ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Модуль автоматизованого розгортання і моніторингу застосунків на хмарних платформах»

на здобуття освітнього ступеня магістра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Дмитро КУЧЕРЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КСДМ-61
Дмитро КУЧЕРЕНКО
Ім'я, ПРІЗВИЩЕ
Керівник: Ярослав ТОРОШАНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти Магістр

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кучеренко Дмитро Андрійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Модуль автоматизованого розгортання і моніторингу застосунків на хмарних платформах»
керівник кваліфікаційної роботи Ярослав Торошанко, канд.техн.наук, доцент,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320,
2. Строк подання кваліфікаційної роботи «20» грудня 2024р.
3. Вихідні дані до кваліфікаційної роботи: Хмарні технології значно спрощують управління IT-інфраструктурою, забезпечуючи гнучкість, масштабованість і доступність ресурсів. Автоматизація процесів розгортання та моніторингу в хмарному середовищі дозволяє мінімізувати людський фактор, прискорити впровадження змін і забезпечити стабільність роботи застосунків. Розробка модуля для автоматизованого розгортання і моніторингу спрямована на оптимізацію цих процесів, підвищення ефективності використання ресурсів та адаптацію до динамічних потреб сучасних технологій..
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1. Аналіз методів та засобів автоматизованого розгортання і моніторингу застосунків з використанням платформ хмарних обчислень
   4.2. Огляд інструментів платформи хмарних обчислень AWS
   4.3. Розробка модуля автоматизованого розгортання і моніторингу застосунків на хмарних платформах
5. Перелік ілюстративного матеріалу: *презентація*
   5.1. Методи автоматизованого розгортання застосунків
   5.2. Платформи хмарних обчислень
   5.3. Результати роботи модулю автоматизованого розгортання та моніторингу
6. Дата видачі завдання «01» вересня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	10.10-12.10.2024	Виконано
2	Обробка матеріалу	12.10-15.10.2024	Виконано
3	Розробка теоретичної частини	15.10-20.10.2024	Виконано
4	Написання web-застосунку з використанням контейнеризації	20.10-22.10.2024	Виконано
5	Створення та налаштування інструментів AWS	22.10-26.10.2024	Виконано
6	Налаштування автоматизованого розгортання та моніторингу застосунку	26.10-30.10.2024	Виконано
7	Висновки за результатами аналізу	30.10-31.10.2024	Виконано
8	Розробка презентації	31.10-01.11.2024	Виконано
9	Передзахист	09.12-11.12.2024	Виконано
10	Здача в деканат	20.12-29.12.2024	

Здобувач вищої освіти _____ Дмитро КУЧЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ярослав ТОРОШАНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістр: 61 стор., 39 рис., 7 джерел.

Мета роботи – Розробити модуль для автоматизованого розгортання та моніторингу застосунків на хмарних платформах, забезпечивши швидкий та ефективний процес впровадження з використанням контейнеризації, CI/CD та сервісів платформи хмарних обчислень AWS.

Об'єкт дослідження – Автоматизоване розгортання та моніторинг застосунків у хмарному середовищі.

Предмет дослідження – Методи та технології контейнеризації, автоматизації CI/CD процесів та управління хмарними ресурсами для підвищення продуктивності та надійності програмних рішень.

Короткий зміст роботи:

У роботі досліджено підходи до автоматизації розгортання та моніторингу програмних застосунків у хмарних середовищах. Проаналізовано сучасні технології контейнеризації, а також автоматизації тестування та розгортання з використанням CI/CD інструментів. Особливу увагу приділено інтеграції з хмарною платформою AWS для забезпечення стабільності, масштабованості та високої доступності.

У рамках дослідження розроблено модуль з автоматизованого розгортання веб-застосунку з використанням сервісів хмарної платформи AWS(EC2, ECR, ElastiCache та CloudWatch). Проаналізовано сучасні тенденції впровадження хмарних рішень, їхні переваги та недоліки.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, МОНІТОРИНГ, КОНТЕЙНЕРИЗАЦІЯ, ТЕСТУВАННЯ, РОЗГОРТАННЯ, ХМАРНІ ТЕХНОЛОГІЇ, GITHUB, GITLAB, CI/CD, SAAS, AWS, AZURE, GCP

ABSTRACT

Text part of the qualification work for the degree of Master: 61 pages, 39 figures, 7 sources.

Purpose of the work – Development of a module for the automated deployment and monitoring of applications on cloud platforms, ensuring a fast and efficient implementation process through the use of containerization, CI/CD and AWS cloud computing services.

Object of research – Automated deployment and monitoring of applications in a cloud environment.

Subject of research – Methods and technologies of containerization, CI/CD process automation, and cloud resource management to improve the performance and reliability of software solutions.

Summary of the work:

This study examines approaches to automating the deployment and monitoring of software applications in cloud environments. It analyzes modern containerization technologies, as well as testing and deployment automation using CI/CD tools. Particular attention is paid to integration with the AWS cloud platform to ensure stability, scalability, and high availability.

As part of the research, a module for automated deployment of a web application using AWS cloud platform services (EC2, ECR, ElastiCache, and CloudWatch) has been developed. Contemporary trends in the implementation of cloud solutions, their advantages, and their drawbacks are also analyzed.

KEYWORDS: AUTOMATION, MONITORING, CONTAINERIZATION, TESTING, DEPLOYMENT, CLOUD TECHNOLOGIES, GITHUB, GITLAB, CI/CD, SAAS, AWS, AZURE, GCP

ЗМІСТ

ВСТУП

У сучасному світі швидкість розробки, тестування та розгортання програмного забезпечення відіграє ключову роль у конкурентоспроможності компаній та організацій. Традиційні методи роботи з додатками поступаються місцем автоматизованим підходам, які забезпечують надійність, стабільність і оперативність внесення змін. З появою хмарних платформ і сучасних інструментів DevOps значно спростилася організація процесів безперервної інтеграції та доставки (CI/CD), а також моніторингу додатків.

Метою даної дипломної роботи є розробка модуля автоматизованого розгортання і моніторингу додатків у хмарному середовищі. У процесі реалізації були використані популярні інструменти, такі як GitHub Actions для автоматизації процесів, Docker для контейнеризації застосунків та сервіси AWS, зокрема EC2, ECR, ElastiCache і CloudWatch, для хмарної інфраструктури. Це дозволило створити комплексну систему, яка автоматизує всі ключові етапи роботи з додатком: від тестування змін до розгортання в робочому середовищі.

Дана робота має на меті не лише описати процеси автоматизації, але й продемонструвати переваги інтеграції сучасних технологій у повсякденні задачі розробки. На прикладі розробленого веб-застосунку показано, як використання таких підходів може підвищити ефективність командної роботи, зменшити час на обслуговування системи і забезпечити її стабільну роботу. Результати роботи можуть бути використані як основа для впровадження подібних рішень у практиці інших організацій.

1. АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ І МОНІТОРИНГУ ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ ПЛАТФОРМ ХМАРНИХ ОБЧИСЛЕНЬ

Сучасні інформаційні системи та програмні рішення все частіше реалізуються у хмарному середовищі. Це пояснюється зростанням вимог до масштабованості, продуктивності, швидкості доставки нових функціональних можливостей та доступності застосунків. Швидке доставлення нових можливостей клієнту, зменшення часу на відновлення після збоїв , гнучке реагування на коливання навантаження. Завдяки цим підходам організації можуть підвищувати конкурентоспроможність, оперативно адаптуватися під нові вимоги та покращувати якість користувацького досвіду.

1.1. Сучасні підходи до автоматизованого розгортання застосунків

Автоматизація розгортання веб-застосунків є важливою складовою сучасної розробки програмного забезпечення, оскільки вона підвищує ефективність роботи команд, зменшує кількість помилок і прискорює впровадження змін. У швидкому ритмі розвитку цифрових технологій автоматизація стала не лише зручністю, а необхідністю, що дозволяє зосередитися на стратегічних аспектах проектів, залишаючи рутинні завдання машинам.

Одним із ключових елементів автоматизації є використання безперервної інтеграції та безперервного розгортання. Ці процеси дозволяють автоматизувати тестування, перевірку та впровадження змін у кодовій базі, що гарантує їхню якість і скорочує час від ідеї до релізу. Інфраструктура як код додає гнучкості та надійності, забезпечуючи можливість швидкого розгортання серверів і середовищ за допомогою скриптів. Це мінімізує людський фактор та прискорює адаптацію до змін, що є особливо важливим для компаній у динамічних ринкових умовах.

1.2. Контейнеризація як основа автоматизованого розгортання

Контейнеризація стала революцією в підході до розробки та розгортання. Завдяки контейнерам веб-застосунки можуть працювати в ізольованих середовищах, що спрощує їх переносимість між різними платформами та оптимізує використання ресурсів. Це рішення економічно вигідне й ефективне, адже дозволяє запускати декілька контейнерів на одному сервері без втрати продуктивності. Водночас хмарні платформи розширюють ці можливості, надаючи інструменти для масштабування системи, забезпечення її доступності та уникнення простоїв. Інтегровані сервіси моніторингу й логування роблять процес управління ще зручнішим.

Одним із найпопулярніших інструментів для контейнеризації є Docker — платформа відкритим вихідним кодом, для розробки, доставки та запуску додатків. Docker дозволяє відокремлювати додатки від інфраструктури, що дає змогу швидше доставляти програмне забезпечення. Docker дозволяє керувати інфраструктурою так само, як і застосунками. Використовуючи методології Docker для доставки, тестування та розгортання коду, можна суттєво скоротити час між написанням коду та його запуском у виробничому середовищі.

Docker дозволяє упаковувати й запускати додатки в ізольованих середовищах, які називаються контейнерами. Ізоляція та безпека дозволяють одночасно запускати багато контейнерів на одному хості. Контейнери є легкими і містять усе необхідне для роботи додатка, тому немає потреби покладатися на програмне забезпечення, встановлене на хості.

Docker надає інструменти та платформу для управління життєвим циклом контейнерів, можливість розробляти додаток і його компоненти, використовуючи контейнери. Контейнер стає одиницею для розповсюдження і тестування додатка. Коли додаток готовий його можна розгорнути у виробничому середовищі як контейнер або як частину оркестрованої служби. Це працює однаково, незалежно

від того, чи виробничим середовищем є локальним дата-центром, хмарним провайдером або їхньою гібридною комбінацією.

Docker використовує клієнт-серверну архітектуру. Клієнт взаємодіє з демоном Docker, який виконує основні операції з побудови, запуску та розповсюдження контейнерів. Клієнт і демон Docker можуть працювати на одній системі, або клієнт може підключатися до віддаленого демона Docker. Взаємодія між клієнтом і демоном відбувається через REST API, за допомогою UNIX сокетів або мережевого інтерфейсу. Іншим клієнтом Docker є Docker Compose, який дозволяє працювати з додатками, що складаються з набору контейнерів.

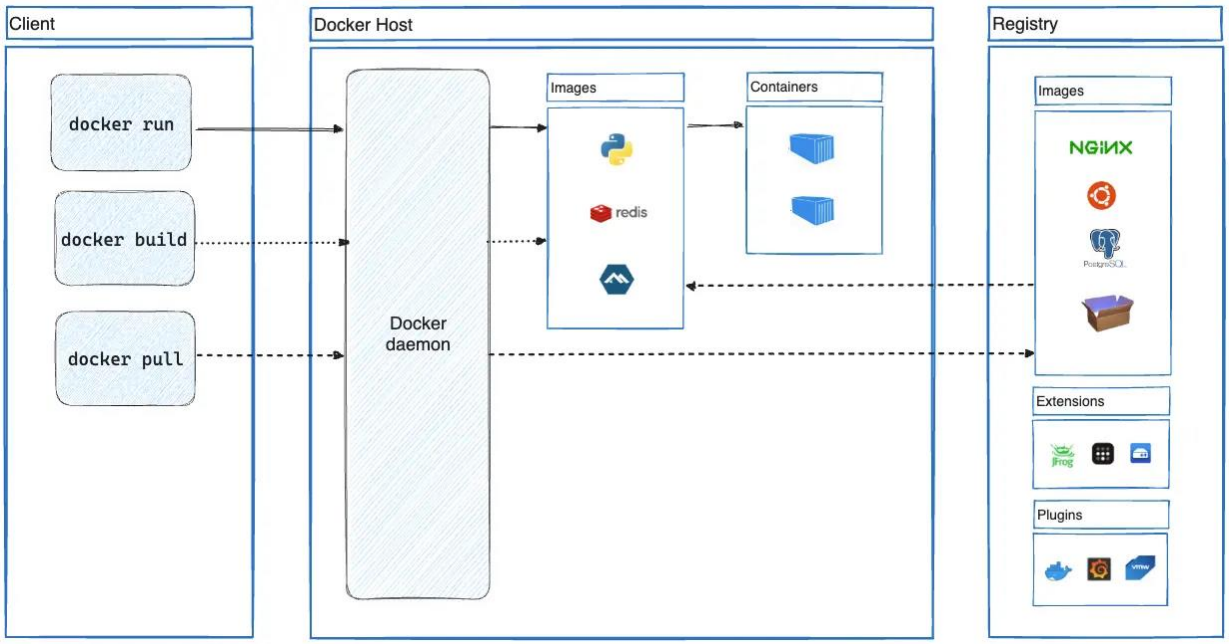

Рисунок 1.1 - клієнт-серверна архітектура Docker

Розглянемо компоненти Docker:

- ***Daemon(Демон)*** - обробляє запити API Docker і управляє об'єктами Docker, такими як образи, контейнери, мережі та томи. Демон також може взаємодіяти з іншими демонами для управління службами Docker.
- ***Docker CLI(Клієнт)*** - є основним інструментом для взаємодії користувачів із Docker. Виконання команд, таких як *docker run*, відправляє запити до

демона, який реалізує ці команди. Клієнт Docker може підключатися до кількох демонів одночасно.

- ***Docker Desktop*** - це зручний додаток для встановлення в середовищах Mac, Windows або Linux, який дозволяє створювати та ділитися контейнеризованими додатками і мікросервісами.
- ***Registries(Реєстри)*** - публічним реєстром є Docker Hub, який доступний усім користувачам за замовчуванням. Можливо також налаштувати власний приватний реєстр.
- ***Images(Образи)*** - це шаблони лише для читання з інструкціями для створення контейнера Docker. Образи можуть бути створені на основі інших образів із додатковою кастомізацією. Створення власних образів відбувається за допомогою Dockerfile — текстового файлу, що містить інструкції для створення образу. Кожна інструкція Dockerfile додає новий шар до образу. Завдяки цьому при зміні Dockerfile перезбираються лише змінені шари, що робить образи легковагими та швидкими у порівнянні з іншими технологіями віртуалізації.
- ***Containers(Контейнери)*** - це виконуваний екземпляр образу. Контейнери можна створювати, запускати, зупиняти, переміщати або видаляти за допомогою API Docker або CLI. Контейнер можна підключати до однієї або кількох мереж, прикріпляти до нього сховище або створювати новий образ на основі його поточного стану. Контейнер за замовчуванням ізольований від інших контейнерів і хост-машини. Рівень ізоляції мережі, сховища або інших підсистем можна налаштувати відповідно до потреб. Контейнер визначається його образом і конфігурацією, яка вказується під час створення або запуску. Усі зміни стану контейнера, які не збережені в постійне сховище, втрачаються після його видалення.

Розглянемо приклад виконання запуску Docker-контейнера, враховуючи що утиліта вже встановлена на локальній машині, виконавши команду ***docker run -i -t***

ubuntu /bin/bash у командному рядку. При виконанні цієї команди відбуваються наступні дії (за умови використання стандартної конфігурації реєстру):

1. Якщо образ ubuntu не доступний локально, Docker завантажує його з налаштованого реєстру. Це еквівалентно команді ***docker pull ubuntu***.
2. Docker створює новий контейнер, що аналогічно виконанню команди ***docker container create***.
3. Для контейнера виділяється файлова система з можливістю запису, яка є фінальним шаром. Це дозволяє контейнеру, що працює, створювати або змінювати файли та каталоги у своїй локальній файловій системі.
4. Docker створює мережевий інтерфейс для підключення контейнера до стандартної мережі, оскільки опції налаштування мережі не були вказані. Це включає призначення IP-адреси контейнеру. За замовчуванням контейнери можуть підключатися до зовнішніх мереж через мережеве з'єднання хост-машини.
5. Docker запускає контейнер і виконує команду ***/bin/bash***. Завдяки інтерактивному режиму (вказується прапорцями ***-i*** і ***-t***), забезпечується можливість вводити команди з клавіатури, а Docker записує вихідні дані в термінал.
6. Коли команда exit завершує виконання /bin/bash, контейнер зупиняється, але не видаляється. Його можна знову запустити або видалити за потреби.

1.3. Автоматизація процесів розгортання та тестування, інструменти CI

Системи безперервної інтеграції відіграють ключову роль у забезпеченні ефективного тестування. Вони автоматизують запуск тестів, збірку та перевірку коду, що дозволяє командам швидше і безпечніше інтегрувати зміни в основну гілку проекту. Завдяки цьому розробники можуть зосередитися на вирішенні бізнес-завдань, а не на рутинних процесах. Розглянемо набори інструментів для автоматизації тестування та розгортання ПО.

1.3.1 Система контролю версій Git

Система контролю версій Git Git — це система контролю версій, яка стала стандартом у розробці програмного забезпечення. Вона дозволяє відстежувати зміни у коді, координувати роботу в команді та зберігати історію розробки. Завдяки своїй розподіленій природі Git забезпечує можливість працювати над проектом незалежно від центрального сховища, що дозволяє розробникам виконувати коміти, працювати з гілками та експериментувати без ризику для основного коду.

