 0.0s
  => => transferring context: 98B 0.0s
  => [internal] load build definition from Dockerfile 0.0s
  => => transferring dockerfile: 667B 0.0s
  => [internal] load metadata for docker.io/library/node:14-alpine 0.6s
    => [1/8] FROM docker.io/library/node:14-alpine@sha256:434215b487a329c9e867202ff89e704d3a75e554822e07f3e0c0f9e606121b33 0.0s
  => [internal] load build context 0.0s
  => => transferring context: 8.71kB 0.0s
  => CACHED [2/8] WORKDIR /usr/src/my-app 0.0s
  => CACHED [3/8] COPY package*.json ./ 0.0s
  => CACHED [4/8] RUN npm ci 0.0s
```

```
  => CACHED [5/8] COPY . .    0.0s
  => #10 [6/8] RUN npm run build:frontend
    #10 0.641
    #10 0.641 > file-manager@1.0.0 build:frontend /usr/src/my-app
    #10  0.641  >  webpack  --config  webpack.client.config.js  --mode
production
    #10 0.641
    ...
    #10 5.913 WARNING in asset size limit: The following asset(s)
exceed the recommended size limit (244 KiB).
    #10 5.913 This can impact web performance.
    #10 5.913 Assets:
    #10 5.913   index_bundle.js (249 KiB)
    #10 5.913
    #10 DONE 6.0s
  => #11 [7/8] RUN npm run build:backend
#11 0.517
#11 0.517 > file-manager@1.0.0 build:backend /usr/src/my-app
#11 0.517 > babel -d dist server
#11 0.517
#11 1.247 Successfully compiled 24 files with Babel (626ms).
#11 DONE 1.4s

  => CACHED [8/8] RUN apk add certbot 0.0s
  => exporting to image 0.0s
  => => exporting layers 0.0s
  =>                  =>                 writing                 image
sha256:3076da134b54a7d747d9ed4a619a7c136b106cd1e5d93c3ae3e16631ecc
1453f                                          0.0s
  => => naming to docker.io/library/file 0.0s

> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS     NAMES
b222d580ce30   file      "docker-entrypoint.s…"   1 day ago   Up 1
day             pedantic_wing
> $ docker stop b222d580ce30 ⬡ system
b222d580ce30

> $ docker run --net=host -it file ⬡ system

> file-manager@1.0.0 start /usr/src/my-app
> npm run migrate && node dist/index.js
> file-manager@1.0.0 migrate /usr/src/my-app
> node_modules/knex/bin/cli.js --knexfile server/config/knexfile.js
migrate:latest

Working directory changed to /usr/src/my-app/server/config
Already up to date
API key does not start with "SG.".
HTTP Server running on port 80
```

```
macho0863@file-manager-app-instance ~/app-temp $ cd ~ && rm -rf app-
temp && docker system prune -af
Deleted Images:
untagged: gcr.io/gce-containers/konlet:v.0.11-latest
untagged:                                                gcr.io/gce-
containers/konlet@sha256:3d29a0ac0d73bae469f1559213d8a1d3d3a752e79
11d1471955a6d342e133303
deleted:
sha256:73e140dc33e100a5ec3c5346be6543637c2492280515ca08f66843302b5
b330a
deleted:
sha256:690e44a27431b3dcab1358d65375c2288741cad62e3abec1bad545dd252
5aded
deleted:
sha256:b763a3f68365ede303caecb1421453cd0f0be5dfead543406c5fb250007
36a35
deleted:
sha256:1fe8fbac6f7b1118d06ab211858f77314e117536d4b8928fe69a2b12b62
1f840
Total reclaimed space: 74.84MB

macho0863@file-manager-app-instance ~ $ date
Thu May 05 07:55:08 UTC 2023
```