Однією з основних переваг Git є його підтримка гілкування. Кожна гілка може бути окремим робочим середовищем, яке дозволяє працювати над новими функціями, виправленням помилок або експериментами, не впливаючи на основний код. Створення, злиття та видалення гілок є простими та швидкими процесами, що сприяє зручній організації роботи.

Інша ключова особливість Git — це здатність працювати з історією змін. Кожен коміт містить детальну інформацію про внесені зміни, автора та часову мітку, що дозволяє легко відстежувати процес розробки, повертатися до попередніх версій або знаходити причину помилок. Це забезпечує прозорість і контроль над проектом.

Git також інтегрується з різноманітними хостинговими платформами, такими як ***GitHub***, ***GitLab*** чи ***Bitbucket***, що спрощує співпрацю в командах. За допомогою цих платформ можна працювати з pull-реквестами, переглядати код інших учасників і обговорювати зміни перед їх злиттям в основну гілку. Це робить Git не лише інструментом контролю версій, а й ефективним середовищем для командної роботи.

1.3.2. GitHub Actions

GitHub Actions — це інструмент для автоматизації робочих процесів безпосередньо в екосистемі GitHub. Він забезпечує можливість створення сценаріїв для автоматичного тестування, збірки та розгортання додатків. GitHub

Actions дозволяє налаштовувати робочі процеси через YAML-файли, які зберігаються у репозиторії проекту. Завдяки цьому будь-яка зміна в коді, така як створення pull request або злиття в основну гілку, може автоматично запускати серію вказаних завдань.

Основні компоненти GitHub Actions включають workflows, events, jobs, actions і runners, розглянемо їх детальніше:

- ***Workflows(Робочі процеси)*** - це налаштований автоматизований процес, що складається з одного чи кількох завдань. Вони визначаються у YAML-файлах, розташованих у каталозі ***.github/workflows*** репозиторію проекта. Робочі процеси запускаються подіями у репозиторії, вручну або за розкладом. Один репозиторій може містити кілька робочих процесів для виконання різних завдань, таких як тестування, деплой або додавання міток.
- ***Events(Події)*** - це конкретна активність у репозиторії яка ініціює виконання робочого процесу. Наприклад, подія може бути викликана створенням pull request, відкриттям issue або внесенням коміту до репозиторію. Робочі процеси також можна запускати за розкладом, через REST API або вручну.
- ***Jobs(Завдання)*** - це набір кроків у робочому процесі, які виконуються на одному runner. Кожен крок може бути скриптом або дією. Кроки виконуються послідовно і можуть обмінюватися даними. За замовчуванням завдання виконуються паралельно, але можна налаштувати залежності між ними, щоб вони виконувались у певному порядку.
- ***Actions(Дії)*** - це індивідуальний компонент, який виконує складні та часто повторювані завдання. Вони допомагають зменшити кількість коду у файлах робочих процесів. Дії можуть виконувати завдання, як-от клонування репозиторію, налаштування середовища або аутентифікація до хмарного провайдера.
- ***Runners(Виконавці)*** - це сервер, що виконує завдання у робочих процесах. Кожен виконавець може виконувати лише одне завдання одночасно. GitHub надає готові виконавці для Ubuntu, Windows та macOS, кожен з яких

запускається у новій віртуальній машині. Також можна налаштувати власні виконавці для специфічних потреб.

Отже, GitHub Actions — це потужний і гнучкий інструмент автоматизації, який дозволяє ефективно організовувати процеси розробки, тестування та розгортання програм. Його компоненти — workflows, events, jobs, actions і runners — надають все необхідне для побудови масштабованих CI/CD процесів.

1.3.3. GitLab CI/CD

GitLab CI/CD — це інтегрована система для безперервної інтеграції та доставки, яка дозволяє автоматизувати процеси розробки, тестування та розгортання коду. Вона тісно інтегрована з GitLab і використовує .gitlab-ci.yml файл для визначення робочих процесів. Розглянемо основні компоненти системи:

- ***Job (Завдання)*** - це основна одиницями роботи у конвеєрі. Кожне завдання виконує конкретну дію, визначену в секції *script*. Завдання можуть використовувати артефакти для передачі результатів виконання між етапами, а також мати залежності від інших завдань. Кожне завдання виконується на окремому виконавці може завершитись успішно, з помилкою або бути пропущеним.
- ***Pipeline (Конвеєр)*** - основний елемент GitLab CI/CD. Він складаються з одного чи кількох завдань, згрупованих у послідовні етапи, такі як збірка, тестування або розгортання. Конвеєри запускаються після подій у репозиторії, таких як коміт, створення merge request або за розкладом. Кожне завдання виконує певну дію, використовуючи команди, визначені у параметрі script, і може передавати результати роботи через артефакти.
- ***Stages(Emanu)*** - Завдання групуються в етапи, які виконуються послідовно. Завдання в одному етапі можуть виконуватись паралельно, що пришвидшує процеси. Наприклад, після успішної збірки програми етап тестування перевіряє її на працездатність. Етапи забезпечують чітку організацію

роботи, і ви можете створювати власні етапи, адаптуючи їх під ваші потреби.

- ***Runners(Виконавці)*** - це сервери, які виконують завдання конвеєрів. GitLab пропонує Shared Runners(спільні виконавці) або можливість налаштувати Specific Runners(власні), що підтримують різні середовища, такі як Linux, Windows, macOS чи Docker. Виконавці забезпечують ізольоване середовище для кожного завдання, що гарантує стабільність виконання. Середовища визначають, де саме запускається програма, наприклад, у staging чи production, і дозволяють автоматизувати розгортання.
- ***Artifacts(Артефакти)*** - це результати виконання завдань, наприклад, файлами збірки чи журналами тестування, які передаються між етапами. Кеш зберігає тимчасові файли, такі як залежності, що пришвидшує повторне виконання завдань. Для запуску конвеєрів використовуються тригери - вони дозволяють запускати процеси через API, вручну або за розкладом.

Отже, GitLab CI/CD — це гнучка система для автоматизації розробки, яка інтегрується у GitLab і забезпечує повний цикл від тестування до розгортання. Завдяки своїм компонентам, включаючи завдання, етапи, виконавців і артефакти, вона дозволяє ефективно керувати процесами розробки, тестування та доставки коду, оптимізуючи роботу над проєктами будь-якого масштабу.

2. ОГЛЯД ІНСТРУМЕНТІВ ПЛАТФОРМИ ХМАРНИХ ОБЧИСЛЕНЬ AWS

У процесі розробки системи автоматизованого розгортання та моніторингу додатків вибір хмарної платформи є одним із ключових рішень. Для цієї роботи була обрана платформа Amazon Web Services (AWS), яка є лідером на ринку хмарних обчислень завдяки своїй надійності, масштабованості та широкому набору інструментів для вирішення завдань будь-якої складності. AWS пропонує інтегровані сервіси для обчислень, зберігання даних, моніторингу, безпеки та інших аспектів, які є важливими для сучасних додатків.

Обрання AWS було зумовлене декількома факторами. По-перше, платформа має глобальну інфраструктуру, яка забезпечує високу доступність і мінімальну затримку доступу до ресурсів. По-друге, AWS надає широкий вибір сервісів, які дозволяють реалізувати всі аспекти CI/CD-процесів, зокрема Amazon EC2 для обчислень, Amazon ECR для роботи з контейнерними образами, ElastiCache для кешування даних і CloudWatch для моніторингу. Крім того, AWS забезпечує гнучке налаштування безпеки через IAM (Identity and Access Management) та інтеграцію з інструментами автоматизації, такими як GitHub Actions.

Цей розділ присвячений детальному огляду ключових сервісів AWS, які були використані в рамках дипломної роботи, та їхньої ролі в побудові системи автоматизації та моніторингу. Завдяки використанню AWS вдалося створити стабільну, продуктивну та безпечну інфраструктуру, що відповідає сучасним вимогам до хмарних додатків.

2.1. Загальний огляд архітектури та принципів роботи AWS

Amazon Web Services (AWS) — це платформа хмарних обчислень, яка пропонує широкий спектр сервісів для розробки, розгортання та підтримки додатків. Вона базується на глобальній інфраструктурі, яка включає центри обробки даних (Data Centers), розподілені по всьому світу, що забезпечує високу доступність, масштабованість і надійність. Архітектура AWS будується навколо ключових принципів хмарних обчислень:

- ***Інфраструктура як послуга (IaaS)*** - надання базових ресурсів, таких як віртуальні сервери (Amazon EC2), сховища даних (Amazon S3) та мережеві компоненти.
- ***Платформа як послуга (PaaS)*** - сервіси, які спрощують розробку додатків, наприклад, Amazon RDS для роботи з базами даних або AWS Lambda для виконання коду без управління інфраструктурою.

- ***Програмне забезпечення як послуга (SaaS)*** - готові рішення, які можуть бути інтегровані в бізнес-процеси, такі як Amazon WorkSpaces для віртуальних робочих місць.

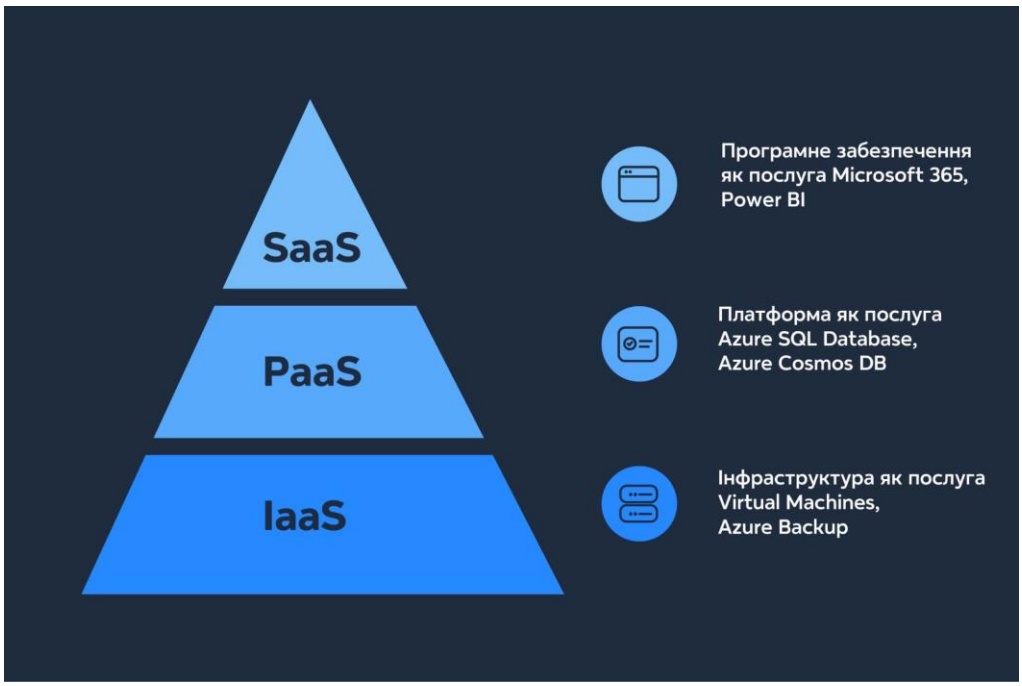

Рисунок 2.1 - модель хмарних сервісів: SaaS, PaaS та IaaS

AWS має три основні рівні інфраструктури:

- ***Регіони (Regions)*** - географічні області, які містять кілька зон доступності. Наприклад, us-east-1 (Північна Вірджинія).
- ***Зони доступності (Availability Zones, AZs)*** - окремі дата-центри в межах одного регіону, які забезпечують високу доступність ресурсів.
- ***Edge Locations*** - географічні точки для оптимізації доставки контенту через сервіси, такі як Amazon CloudFront.

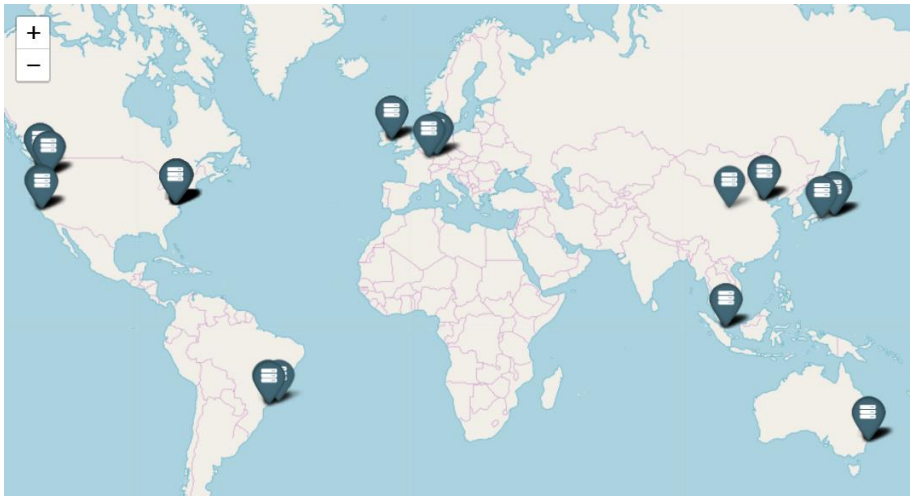

Рисунок 2.2 - мапа дата центрів AWS

Сервіси зберігання та обчислень:

- ***Amazon EC2*** - віртуальні сервери для запуску додатків із можливістю масштабування залежно від навантаження.
- ***Amazon S3*** - сервіс для зберігання даних будь-якого обсягу з високою доступністю.
- ***Amazon ECR*** -репозиторій для зберігання контейнерних образів.

Мережеві сервіси:

- ***Amazon VPC*** - віртуальні приватні мережі для ізоляції ресурсів.
- ***Elastic Load Balancing*** - Автоматичний розподіл трафіку між серверами для забезпечення стабільності додатків.

Моніторинг та управління

- ***Amazon CloudWatch*** - сервіс для збору метрик і моніторингу додатків та інфраструктури.
- ***AWS IAM*** -інструмент для управління доступом до ресурсів.

Принципи роботи AWS:

- ***Масштабованість*** - AWS дозволяє автоматично змінювати ресурси залежно від потреб додатка.

- ***Оптимізація витрат*** - платформа використовує модель “Pay-as-you-go”, де клієнт платить лише за використані ресурси.
- ***Безпека*** - AWS забезпечує багаторівневий захист даних, включаючи шифрування, управління доступом через IAM, аудит і моніторинг.
- ***Надійність*** - завдяки розподіленій архітектурі, сервіси AWS залишаються доступними навіть у разі збоїв у окремих дата-центрах.

2.2. Керування доступом та безпекою IAM

Amazon Identity and Access Management (IAM) — це сервіс AWS, що забезпечує безпечний і гнучкий контроль доступу до ресурсів AWS. IAM дозволяє створювати і керувати користувачами, групами, ролями та політиками доступу, забезпечуючи багаторівневий захист інфраструктури.

Основні компоненти ***IAM***:

- ***Користувачі (Users)*** - IAM дозволяє створювати облікові записи користувачів для фізичних осіб або додатків, які потребують доступу до ресурсів AWS. Кожен користувач отримує унікальні облікові дані, такі як ключі доступу або пароль, для автентифікації. Користувачі можуть бути асоційовані з конкретними політиками доступу.
- ***Групи (Groups)*** - спрощують управління доступом, дозволяючи об'єднувати користувачів із подібними правами. Політики доступу застосовуються до групи, і всі її учасники отримують ті самі дозволи.
- ***Ролі (Roles)*** - дозволяють призначати тимчасові права доступу сервісам AWS або стороннім додаткам без надання постійних облікових даних. Це особливо корисно для сценаріїв, коли один сервіс AWS (наприклад, Lambda) повинен взаємодіяти з іншим (наприклад, S3).
- ***Політики (Policies)*** - це JSON-документи, які визначають дозволи на доступ до ресурсів AWS. Політики можуть бути керованими (AWS або користувачем) або inline-політиками, що прив'язуються до окремих ресурсів.

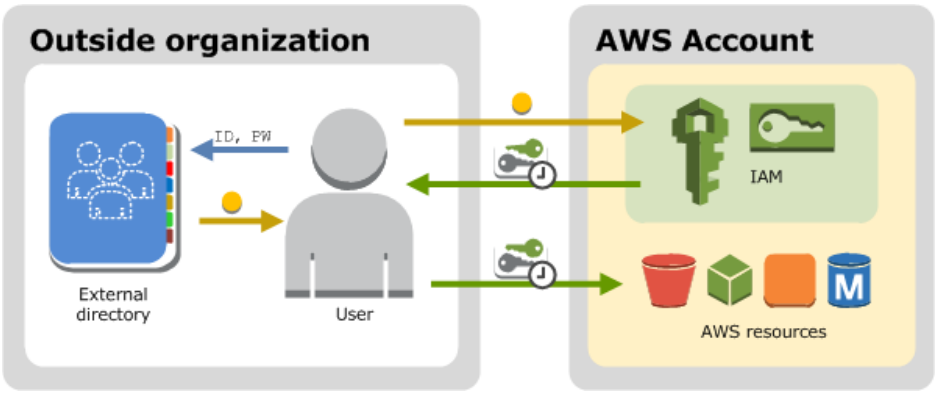

Рисунок 2.3 - Схема роботи сервісу AMI

Безпека та управління доступом:

- ***Принцип мінімального доступу (Least Privilege)*** - IAM дозволяє надавати користувачам, групам чи ролям тільки ті права, які необхідні для виконання їхніх завдань. Це зменшує ризик ненавмисного доступу до критичних ресурсів.
- ***Багатофакторна автентифікація (MFA)*** - AWS підтримує використання багатофакторної автентифікації для підвищення рівня безпеки. Користувачі повинні вводити не лише пароль, а й одноразовий код з мобільного пристрою.
- ***Журналювання і аудит Сервіс AWS CloudTrail*** - дозволяє відстежувати всі дії, пов'язані з IAM, такі як зміна політик чи створення нових користувачів. Це важливий інструмент для забезпечення прозорості та відповідності вимогам безпеки.
- ***Тимчасові ключі доступу IAM*** - підтримує використання тимчасових облікових даних для ролей і сервісів. Це дозволяє уникати довготривалого зберігання секретів і знижує ризик їх компрометації.

Отже, IAM є ключовим інструментом для керування доступом і забезпечення безпеки в AWS. Завдяки гнучкості та потужним інструментом IAM дозволяє створювати ефективні та безпечні механізми доступу, що відповідають сучасним вимогам до управління хмарними ресурсами. Дотримання принципів

безпеки та регулярний моніторинг дозволяють мінімізувати ризики й захистити критичні дані та сервіси.

2.3. Обчислювальні ресурси EC2

Amazon Elastic Compute Cloud (EC2) — це основний сервіс AWS, який надає масштабовані обчислювальні ресурси у хмарному середовищі. Він дозволяє запускати віртуальні сервери (“інстанси”), налаштовувати їх під конкретні потреби і оплачувати тільки за фактичне використання ресурсів.

Основні особливості EC2:

- ***Різноманітність*** - EC2 пропонує широкий вибір конфігурацій обчислювальних ресурсів (процесор, оперативна пам'ять, накопичувач), що дозволяє оптимізувати продуктивність і витрати. Типи інстансів варіюються від економічних (наприклад, T-серія) до високопродуктивних (наприклад, M-, C- чи GPU-інстанси).
- ***Масштабованість*** - користувачі можуть автоматично змінювати кількість і тип інстансів залежно від навантаження додатків. Це дозволяє ефективно використовувати ресурси під час пікових періодів і зменшувати витрати в часи простою.
- ***Опціональність*** - інстанси EC2 можуть використовувати різні типи дискових сховищ, такі як Elastic Block Store (EBS) для постійного зберігання даних або Instance Store для тимчасових даних.
- ***Інтеграція з іншими сервісами AWS*** - EC2 легко інтегрується з Amazon S3 для зберігання файлів, Amazon RDS для баз даних і CloudWatch для моніторингу та автоматизації.

Принципи роботи:

- ***Запуск інстансу*** - користувачі можуть запускати інстанси EC2, вибираючи з попередньо налаштованих Amazon Machine Images (AMI), які включають операційну систему і додаткове програмне забезпечення.
- ***Налаштування мережі*** - інстанси EC2 розгортаються у віртуальних приватних хмарах (VPC), що забезпечує ізоляцію мережі та контроль доступу.
- ***Безпека*** - EC2 використовує Security Groups для контролю вхідного і вихідного трафіку, а також підтримує використання ключів SSH для безпечного доступу.

Отже, EC2 є універсальним і потужним інструментом для роботи з обчислювальними ресурсами у хмарі. Завдяки гнучким налаштуванням, масштабованості та інтеграції з іншими сервісами AWS, EC2 дозволяє задовольняти потреби широкого спектра додатків — від невеликих тестових середовищ до складних високонавантажених систем. Використання різноманітних моделей оплати та типів інстансів дозволяє оптимізувати витрати, забезпечуючи при цьому необхідний рівень продуктивності та надійності.

2.4. Контейнерний реєстр ECR

Amazon Elastic Container Registry (ECR) — це повністю керований сервіс AWS, призначений для зберігання, управління та розповсюдження контейнерних образів. Він інтегрується з іншими сервісами AWS та інструментами контейнеризації, такими як Docker, забезпечуючи безперебійну роботу в середовищі DevOps.

Основні особливості ECR:

- ***Безпечне зберігання образів*** - ECR автоматично шифрує збережені контейнерні образи за допомогою AWS Key Management Service (KMS), що забезпечує високий рівень безпеки даних.
- ***Інтеграція з AWS*** - ECR легко інтегрується з Amazon ECS, EKS та AWS Lambda, що спрощує процес розгортання контейнерів у хмарі.

- ***Глобальна доступність*** - завдяки використанню регіонів AWS, ECR дозволяє швидкий і надійний доступ до контейнерних образів у будь-якій точці світу.
- ***Оптимізація витрат*** - оплата проводиться лише за фактично використане сховище та передачу даних, що робить сервіс економічно ефективним.

ECR є важливим компонентом екосистеми AWS, який значно спрощує управління контейнерними образами та забезпечує їхню доступність для розгортання. Завдяки інтеграції з іншими сервісами AWS, ECR є ідеальним вибором для сучасних додатків, які базуються на контейнерній архітектурі. Гнучкість у налаштуванні доступу та автоматизація безпекових перевірок роблять ECR зручним і безпечним рішенням для розробників.

2.5. Масштабування та кешування даних ElastiCache

Amazon ElastiCache — це керований сервіс AWS, який забезпечує високошвидкісне кешування даних у пам’яті. Він дозволяє прискорювати роботу додатків, зменшуючи навантаження на базу даних і забезпечуючи низьку затримку доступу до часто використовуваних даних.

Основні особливості ElastiCache:

- ***Підтримка Redis і Memcached*** - ElastiCache дозволяє вибирати між Redis та Memcached як механізми кешування, залежно від вимог додатка.
- ***Масштабованість*** - сервіс підтримує горизонтальне масштабування, додаючи вузли до кластеру, і вертикальне масштабування, збільшуючи ресурси вузлів.
- ***Автоматизація управління*** - ElastiCache автоматично виконує резервне копіювання, моніторинг і оновлення програмного забезпечення, зменшуючи операційні витрати.
- ***Низька затримка*** - Завдяки зберіганню даних у пам’яті ElastiCache забезпечує доступ до даних із мінімальною затримкою.

Принципи роботи:

- ***Створення кластеру*** - користувачі можуть налаштувати кластер із необхідною кількістю вузлів, типами ресурсів і параметрами кешування.
- ***Інтеграція з додатками*** - ElastiCache легко інтегрується з додатками через стандартні клієнтські бібліотеки Redis або Memcached.
- ***Моніторинг і масштабування*** - Сервіс підтримує моніторинг продуктивності через Amazon CloudWatch і дозволяє додавати або видаляти вузли залежно від потреб.
- ***Безпека*** - ElastiCache працює у межах Amazon VPC, використовуючи Security Groups для контролю доступу, і підтримує шифрування даних у русі.

ElastiCache є ефективним рішенням для додатків, які потребують швидкого доступу до даних і мінімального навантаження на базу даних. Завдяки підтримці масштабування, автоматизації управління та інтеграції з іншими сервісами AWS, ElastiCache дозволяє розробникам створювати продуктивні й масштабовані системи для роботи з великими обсягами даних.

2.6. Моніторинг та логування CloudWatch

Amazon CloudWatch — це сервіс AWS для моніторингу, збору та аналізу метрик і журналів з додатків та інфраструктури. Він дозволяє відстежувати продуктивність, встановлювати оповіщення та автоматизувати реакції на події в реальному часі.

Особливості CloudWatch:

Моніторинг метрик - CloudWatch збирає метрики з ресурсів AWS, таких як EC2, RDS, Lambda, та інших, надаючи детальну інформацію про їхню продуктивність.

Журналювання - сервіс зберігає журнали додатків, операційної системи та інших процесів, що допомагає аналізувати події та виявляти проблеми.

Оповіщення - CloudWatch дозволяє налаштовувати оповіщення на основі метрик і реагувати на них, наприклад, запускати скрипти або надсилати повідомлення через Amazon SNS.

Автоматизація дій - використання CloudWatch Events дозволяє створювати автоматизовані реакції на певні події, наприклад, масштабування ресурсів при перевищенні порогів завантаження.

Принципи роботи:

- ***Збір даних*** - CloudWatch автоматично збирає метрики з підтримуваних сервісів AWS, а також дозволяє користувачам надсилати власні метрики через API.
- ***Аналіз метрик*** - Зібрані дані відображаються у вигляді графіків і діаграм, що допомагає ідентифікувати аномалії та оптимізувати роботу системи.
- ***Управління журналами*** - CloudWatch Logs дозволяє централізовано зберігати та аналізувати журнали додатків, виявляючи помилки чи потенційні загрози.
- ***Налаштування оповіщень*** - Користувачі можуть створювати правила, які запускають оповіщення чи автоматизовані дії, якщо певні метрики досягають критичних значень.

Amazon CloudWatch є незамінним інструментом для моніторингу та управління інфраструктурою в AWS. Завдяки своїм функціям, таким як збір метрик, аналіз журналів та автоматизація дій, CloudWatch допомагає забезпечити стабільність, продуктивність і надійність хмарних додатків.

3. РОЗРОБКА МОДУЛЯ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ І МОНІТОРИНГУ ЗАСТОСУНКІВ НА ХМАРНИХ ПЛАТФОРМАХ

3.1. Розробка та тестування WEB-додатку

У цьому розділі розглянуто процес розробки та тестування невеликого веб-застосунку, реалізованого з використанням фреймворку Sinatra, а також підходи до інтеграції з сервісом кешування Redis. Створений застосунок виконує просте завдання – лічить кількість відвідувань сторінки, використовуючи лічильник, що зберігається у базі даних Redis. Такий приклад дозволяє продемонструвати базові принципи побудови web-додатків, інтеграції зі сторонніми сервісами, а також налаштування автоматизованого тестування.

Для розробки веб-застосунку було обрано фреймворк ***Sinatra*** – мініатюрний та гнучкий фреймворк для мови програмування Ruby, який дозволяє швидко та просто створювати веб-сервіси та REST API. Головною перевагою Sinatra є мінімалістичний підхід: він не нав’язує складну структуру директорій та конфігурацій, а надає розробнику свободу будувати додаток за власним дизайном. Фреймворк побудований на Rack - бібліотеці з набором стандартів для веб-серверів і фреймворків на Ruby. Rack працює за принципом проміжного програмного забезпечення, дозволяючи додавати додаткові рівні обробки запитів, як-от логування, кешування чи автентифікація, перед тим як запит потрапить до вашого додатка. По замовчуванню в Sinatra використовується веб-сервер Puma, використовуватимемо його і надалі.

3.1.1. Структура та залежності додатку

Для забезпечення чистої архітектури та легкості розширення було обрано наступну структуру проекту:

- **app.rb** – основний файл, що містить клас застосунку, контролер маршруту та логіку обробки HTTP-запитів у кореневій директорії.
- **config.ru** - файл який використовується для налаштування веб-сервера.

- **config/** – директорія для конфігурацій та налаштувань.
- **views/** – директорія для представлень.
- **spec/** – директорія для тестів.

Для керування залежностями використовуємо інструмент ***Bundler***, який дозволяє прозоро керувати всіма необхідними бібліотеками та забезпечити повторюваність середовища розробки. Основні бібліотеками використанні з розробці WEB-застосунку є:

- ***sinatra*** - згаданий раніше фреймворк для Ruby.
- ***rackup*** - інструмент для запуску Rack-додатків, який дозволяє легко підняти сервер і взаємодіяти з вашим додатком.
- ***puma*** - високопродуктивний багатопотоковий веб-сервер, оптимізований для розгортання Ruby-додатків.
- ***redis*** - клієнт для підключення до Redis — високошвидкісної бази даних типу key-value, часто використовується для кешування чи зберігання сесій.
- ***rspec*** - сучасний фреймворк для тестування Ruby-коду, що дозволяє писати структуровані та читабельні тести.

3.1.2. Контейнеризація додатку для локального та віддаленого розгортання

Для забезпечення контейнеризації та оркестрації компонентів додатка використовується Docker. Dockerfile визначає, як створюється контейнер для Ruby-додатка, включаючи залежності та конфігурації середовища. docker-compose.yml забезпечує оркестрацію декількох контейнерів, таких як додаток та Redis, для організації їхньої взаємодії, визначення мережевих портів, а також управління персистентними даними.

Ці файли дозволяють стандартизувати процеси розгортання, забезпечити ізоляцію середовища та спрощують роботу з багатокомпонентними додатками.

```
1. FROM ruby:3.3.6
2.
3. WORKDIR /app
4. COPY . /app
5. RUN bundle install
6.
7. EXPOSE 4567
8.
9. CMD ["bundle", "exec", "rackup", "--host", "0.0.0.0", "-p", "4567"]
```

Лістинг файлу Dockerfile

Розглянемо Dockerfile:

- ***FROM ruby:3.3.6*** - Використовується офіційний образ Ruby версії 3.3.6 як базовий для додатка.
- ***WORKDIR /app*** - Встановлюється робоча директорія /app у контейнері, де будуть виконуватись команди.
- ***COPY . /app*** - Копіює всі файли та каталоги з поточної директорії на хост-машині до робочої директорії контейнера.
- ***RUN bundle install*** - Виконує встановлення залежностей, вказаних у Gemfile, за допомогою Bundler.
- ***EXPOSE 4567*** - Вказує, що контейнер прослуховуватиме порт 4567 (використовується Sinatra-додатком).
- ***CMD ["bundle", "exec", "rackup", "--host", "0.0.0.0", "-p", "4567"]*** - Встановлює команду за замовчуванням для запуску Rack-додатка з доступом на всіх мережевих інтерфейсах (0.0.0.0) і порті 4567.

Розглянемо конфігураційний файл для Docker Compose:

```
1. services:
2.  redis:
3.    image: redis:latest
4.    restart: always
5.    ports:
6.      - "6379:6379"
7.    command: redis-server --save 15 1 --loglevel warning
8.    volumes:
9.      - ./redisdata:/data
```

```
  app:
    build:
      context: .
      dockerfile: ./Dockerfile
    tty: true
    stdin_open: true
    depends_on:
      - redis
    ports:
      - "4567:4567"
    environment:
      REDIS_URL: redis://redis:6379
volumes:
  redisdata:
```

Лістинг файлу docker-compose.yml

У цьому файлі описані два сервіси: перший використовується для запуску бази даних Redis, другий - базується на Dockerfile і містить сам Ruby-додаток, створений на основі Sinatra.

Сервіс ***Redis***:

- Використовує офіційний образ ***redis:latest*** для запуску сервера Redis.
- Відкриває порт 6379 на хості для взаємодії з Redis.
- Налаштовує збереження даних у локальній директорії через volume ./redisdata:/data, щоб забезпечити персистентність даних.
- Задає параметри конфігурації Redis (***--save 15 1 --loglevel warning***), щоб зменшити обсяг логів і налаштувати автозбереження.

Сервіс ***App***:

- Будує контейнер на основі ***Dockerfile***, описаного раніше.
- Встановлює залежність від сервісу ***redis*** через ***depends_on***, щоб додаток запускався тільки після доступності другого сервісу.
- Відкриває порт 4567 для доступу до веб-додатка на Sinatra.
- Передає змінну середовища ***REDIS_URL***, яка вказує на розташування сервісу Redis.

Ці конфігураційні файли забезпечують повну автоматизацію побудови, налаштування та запуску додатка разом із залежним сервісом Redis. Вони

сприяють стандартизації розгортання додатка, підвищенню стабільності та простоті обслуговування системи.

3.1.3. Тестування роботи WEB-додатку

Попередньо встановивши Docker та Docker Compose, для тестування роботи локально розгорнемо додаток, створивши образи сервісів вказаних у файлі ***docker-compose.yml*** виконавши команду ***docker compose up*** через командний рядок знаходячись у кореневій директорії проекту:

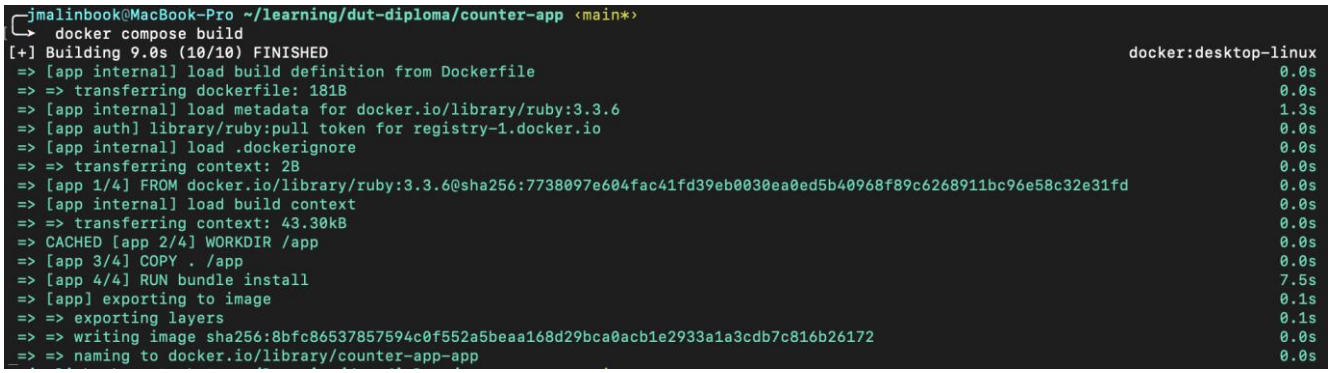

Рисунок 3.1. Створення образів сервісів проекту

Після успішного збирання проекту, запустимо його виконавши командою ***docker compose up.***

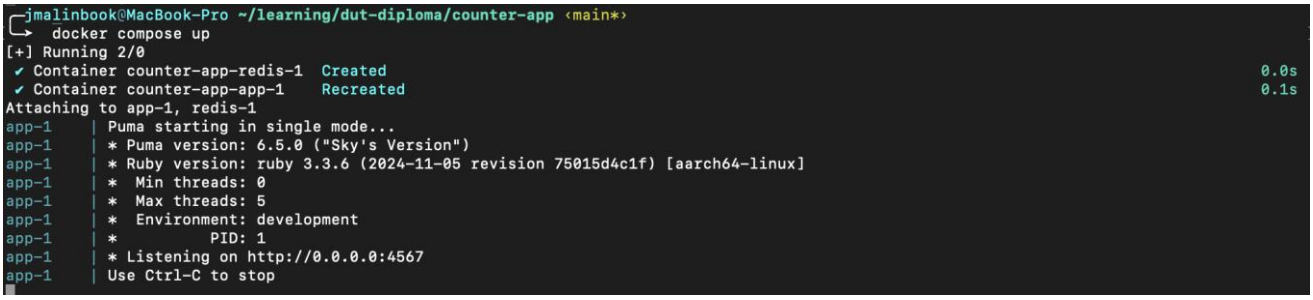

Рисунок 3.2. Створення образів сервісів проекту

Команда запустила два контейнери: ***redis*** і ***app***. Контейнер redis створений і працює на порту ***6379***, готовий до взаємодії. Контейнер app був оновлений і запущений на сервері Puma, доступному за адресою ***http://localhost:4567***. У консолі відображено повідомлення про успішний запуск додатка і його готовність

до роботи. Процес роботи сервера можно перервати використавши комбінацію клавіш **Ctrl + C**. Перейдемо на адресу на якій повинен знаходитись додаток.

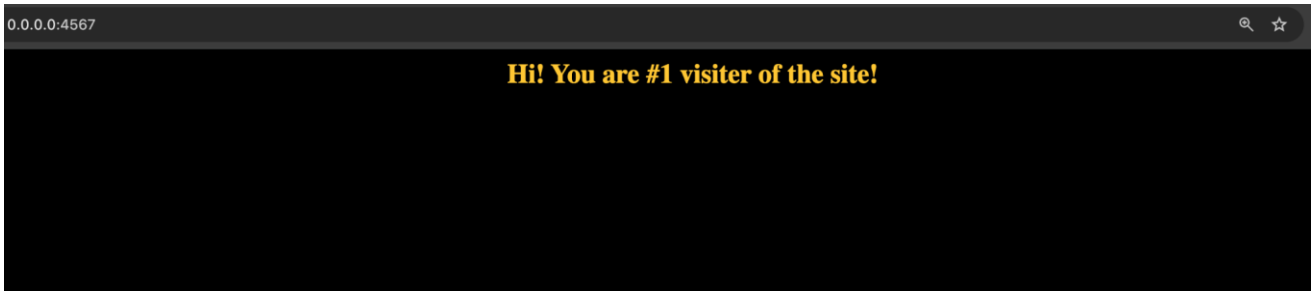

Рисунок 3.3. Вигляд головної сторінки WEB-додатку

В результаті відвідування адреси, і виконавши певні маніпуляції переконуємось у працездатності контейнеризованного застосунку.

3.2. Налаштування автоматизованого тестування

Для управління версіями проекту та інтеграції з інструментами автоматизації виконаємо ініціалізацію репозиторію Git і завантаження проекту на GitHub. Це дозволяє централізовано зберігати код, забезпечувати командну роботу та налаштовувати CI/CD процеси за допомогою GitHub Actions.

3.2.1. Ініціалізація GitHub репозиторію

Послідовно виконуємо команди:

- ***git init*** - команда створює новий локальний репозиторій у кореневій директорії проекту, дозволяючи відстежувати зміни файлів.
- ***git add .*** - додаємо всі файли проекту до індексації для відстеження у Git.
- ***git commit -m "Initial"*** - зберігаємо поточний стан файлів у репозиторії з поясненням змін.

Наступним кроком є створення віддаленого репозиторію на GitHub.

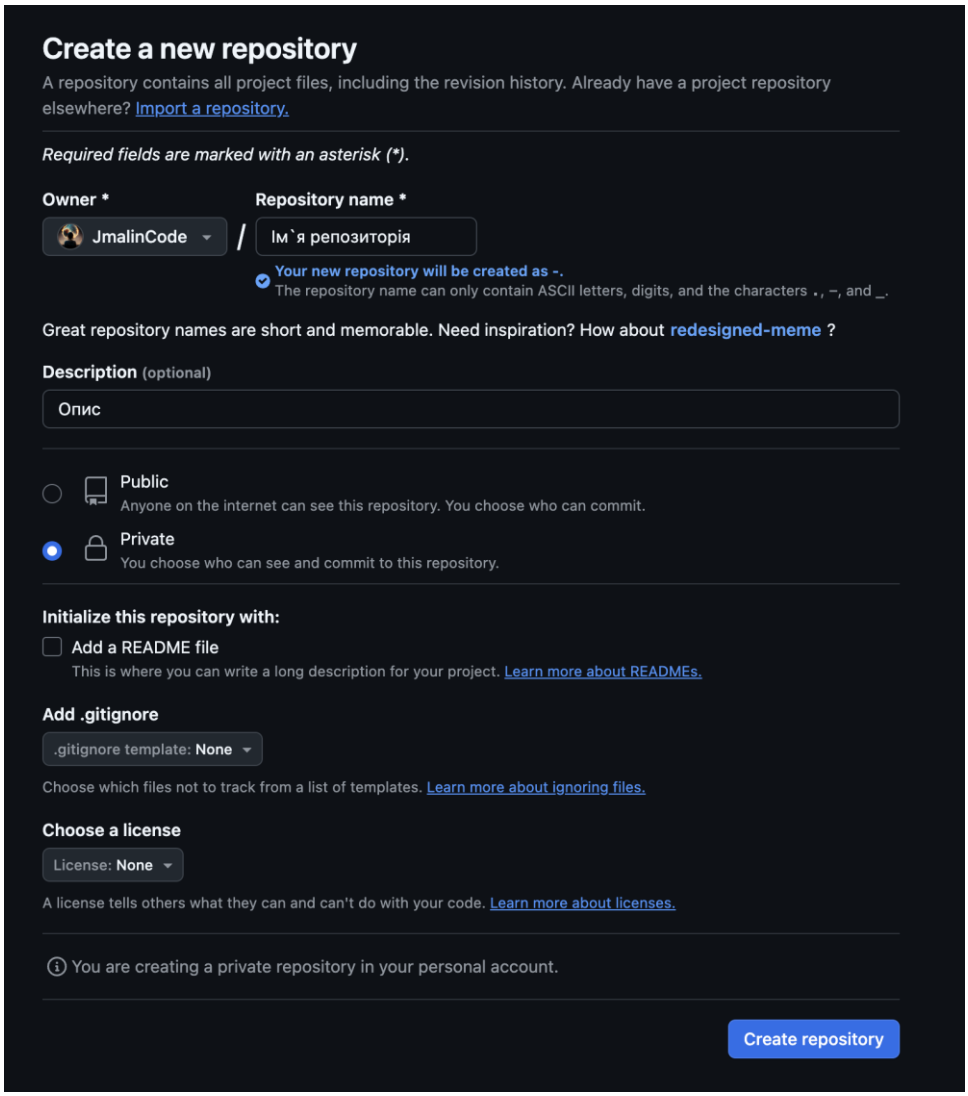

Рисунок 3.4. Сторінка створення репозиторію у веб-ресурсі GitHub

Після заповнення полів ім`я та опису репозиторія, натискаємо кнопку ***Create repository***. Потряпляємо на головну сторінку репозиторія, де нам запропоновано завантажити проект, і інструкція для цього. У командному рядку виконуємо команди з надоної іструкції:

- ***git remote add origin git@github.com:JmalinCode/counter-app.git*** - додає віддалений репозиторій до локального Git-репозиторію, після виконання його можна використовувати для завантаження та синхронізації змін.
- ***git push -u origin main*** - завантажуємо локальну гілку ***main*** до віддаленого репозиторію ***origin*** встановлюючи її як основну гілку для синхронізації.

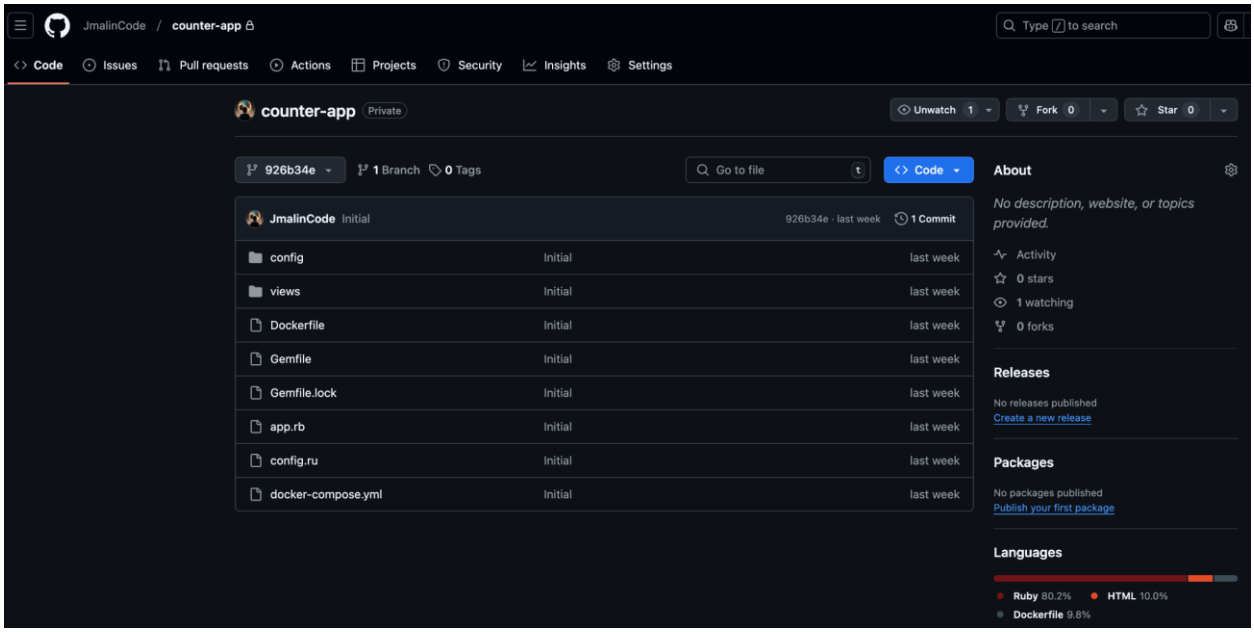

Рисунок 3.4. Сторінка новоствореного репозиторію у веб-ресурсі GitHub

3.2.2. Налаштування GitHub Actions Workflow

Автоматизація процесів розробки є важливим аспектом сучасного програмування. Використання GitHub Actions дозволяє автоматизувати ключові етапи розробки, такі як тестування, збірка або розгортання додатка, забезпечуючи стабільність і контроль якості. Налаштування workflow дозволяє гарантовано виконувати тестування коду щоразу, коли вносяться або пропонуються нові зміни коду забезпечуючи його безперервну інтеграцію.

Workflow створюється у YAML-файлі, який зберігається у директорії ***.github/workflows*** репозиторію.

```yaml
name: Application Build

on:
 push:
 pull_request:
   types: [opened, reopened]

jobs:
 build:
   runs-on: ubuntu-latest
   container:
     image: ruby:3.3.6
     env:
       APP_ENV: test

   steps:
```

```
- uses: actions/checkout@v4

- name: Install requirements
  run: |
    bundle install

- name: Run tests
  run: bundle exec rspec
```

Лістинг файлу .github/workflows/build.yml

Workflow запускається автоматично за певних умов: коли виконується push до репозиторію або відкривається/перевідкривається pull request. GitHub перевіряє тригери, зазначені у конфігурації workflow, і активує його, якщо умови виконуються.

Завдання виконується на платформі ***ubuntu-latest***, де автоматично розгортається контейнер з образом Ruby 3.3.6. Контейнер створює ізольоване середовище для запуску процесів, що гарантує однаковість умов для тестування та виконання додатка.

У середовищі контейнера встановлюються всі необхідні бібліотеки та пакети, зазначені в Gemfile. Цей етап гарантує наявність всіх інструментів, потрібних для роботи додатка та виконання тестів.

Після виконання тестів результати записуються у лог-файли, які стають доступними у вкладці "Actions" на GitHub. Якщо всі етапи успішно завершені, Workflow завершується із позначкою "успішно". У разі помилки, Workflow припиняє виконання, і розробники отримують сповіщення про проблему.

Цей workflow є основою для забезпечення стабільної розробки та швидкого виявлення проблем у коді.

3.3. Налаштування сервісів AWS

Для основи модуля розгортання та моніторингу застосунків було обрано платформу AWS - через її надійність, масштабованість і широкий набір хмарних

сервісів, які відповідають різним вимогам сучасних додатків. Сервіс ***EC2*** був обраний для створення віртуального сервера, що дозволяє гнучко налаштовувати ресурси для розгортання додатка. **Amazon *ECR*** використовується для зберігання Docker-образів, забезпечуючи зручність інтеграції з контейнерними сервісами. ***ElastiCache*** використовується як основна база даних в якій зберігатиметься лічильник візитів додатку. **Amazon *CloudWatch*** забезпечує моніторинг додатка та інфраструктури, дозволяючи відстежувати продуктивність, автоматизувати реагування на проблеми та збирати метрики для аналізу. Ці сервіси працюють у тісній інтеграції, що робить AWS оптимальним вибором для розробки власного модуля. Проводимо реєстрацію нового Root юзера та отримуємо доступ до ресурсів.

3.3.1. Створення EC2 Instance

Для створення EC2 Instance обираємо таку ж назву як і для раніше створеного GitHub репозиторію - ***counter-app***. Також обираємо образ операційної системи **Ubuntu Server** через його широкий функціонал та зручне налаштування.

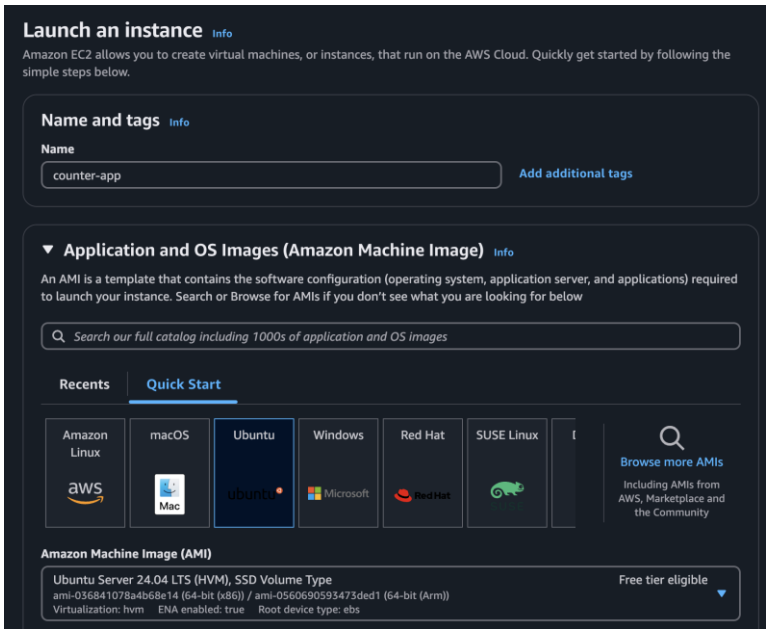

Рисунок 3.5. Створення EC2 Instance - Вибір назви та операційної системи

Наступним кроком є вибір чіпу на якому буде працювати інстанс, так як додаток на даному етапі не вимагає багатьох ресурсів, обираємо найпростішу опцію - ***t2.micro***. Для віддаленого доступу до майбутнього інстансу створюємо новий

ключ-пару під назвою counter_aws. Сгенерований контент файла формату **.pem** дозволятиме підключатись до командного рядка інстансу через ssh, а також взаємодіяти з інстансом через AWS CLI - набір інструментів та утиліт для роботи з платформою для різних мов програмування та середовищ.

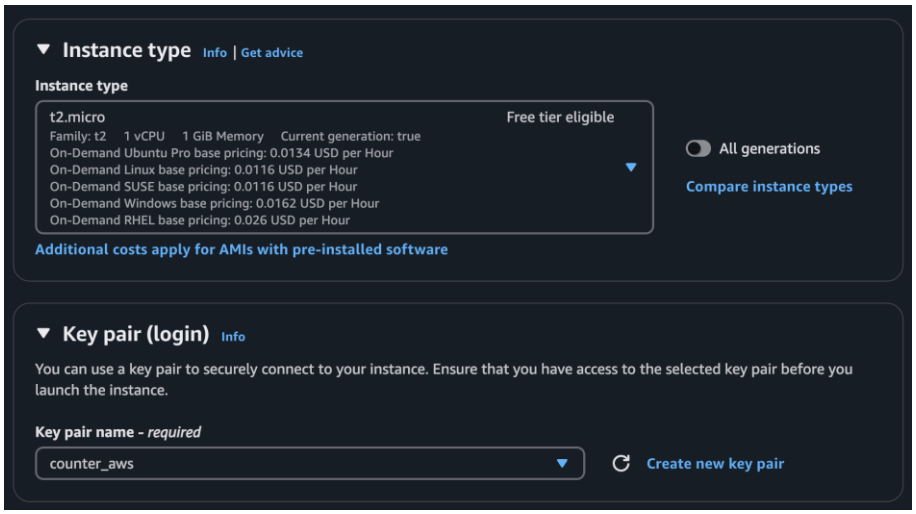

Рисунок 3.6. Створення EC2 Instance - Вибір чіпу та опції доступу

Дозволяємо трафік через HTTP та HTTPS для всіх, щоб була можливість отримати доступ до ресурсів майбутнього застосунку з будь якої IP адреси. Також обираємо опцію для жорсткого диску майбутнього інстансу - 8 Gb пам`яті буде достатньо.

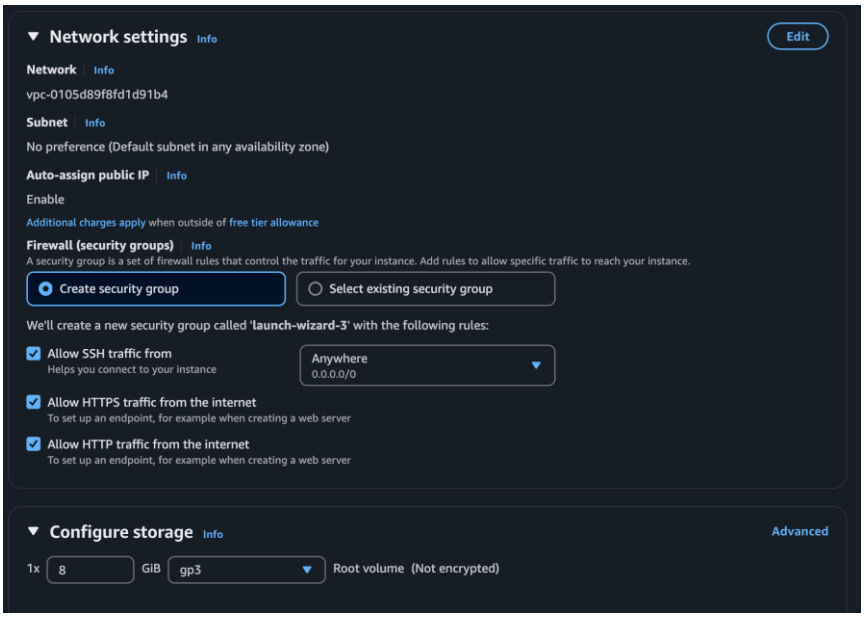

Рисунок 3.7. Створення EC2 Instance - Налаштування локальної мережі та об`єму пам'яті

Зберігаємо налаштування натискаючи кнопку ***Launch Instance***.

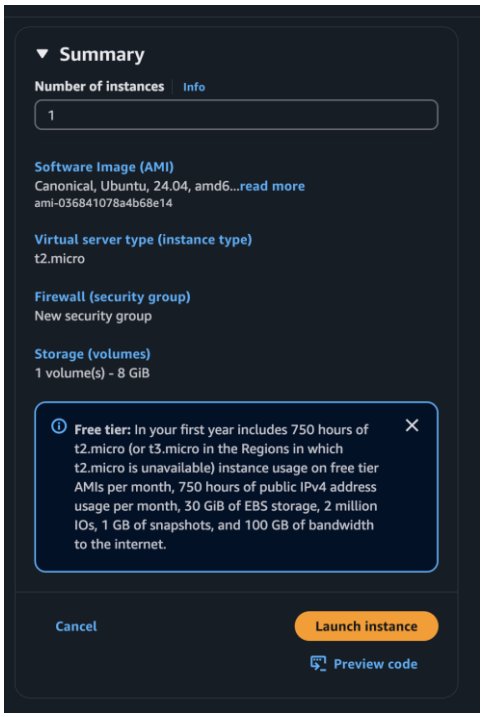

Рисунок 3.8. Підсумки налаштувань EC2 Instance

Через деякий час відвідаємо сторінку AWS дашборда де відображаються всі доступні EC2 Instance, у списку присутній новостворений інстанс налаштування котрого ми проводили.

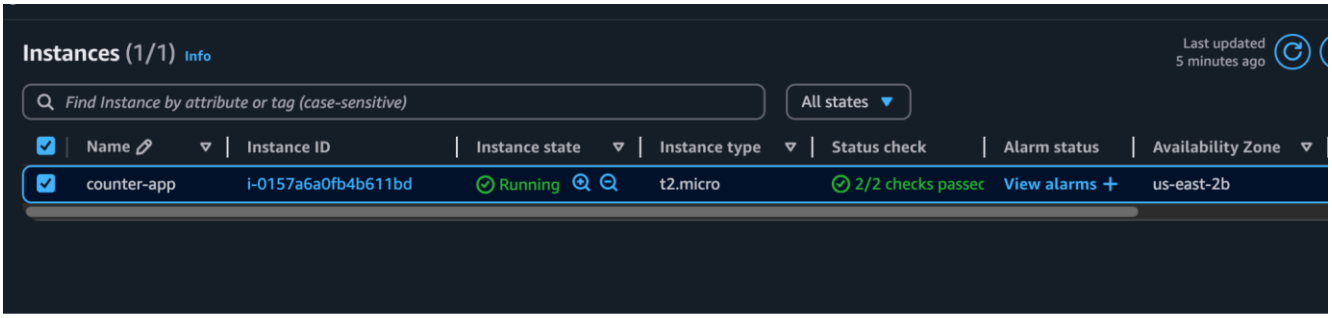

Рисунок 3.9. Список доступних EC2 Instances

В результаті виконаної роботи було створено та налаштовано EC2 Instance для майбутньої інтеграції з інструментами автоматизованого розгортання та іншими хмарними сервісами.

3.3.2. Створення репозиторія ECR

Для створення приватного репозиторію зберігання, управління та розповсюдження контейнерних образів використовуватимемо сервіс Amazon ECR. Далі налаштовуються параметри Image tag mutability, які визначають, чи можна буде змінювати теги контейнерних образів. Опція Mutable дозволяє перезаписувати існуючі образи з однаковими тегами, тоді як Immutable забезпечує стабільність, забороняючи такі дії. Вибір цього параметра важливий для забезпечення контролю версій образів.

Наступним етапом є налаштування Encryption settings. Репозиторій може використовувати шифрування за замовчуванням на основі стандарту AES-256 або ключі, створені через ***AWS Key Management Service***. Варто зазначити, що після створення репозиторію ці налаштування змінити неможливо. Шифрування захищає дані образів під час їхнього зберігання в ECR.

Завершуємо процес натисканням кнопки Create, після чого репозиторій стає доступним для завантаження та управління контейнерними образами.

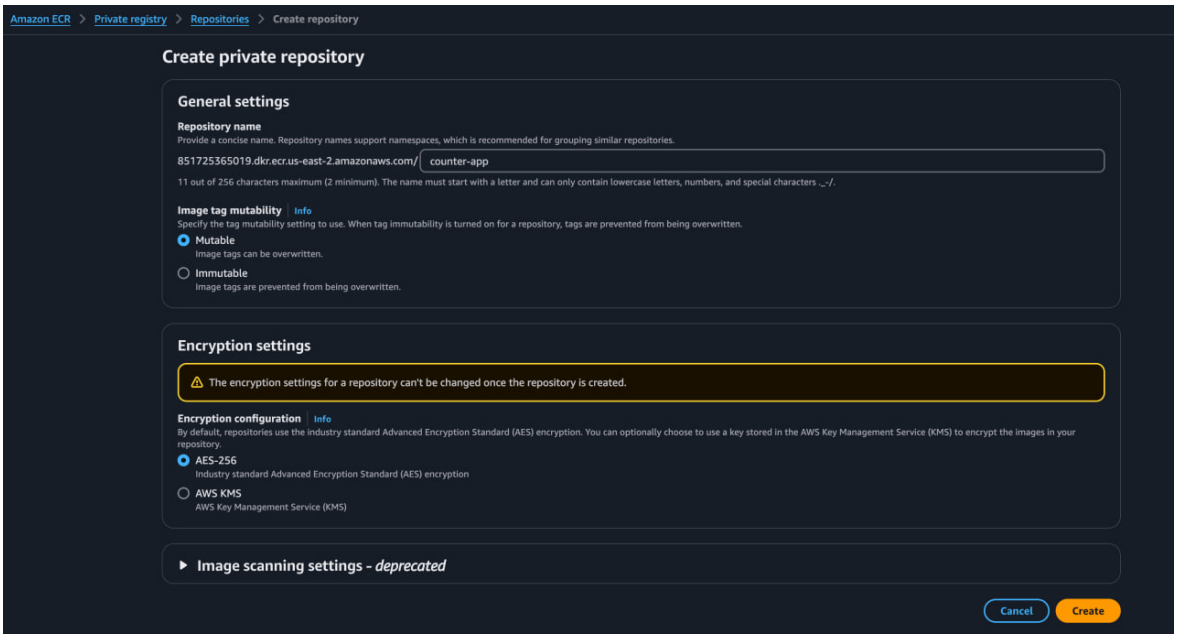

Рисунок 3.10. Створення ECR репозиторію - налаштування

Відвідаємо сторінку дашборда де відображаються всі доступні приватні репозиторії, бачимо новостворений репозиторій який успішно створено та налаштовано.

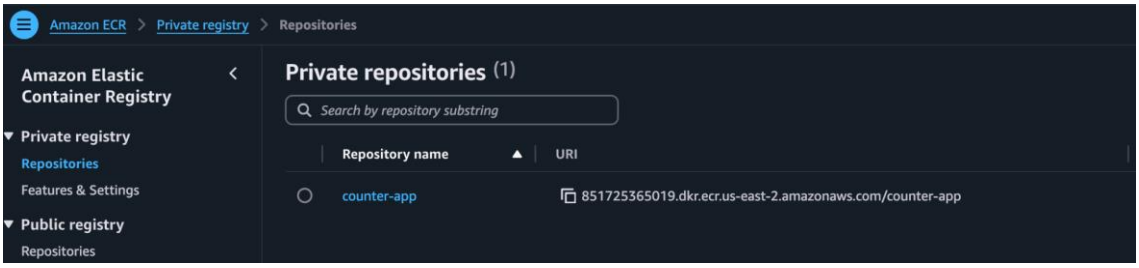

Рисунок 3.11. Список доступних приватних репозиторій ECR

3.3.3. Створення користувача IAM

Створення IAM користувача необхідне для забезпечення програмного доступу до сервісів AWS через AWS CLI який буде використовуватись у автоматизованому розгортанні додатку.

Використовуючи сервіс IAM переходимо до вкладки ***Users***, де обираємо опцію створення нового юзера(***Create User***). Потрапляємо до покрокового створення нового користувача.

У полі User name вказуємо ім'я користувача - counter-app. Це ім'я використовується для ідентифікації користувача в системі. Опція Provide user access to the AWS Management Console не вибрана, що означає, що користувач не матиме доступу до веб-консолі AWS, а лише до API, CLI чи SDK. Цей підхід застосовується, якщо необхідний виключно програмний доступ. Натискаємо ***Next.***

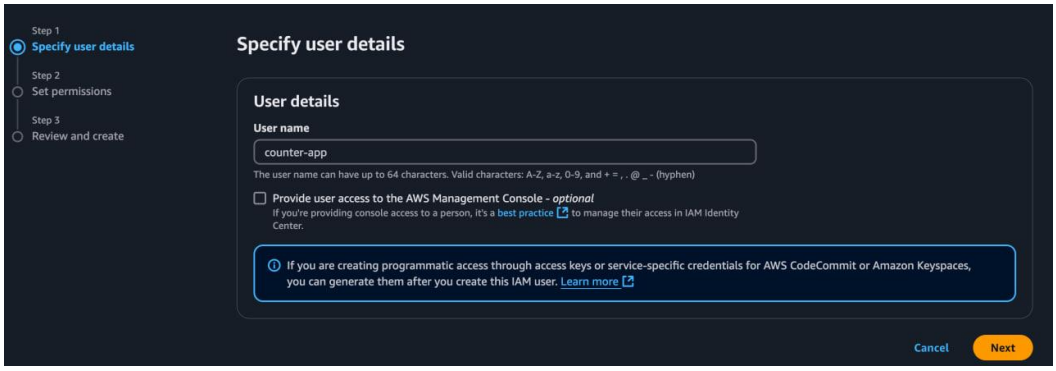

Рисунок 3.12. Створення користувача - Вибір ім`я та доступу до AWS Management Console

На наступному етапі задаються дозволи для нового користувача. Обирається опція Attach policies directly, яка дозволяє додавати політики доступу

безпосередньо до користувача, замість використання груп чи копіювання існуючих дозволів. Із запропонованого списку політик обираємо опції для повного доступу до сервісів EC2 та ECR. Після вибору необхідних політик натискаємо кнопку ***Next***, щоб підсумувати та завершити створення користувача.

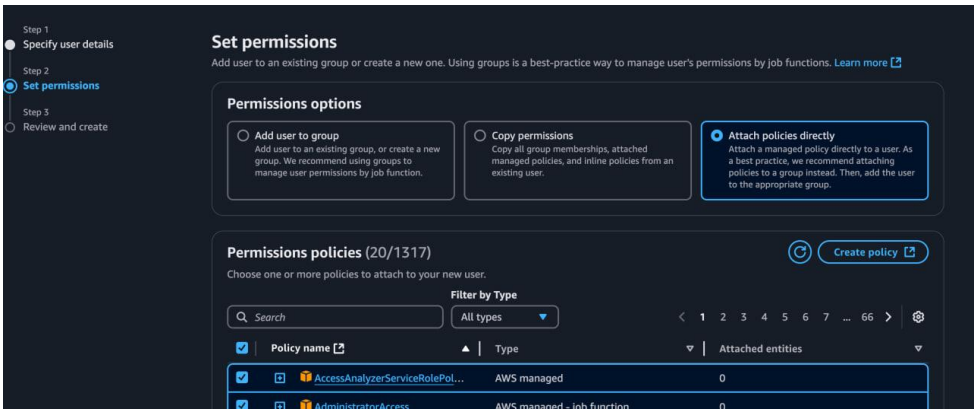

Рисунок 3.13. Створення користувача - надання дозволів до ресурсів

Після створення користувача його дані з'являються в загальному списку користувачів.

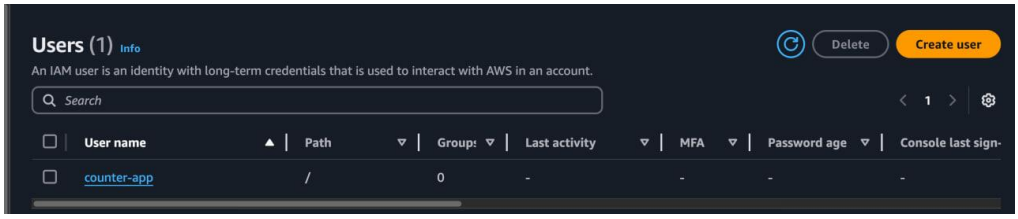

Рисунок 3.14. Список створених користувачів

3.3.4. Створення кластеру ElastiCache

На першому етапі відбувається вибір типу вузлів, їх розмір та кількість реплік. В даному модулі нас не цікавить імплементація реплік, тож обираємо опцію ручного розгортання і вимикаємо cluster mode для створення одного шарду.

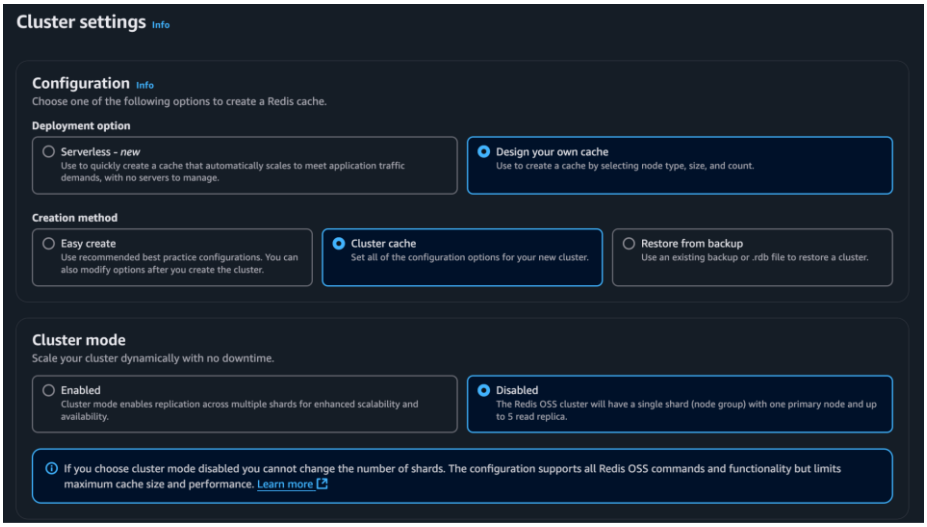

Рисунок 3.15. Створення ElastiCache кластеру - Вибір опцій розгортання та режиму кластеру

Далі вказуємо назву кластеру - ***counter-app-redis***, та обираємо розташування в хмарі AWS.

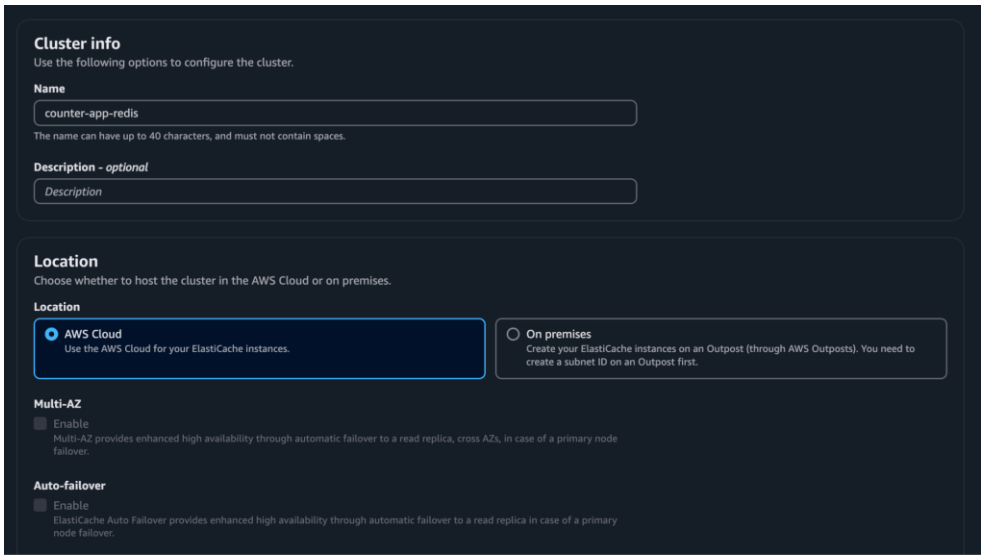

Рисунок 3.16. Створення ElastiCache кластеру - Вказання імені та вибір розташування

Залишаємо опцію Engine version по замовчуванню 7.1 - найсвіжіша версія Redis. Вказуємо порт ***6379*** - стандартний для Redis та тип вузла cache.t2.micro для економії ресурсів і кількість реплік 0 для роботи лише з основним вузлом.

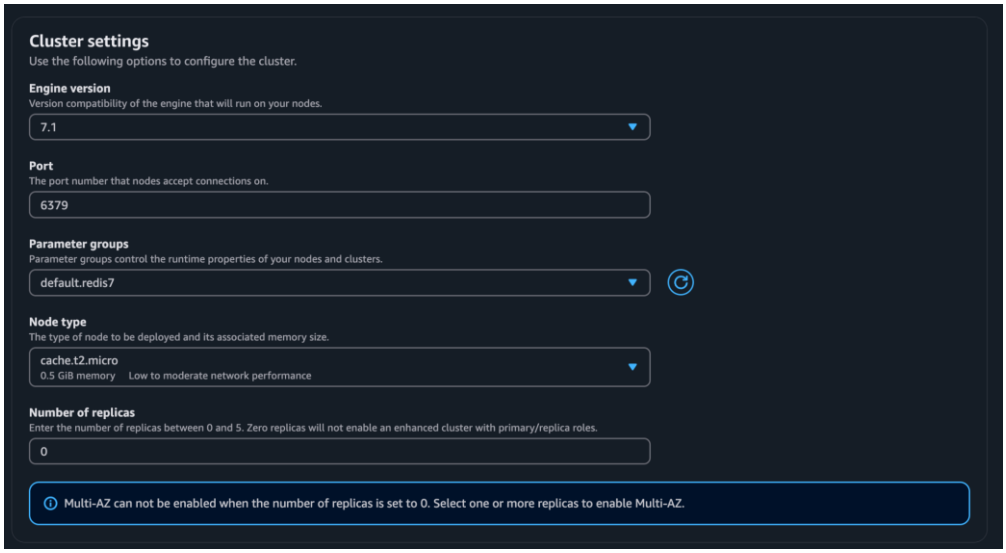

Рисунок 3.17. Створення ElastiCache кластеру - Налаштування параметрів Redis, типу вузла та реплік

Далі обираємо існуючу група підмереж, ***counter-app-subnet***, яка вже використовується для раніше створеного EC2 Instance. Обираємо регіон в рамках якого буде доступний наш кластер - ***us-east-2b***, така сама опція що і для раніше створених EC2 інстансу та ECR репозиторія.

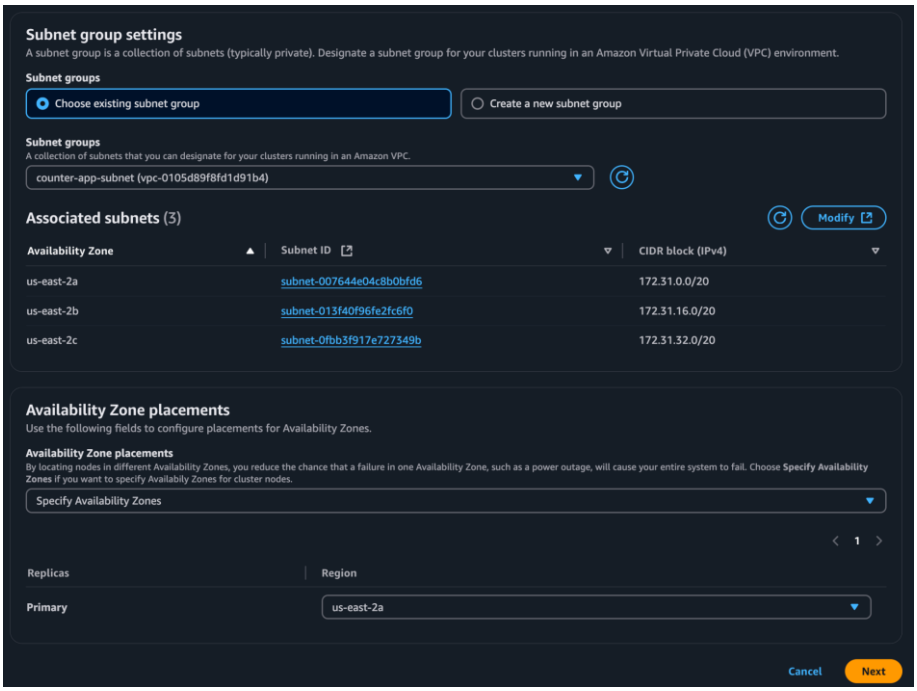

Рисунок 3.18. Створення ElastiCache кластеру - Вибір групи підмереж та зони доступності

Останнім етапом є налаштування Security Group - віртуальний брандмауер у AWS, що дозволяє задавати правила доступу на основі IP-адрес, портів і протоколів, забезпечуючи безпеку та ізоляцію мережевих з'єднань. Створимо нову безпекову групу під назвою ***Redis***, в якій дозволимо доступ до порту ***6379*** для TCP з'єднання з будь якої адреси. Необхідно включити цю групу не тільки до ElastiCache кластеру який ми налаштовуємо, а і до вже створеного EC2 Instance, щоб ми могли користуватись кластером з інстансу. Також вкажемо періодичність бекапів бази даних - раз у день.

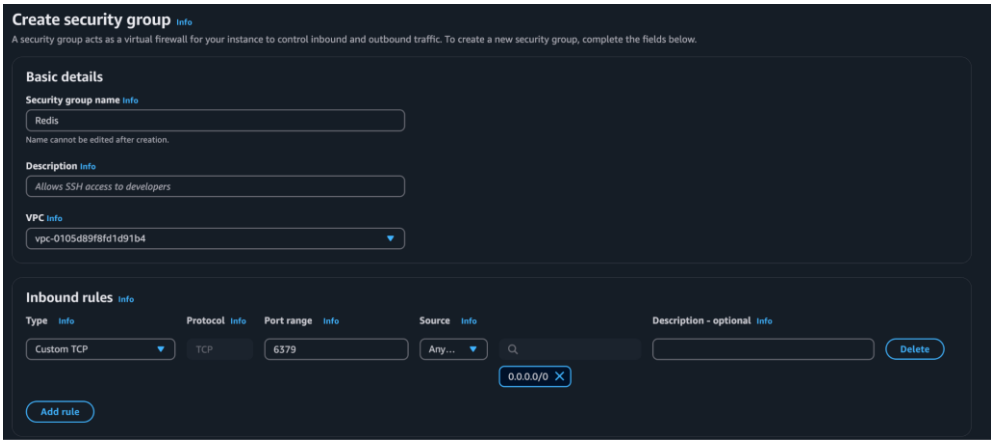

Рисунок 3.19. Створення безпекової групи

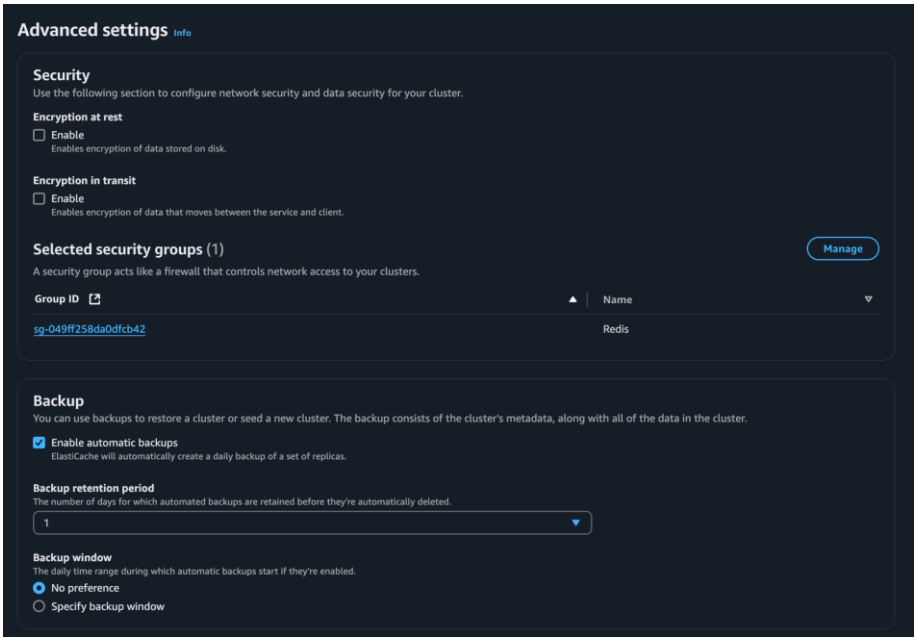

Рисунок 3.20. Створення ElastiCache кластеру - вибір безпекової групи та періодичності бекапів

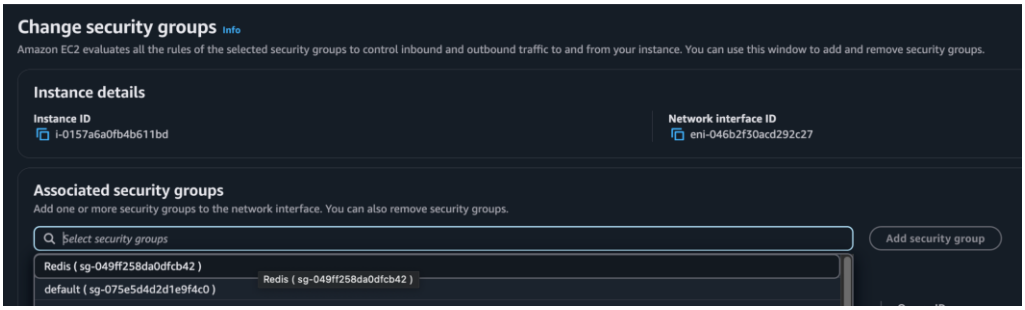

Рисунок 3.21. Додавання новоствореної безпекової групи до EC2 Instance для майбутньої взаємодії з ElastiCache кластером

3.4. Налаштування автоматизованого розгортання WEB-додатку

Для налаштування автоматизованого розгортання WEB-додатку з використанням GitHub Actions Workflow, який вже використовується для автоматизованого тестування(CI) потрібно написати скрипт і вказати секрети доступу і відповідні URL для отримання доступу workflow job до налаштованого середовища AWS.

3.4.1. Додавання секретів у GitHub репозиторій

Для налаштування потрібно визначити наступні секрети: AWS_ACCESS_KEY, AWS_SECRET_ACCESS_KEY, AWS_SSH_KEY, AWS_PUBLIC_KEY, AWS_REDIS_URL.

Для отримання AWS_SSH_KEY потрібно зчитати файл, що було завантажено на етапі створення EC2 Instance, а саме - ***counter-app.pem***. Відкриваємо файл у редакторі коду, та копіюємо контент. Для додавання секрету у репозиторій GitHub, необхідно через WEB-ресурс зайти в налаштування раніше створеного репозиторія, і перейти у вкладку Secrets and Variables / Actions. Далі додаємо нову пару ключ-значення натискаючи кнопку ***New Repository Secret***. Назва ключа - ***AWS_SSH_KEY***, значення - раніше скопійований секретний ключ формату ***.pem***.

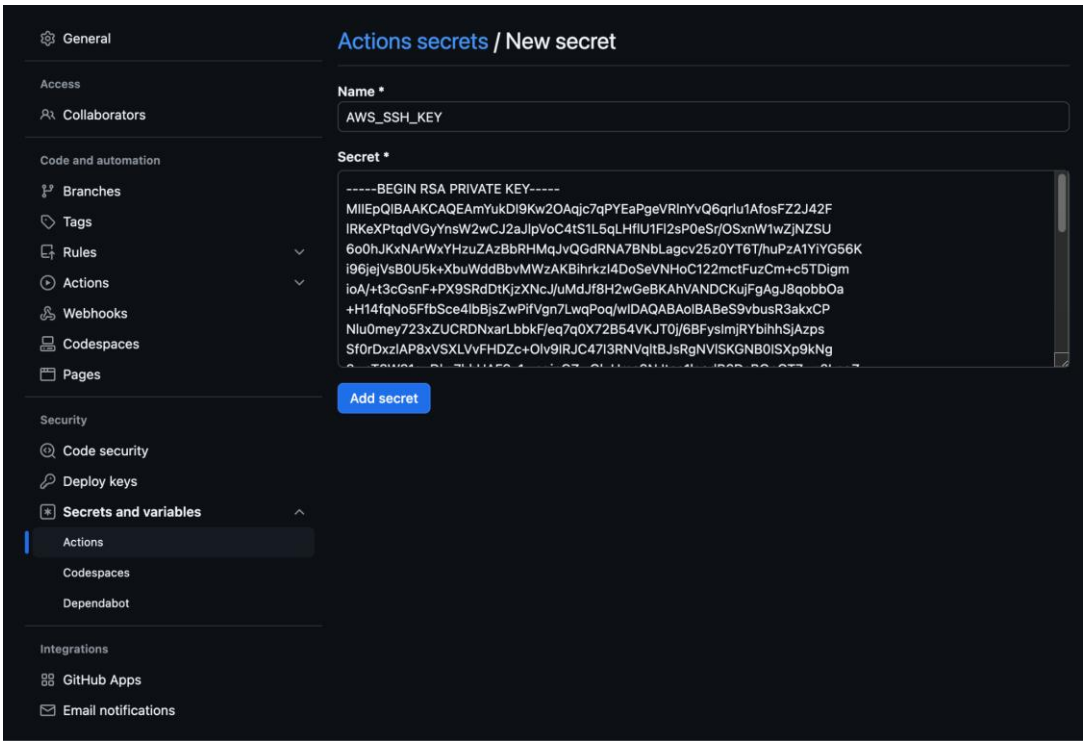

Рисунок 3.22. Додавання секретів до Github репозиторія - інтерфейс

В результаті, було створено перший секрет GitHub репозиторія додатку, продовжимо пошук і додавання наступних.

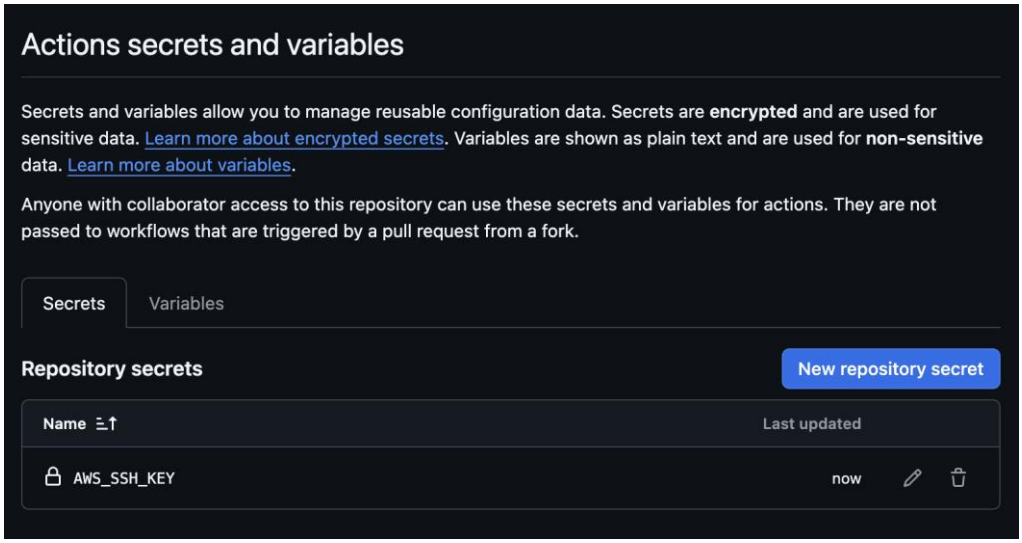

Рисунок 3.23. Додавання секретів до Github репозиторія - перший секрет

AWS_ACCESS_KEY та AWS_SECRET_ACCESS_KEY потрібно згенерувати для раніше створеного через сервіс ***IAM*** користувача. Знаходимо через список створених користувачів юзера з ім’ям counter-app, переходимо до його

налаштувань, та обираємо опцію ***Create access key***. У формі створення вибираємо опцію для use-case - Command Line Interface, що дозволить отримати доступ до командного рядка середовища AWS через AWS CLI який використовуватиметься у скрипті автоматизованого розгортання.

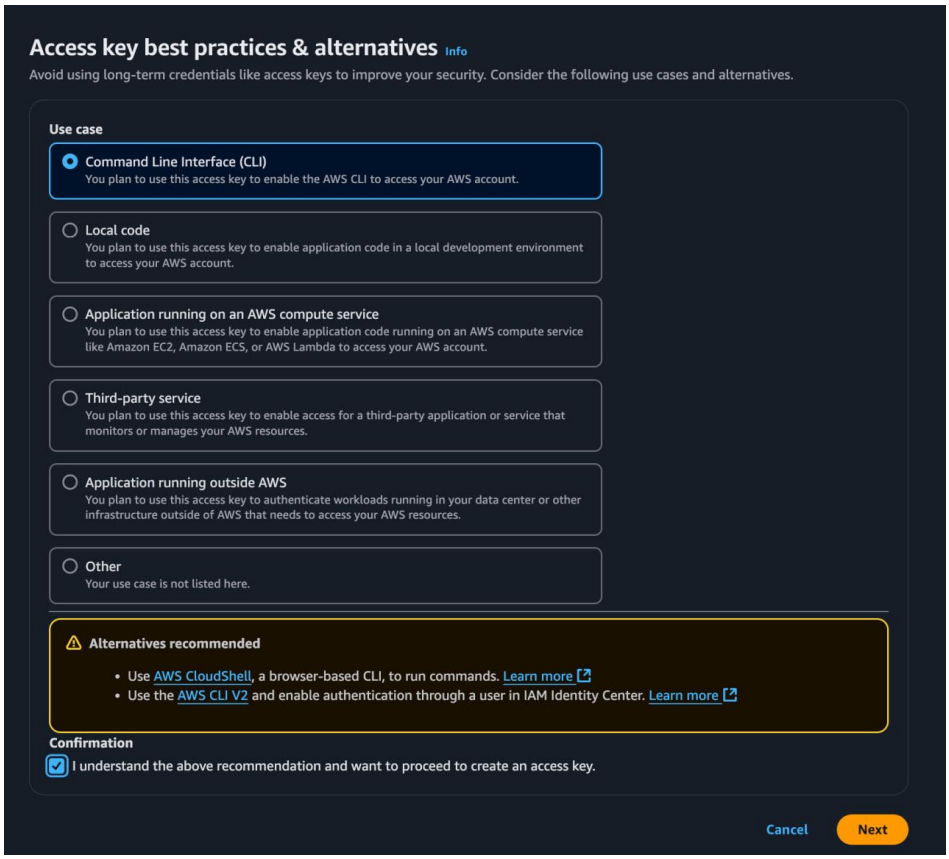

Рисунок 3.23. Створення пари Access Key для IAM користувача - вибір використання

Пропускаємо наступний крок з вибором тегу для майбутніх ключів, та обираємо опцію ***Create Access Key***. Ключі доступу готові до копіювання, додамо їх до GitHub секретів репозиторію.

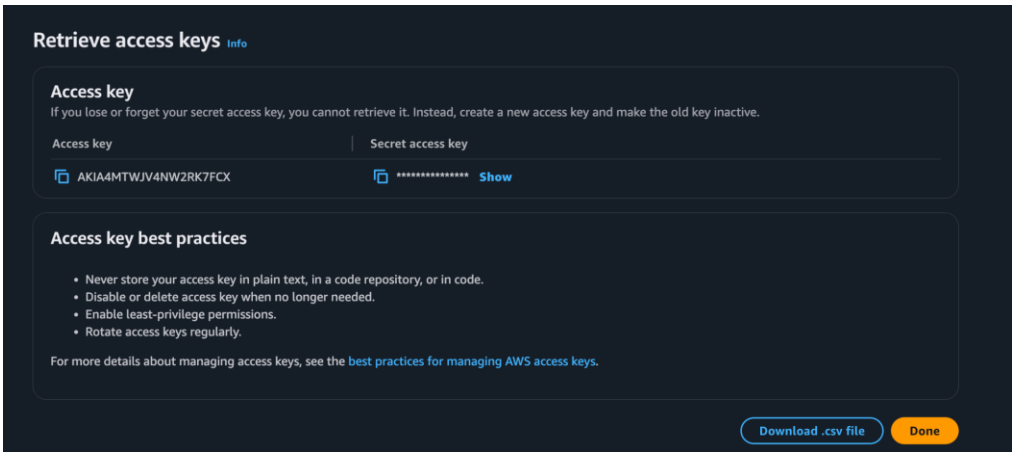

Рисунок 3.23. Створення пари Access Key для IAM користувача - результат

Наступними ключами є AWS_PUBLIC_KEY та AWS_REDIS_URL, для їх отримання потрібно перейти в деталі EC2 Instance та ElastiCache кластеру відповідно. В інформації про інстанс знаходимо ***Public IPv4 address*** та копіюємо значення. AWS_REDIS_URL є посилання на ендпоінт ElastiCache кластеру, з доданням стандартизованого префіксу **redis://** для посилання на базу даних Redis.

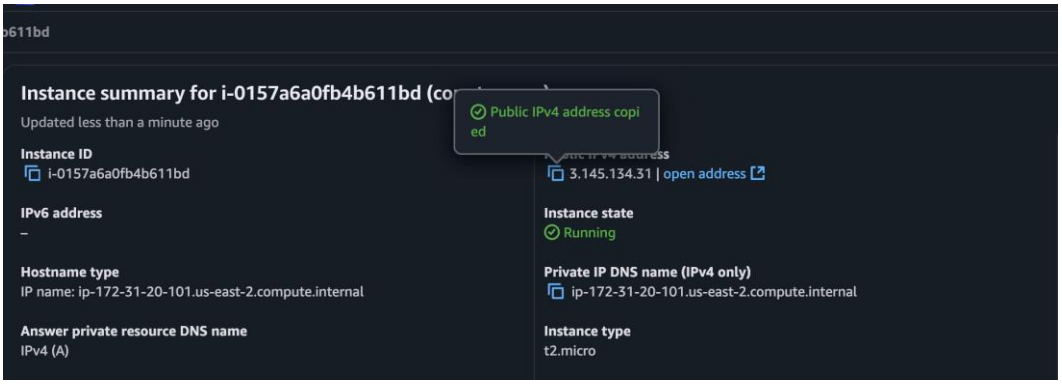

Рисунок 3.24. EC2 Instance - копіювання AWS_PUBLIC_KEY

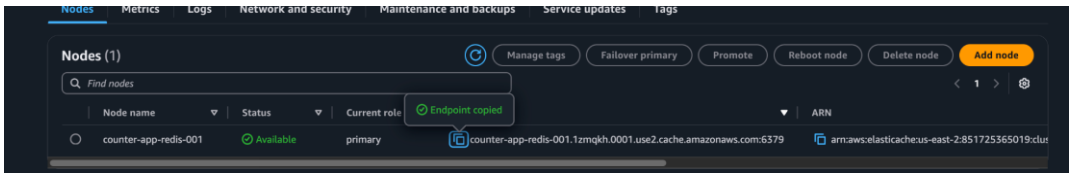

Рисунок 3.25. ElastiCache кластер - копіювання AWS_REDIS_URL

В результаті отримаємо 5 секретів, які використовуватимуться у наступних кроках автоматизації розгортання додатку на хмарну платформу AWS.

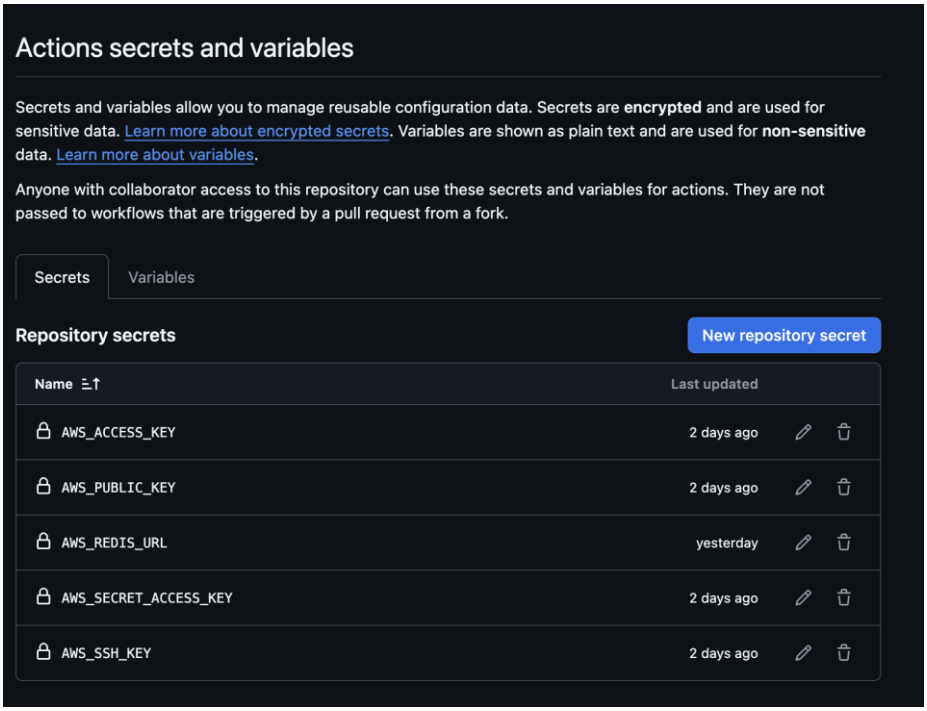

Рисунок 3.26. Додавання секретів до Github репозиторія - результат

3.4.2. Налаштування GitHub Actions Workflow

Розробимо GitHub Action workflow для автоматичного розгортання додатку у хмарному середовищі AWS у разі внесення змін у кодову базу/налаштування проекту, для цього створюємо файл у директорії ***.github/workflows*** *з назвою* ***aws-deploy.yml***.

```
name: AWS Deploy
on:
 push:
   branches:
     - "main"
env:
 AWS_REGION: us-east-2
 AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY }}
 AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
 PRIVATE_SSH_KEY: ${{ secrets.AWS_SSH_KEY }}
 SERVER_PUBLIC_IP: ${{ secrets.AWS_PUBLIC_KEY }}
 AWS_REDIS_URL: ${{ secrets.AWS_REDIS_URL }}
jobs:
 deploy:
   runs-on: ubuntu-latest
   steps:
     - name: Checkout
       uses: actions/checkout@v3
     - name: Login to AWS ECR
       id: login-ecr
       uses: aws-actions/amazon-ecr-login@v1
     - name: Build, push docker image
       env:
         REGISTRY: ${{ steps.login-ecr.outputs.registry }}
```

```
          REPOSITORY: counter-app
          IMAGE_TAG: ${{ github.sha }}
        run: |-
          docker build -t $REGISTRY/$REPOSITORY:$IMAGE_TAG .
          docker push $REGISTRY/$REPOSITORY:$IMAGE TAG
      - name: Deploy docker image to EC2
        env:
          REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          REPOSITORY: counter-app
          IMAGE_TAG: ${{ github.sha }}
          AWS_DEFAULT_REGION: us-east-2
        uses: appleboy/ssh-action@master
        with:
          host: ${{ env.SERVER_PUBLIC_IP }}
          username: ubuntu
          key: ${{ env.PRIVATE_SSH_KEY }}
          envs:
PRIVATE_SSH_KEY,REGISTRY,REPOSITORY,IMAGE_TAG,AWS_ACCESS_KEY_ID,AWS_SECRET_ACCESS
_KEY,AWS_DEFAULT_REGION,AWS_REGION,AWS_REDIS_URL
          script: |-
            sudo apt update
            sudo apt install unzip
            sudo apt install docker.io -y
            curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
            unzip awscliv2.zip
            sudo ./aws/install
            aws ecr get-login-password --region us-east-2 | sudo docker login --
username AWS --password-stdin 851725365019.dkr.ecr.us-east-2.amazonaws.com
            sudo docker stop counter-app || true
            sudo docker rm counter-app || true
            sudo docker pull $REGISTRY/$REPOSITORY:$IMAGE_TAG
            sudo docker run -d --name counter-app -p 4567:4567 -e
REDIS_URL=$AWS_REDIS_URL $REGISTRY/$REPOSITORY:$IMAGE_TAG
```

Лістинг файлу .github/workflows/aws-deploy.yml

Даний workflow автоматизує процес розгортання застосунку на інфраструктурі AWS. Він виконується щоразу, коли в гілку ***main*** репозиторію здійснюється push. У процесі workflow налаштовується середовище, будуються контейнерні образи, завантажуються в AWS Elastic Container Registry (ECR) і розгортаються на віртуальному сервері Amazon EC2.

Workflow починається з визначення тригеру виконання — push до гілки ***main***. У розділі ***env*** задаються глобальні змінні середовища, необхідні для роботи. Секрети, такі як ключі доступу AWS, приватний SSH-ключ, IP-адреса сервера та URL Redis, передаються через GitHub Secrets для безпечного зберігання й використання.

Основна робота відбувається в задачі ***deploy***, яка виконується на базі ***ubuntu-latest***. Спочатку виконується крок Checkout, що завантажує останню версію коду з репозиторію. Потім через екшн ***aws-actions/amazon-ecr-login@v1*** виконується аутентифікація до AWS Elastic Container Registry (ECR).

На етапі створення контейнерного образу будується Docker-образ за допомогою команди docker build і завантажується в репозиторій ECR через docker push. Образ позначається тегом, який відповідає SHA хешу останнього коміту, що забезпечує унікальність кожного завантаженого образу.

Наступний етап — розгортання застосунку на сервері EC2. Через екшн ***appleboy/ssh-action*** встановлюється SSH-з'єднання з EC2-інстансом, використовуючи IP-адресу сервера, ім’я користувача (ubuntu) і приватний SSH-ключ. На сервері виконується низка команд для підготовки середовища: оновлення пакетів, встановлення Docker та AWS CLI. Далі виконується автентифікація в ECR, і Docker-образ завантажується на сервер.

Перед запуском нового контейнера завершується робота попереднього екземпляра додатка через ***docker stop*** та ***docker rm***. Потім запускається новий контейнер із завантаженим образом. Контейнер працює на порту ***4567***, а URL до Redis передається як змінна середовища REDIS_URL. Це дозволяє застосунку інтегруватися з Redis для кешування даних.

Таким чином, цей workflow забезпечує повний цикл CI/CD: від побудови і завантаження образів до їхнього автоматизованого розгортання на інфраструктурі AWS. Це спрощує управління додатком і забезпечує швидкий розгортання змін.

Модуль розроблений і готовий до тестування.

3.5. Тестування роботи розробленого модуля

Для початку, відправимо всі локальні зміни до репозиторію, це повинно слугувати тригером до виконання написаних GitHub Actions Workflow - автоматизованого тестування та розгортання.

В результаті надсилання змін з коментарем ***Test build workflow*** на віддалений репозиторій було виконано дві задачі, а саме ***Application Build*** та ***AWS Deploy.***

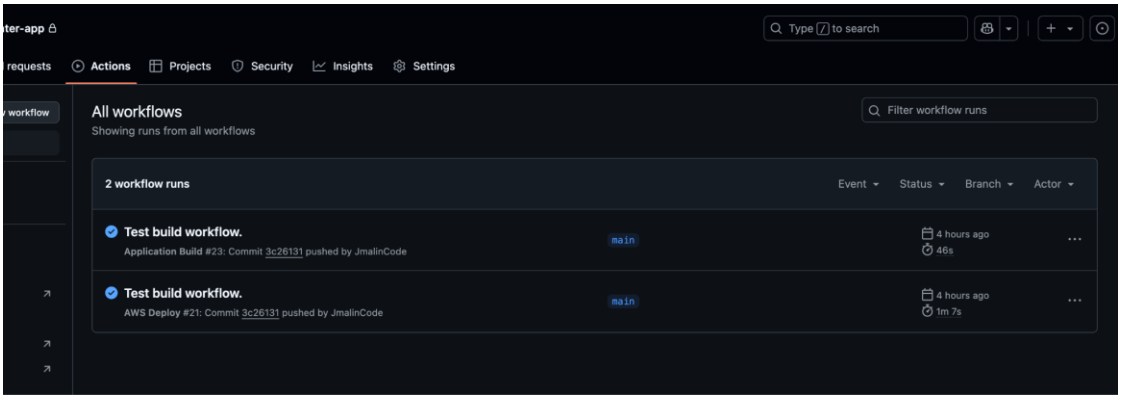

Рисунок 3.27. Виконані GitHub Actions робочі процеси

Результатом виконання робочого процесу ***Application Build*** є успішне автоматизоване виконання тестів. Що дозволяє перейти до розгортання додатку у хмарному середовищі

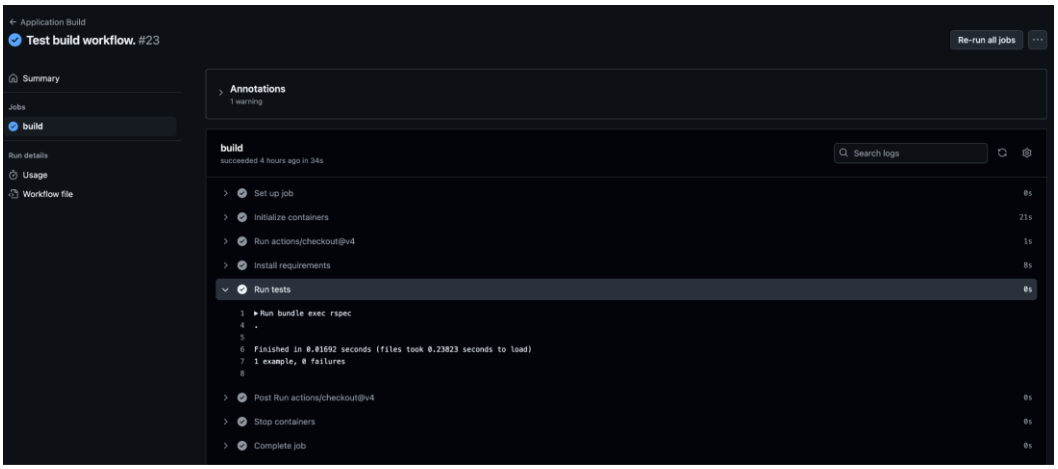

Рисунок 3.28. Результат виконання автоматизованого тестування

Результатом виконання робочу процесу ***AWS Deploy*** стало додавання образу контейнеру застосунка у репозиторій ECR, з подальшим використання цього образу у створеному EC2 Instance.

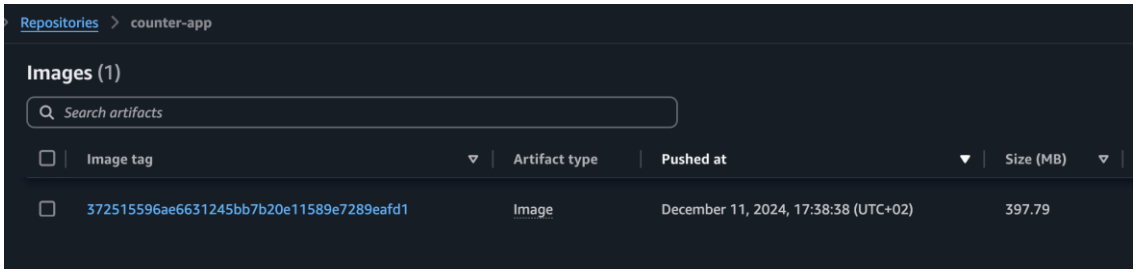

Рисунок 3.29. Список образів приватного репозиторію ECR

Перевіримо роботу додатку через публічну адресу EC2 Instance. Оновімо сторінку кілька разів для перевірки функціоналу додатку. Додаток працює відмінно

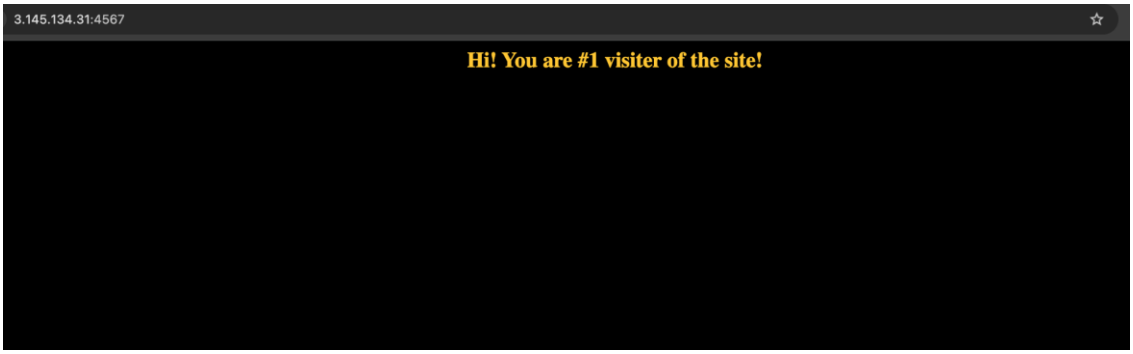

Рисунок 3.30. Головна сторінка додатку-лічильника

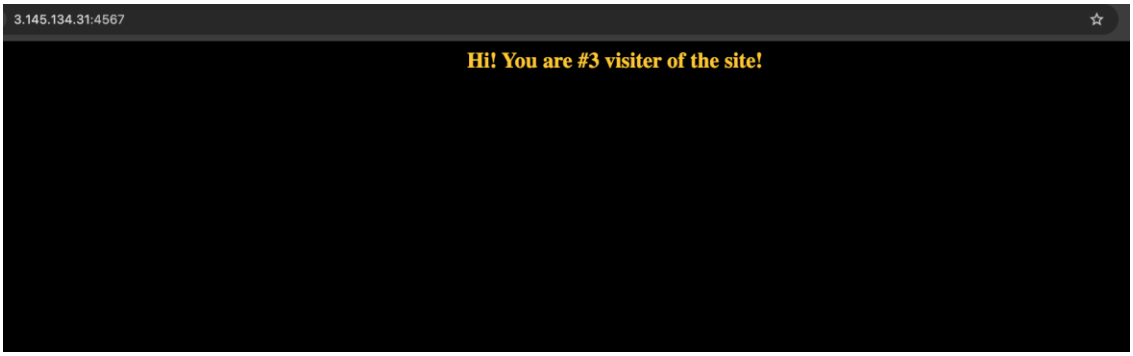

Рисунок 3.31. Головна сторінка додатку-лічильника після двох оновлень сторінки

Проведемо тестування розробленого модуля шляхом відтворення процесу розробки програмного забезпечення, а саме додавання запиту на додавання змін у коді, з подальшим злиттям змін у головну гілку main, що у свою чергу повинно виконати автоматизоване розгортання змін застосунку.

Для тесту підійде зміна тексту головної сторінки додатку. Адаптуємо його під українську мову, для цього змінимо контент файлу представлення

головної сторінки додатку, створимо фіч-гілку зі змінами, через яку створимо ***Pull Request*** для злиття змін в основну гілку, та надішлемо зміни до віддаленого репозиторію.

```
<body style="background:black;">
 <h2 style="color:#ffc733; text-align:center;" >Привіт! Ти <%= @counter %>-й
відвідувач сайту!</h2>
</body>
```

Лістинг файлу views/index.erb

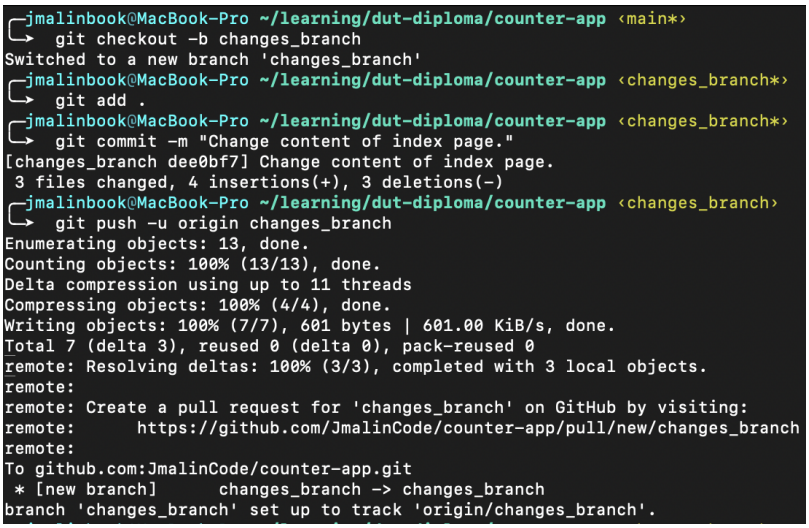

Рисунок 3.32. Впровадження змін кодової бази через створення нової гілки

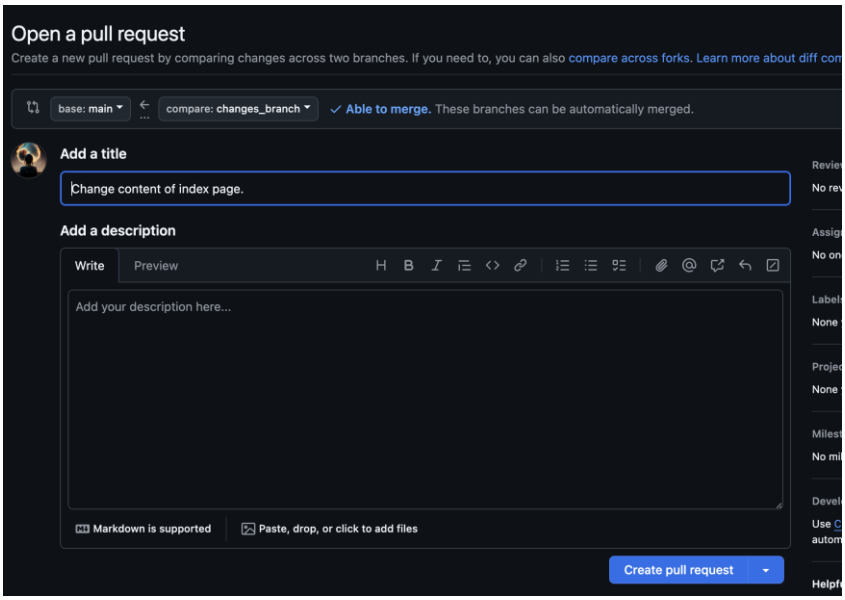

Рисунок 3.33. Створення Pull Request зі змінами кодової бази

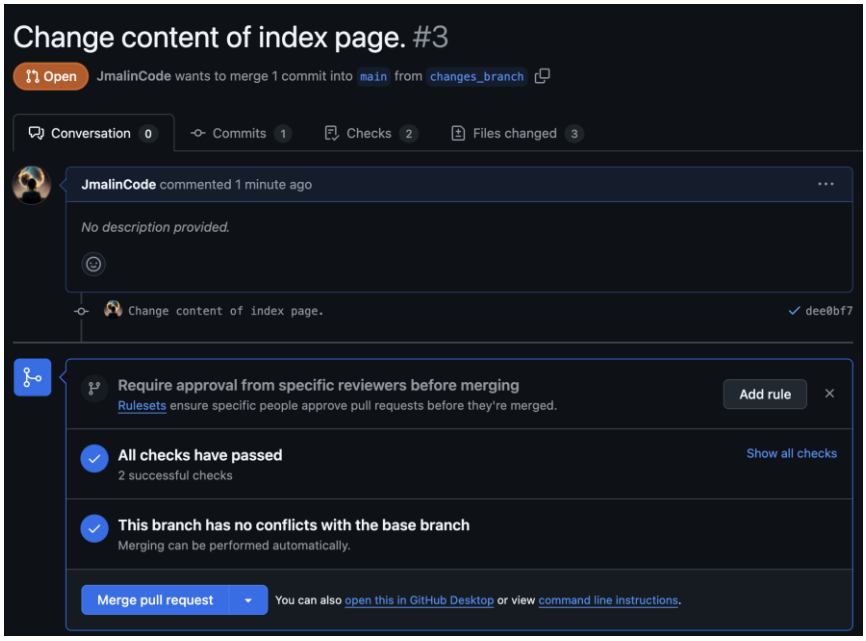

Рисунок 3.34. Успішне виконання автоматизованого тестування внесених змін

В результаті злиття змін в основну гілку проекту, було виконано робочий процес розгортання додатку у хмарному середовищі. Оновлюємо сторінку додатку у браузері, контент змінено без втрати даних про минулих відвідувачів.

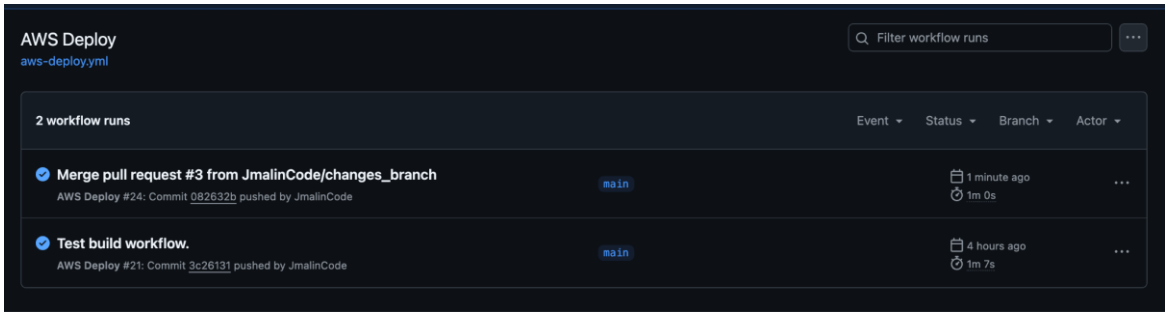

Рисунок 3.35. Успішне виконання автоматизованого розгортання внесених змін

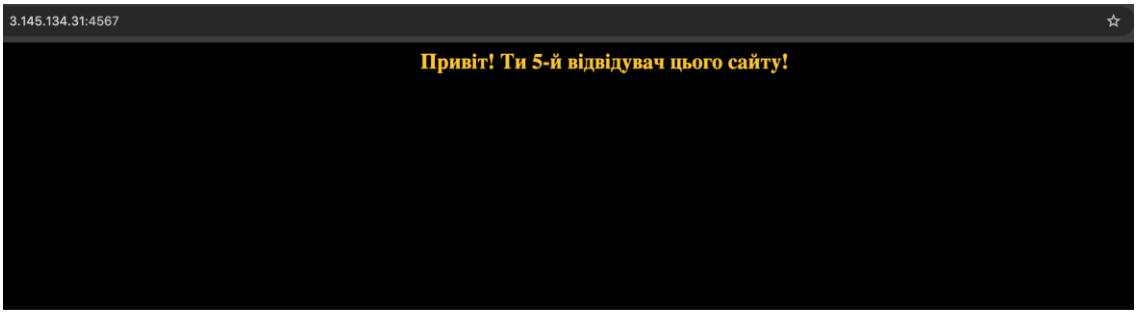

Рисунок 3.36. Успішне застосування внесених змін у хмарному середовищі

ВИСНОВКИ

У цій роботі було досліджено, спроектовано та реалізовано модуль автоматизованого розгортання та моніторингу додатків у хмарному середовищі. Використовуючи сучасні інструменти та технології, такі як GitHub Actions, Docker, та сервіси AWS, вдалося створити ефективну архітектуру, яка забезпечує безперервну інтеграцію та доставку програмного забезпечення. Це дозволило автоматизувати процеси тестування, збірки та розгортання, мінімізуючи ризик людських помилок і скорочуючи час доставки оновлень.

У рамках реалізації було розроблено веб-застосунок із використанням фреймворку Sinatra, його структура була контейнеризована за допомогою Docker, а система розгортання інтегрована з хмарними сервісами AWS, такими як EC2, ECR і ElastiCache. Для кожного етапу, включно з тестуванням коду, збиранням контейнерних образів, їхнім завантаженням у репозиторій і подальшим розгортанням на сервері, було налаштовано окремі робочі процеси GitHub Actions. Це дозволило забезпечити автоматизацію всіх ключових кроків роботи з додатком.

Окрему увагу було приділено моніторингу та забезпеченню стабільності системи. Використання CloudWatch дало змогу відстежувати стан розгортання, збирати метрики продуктивності й оперативно реагувати на можливі помилки. Це підвищило надійність системи, що є критично важливим для її роботи у виробничому середовищі.

Таким чином, результати цієї роботи демонструють ефективність інтеграції сучасних інструментів автоматизації та хмарних сервісів у процес розробки і розгортання програмного забезпечення. Розроблений підхід може бути використаний як основа для впровадження подібних рішень у інших проектах, спрямованих на оптимізацію процесів DevOps.

ПЕРЕЛІК ПОСИЛАНЬ

1 Martin Fowler - Continuous Integration (Мартін Фаулер - Безперервна інтеграція) [Електронний ресурс] - https://martinfowler.com/articles/continuousIntegration.html [1]

2 What is DevOps? Meaning, methodology and guide (Що таке DevOps? Значення, методологія та посібник) [Електронний ресурс] - https://www.techtarget.com/searchitoperations/definition/DevOps [2]

3 Understanding the Fundamentals of Continuous Integration and Continuous Delivery (Розуміння основ безперервної інтеграції та безперервної доставки) [Електронний ресурс] - https://attractgroup.com/blog/understanding-the-fundamentals-of-continuous-integration-and-continuous-delivery/ [3]

4 Deploying with GitHub Actions (Розгортання за допомогою GitHub Actions) [Електронний ресурс] - https://docs.github.com/en/actions/use-cases-and-examples/deploying/deploying-with-github-actions [4]

5 Документація Sinatra - https://github.com/sinatra/sinatra [5]

6 Документація Docker [Електронний ресурс] - https://docs.docker.com/ [6]

7 Документація Amazon Web Services [Електронний ресурс] - https://docs.aws.amazon.com/ [7]

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістра

НА ТЕМУ:«МОДУЛЬ АВТОМАТИЗОВАНОГО РОЗГОРТАННЯ І МОНІТОРИНГУ ЗАСТОСУНКІВ НА ХМАРНИХ ПЛАТФОРМАХ»

Виконав студент групи КСДМ-61
Кучеренко Дмитро Андрійович

Керівник Дипломної роботи:
доцент кафедри, к.т.н., с.н.с. Торошанко Ярослав Іванович

Мета роботи - дослідження і використання актуальних інструментів для автоматизованого розгортання та моніторингу застосунків на хмарних платформах.

Об`єкт дослідження - хмарні платформи та засоби для розгортання і моніторингу застосунків.

Предмет дослідження - процеси автоматизації розгортання та моніторингу застосунків у хмарному середовищі.

Актуальність теми

Автоматизація процесів розгортання та моніторингу застосунків у хмарних середовищах є важливим аспектом сучасної IT-індустрії та стала ключовим елементом для забезпечення конкурентоспроможності бізнесу. Оскільки вона знижує кількість людських помилок, забезпечує швидке внесення змін та дозволяє масштабувати системи в залежності від динамічних потреб.

В контексті сучасної розробки програмного забезпечення, з автоматизацією розгортання в тому числі мається на увазі використання хмарних платформ, таких як AWS, Google Cloud та Azure, що надають спектр інструментів для налаштування, масштабування та моніторингу середовища розгортання.

Використані технології

Назва	Призначення
Ruby	Серверна мова програмування
Sinatra	Фреймворк для розробки web застосунку
Redis	База даних
Docker Compose	Інструмент для контейнеризації та розгортання web застосунку
GitHub Actions	Платформа автоматизації
AWS EC2	Середовище розгортання
AWS ElastiCache	Сховище даних
AWS ECR	Реєстр контейнерів
AWS CloudWatch	Інструмент для моніторингу

Результати роботи

У рамках реалізації було розроблено веб-застосунок із використанням Ruby-фреймворку Sinatra, його структура була контейнеризована за допомогою Docker, а система розгортання інтегрована з хмарними сервісами AWS, такими як EC2, ECR і ElastiCache. Для кожного етапу, включно з тестуванням коду, збиранням контейнерних образів, їхнім завантаженням у репозиторій і подальшим розгортанням на сервері, було налаштовано окремі робочі процеси GitHub Actions. Це дозволило забезпечити автоматизацію всіх ключових кроків роботи з додатком.

Результати роботи

Схема Модуля автоматизованого розгортання і моніторингу застосунків на хмарних платформах

Висновок

В ході роботи було досліджено, спроектовано та реалізовано модуль автоматизованого розгортання та моніторингу додатків у хмарному середовищі. Використовуючи сучасні інструменти та технології, такі як GitHub Actions, Docker та сервіси AWS, вдалося створити ефективну архітектуру, яка забезпечує безперервну інтеграцію та доставку програмного забезпечення. Це дозволило автоматизувати процеси тестування, збірки та розгортання, мінімізуючи ризик людських помилок і скорочуючи час доставки оновлень кодової бази застосунків.