Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» _____ 2023 р

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «**МОДЕЛЬ ПРОЦЕСУ ІНТЕГРАЦІЇ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ В ПЛАТФОРМУ AMAZON WEB SERVICES**»

Захищено на засіданні
Атестаційної комісії № 42
протокол № __ від __.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ **СКОБ Ю. О.**
(підпис) (прізвище та ініціали)

Виконав:
студент групи КУ– 61
Галузь знань: 15 – Автоматизація та приладобудування
спеціальність: 151 – «Автоматизація та комп'ютерно-інтегровані технології»
МІРОШНИЧЕНКО
Дмитро Олександрович

Керівник: д.т.н.,с.н.с, професор кафедри теоретичної
та прикладної системотехніки
ТОЛСТОЛУЗЬКА
Олена Геннадіївна

Рецензент: к.ф.-м.н., доцент, в.о. зав. кафедри електроніки і управляючих систем
ХРУСЛОВ Максим Михайлович

Харків – 2023

АНОТАЦІЯ

Пояснювальна записка до магістерської кваліфікаційної роботи роботи складається зі вступу, трьох розділів, висновків, переліку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 87 сторінок, із яких 59 сторінок основної частини з 20 рисунками, 1 таблицею, 27 найменувань списку використаних джерел та чотирма додатками на 17 сторінках.

В сучасному світі інформаційних технологій є актуальною проблема запровадження автоматизації для розгортання інфраструктури програмних виробів у хмарних середовищах. Тому метою роботи було підвищення ефективності, гнучкості та масштабованості процесу створення хмарних інфраструктур за допомогою підходів інфраструктури як коду та контейнерної оркестрації

При виконанні кваліфікаційної роботи були використані моделі хмарних обчислень, UML діаграми, підхід інфраструктури як коду, контейнерна оркестрація, засоби програмного забезпечення Terraform, Docker-compose, мови програмування Python.

У ході виконання роботи було отримано: огляд моделей хмарних обчислень, методи взаємодії з хмарним середовищем та модель процесу автоматизованого створення інфраструктури у хмарі.

Головною областю використання моделі є технологічні процеси, які мають на меті автоматизацію налаштування програмного забезпечення, сервісів та компонентів інфраструктур.

Ключові слова: AWS, Terraform, Docker-compose, Python, CMS, WordPress, MySQL, модель, інфраструктура, конфігурація.

ABSTRACT

The explanatory note to the master's qualification work consists of an introduction, three sections, conclusions, a list of used sources, and a summary. The total volume of the work is 87 pages, of which 59 pages are the main part with 20 pictures, 1 table, 27 names of the list of used sources and three appendices on 17 pages.

In the modern world of information technologies, the problem of introducing automation for deploying the infrastructure of software products in cloud environments is urgent. Therefore, the aim of the work was to develop a model of the infrastructure integration process for the content management system to increase the level of efficiency, flexibility and scalability of the process of creating cloud infrastructures using infrastructure-as-code and container orchestration approaches.

Cloud computing models, UML diagrams, the infrastructure-as-code approach, container orchestration, Terraform software tools, Docker-compose, and Python programming languages were used in the qualification work.

In the course of the work, the following was obtained: an overview of cloud computing models, methods of interaction with the cloud environment, and a model of the process of automated creation of infrastructure in the cloud.

The main area of use of the model is technological processes aimed at automating the configuration of software, services and infrastructure components.

Keywords: AWS, Terraform, Docker-compose, Python, CMS, WordPress, MySQL, model, infrastructure, configuration.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

AWS – Amazon Web Services;

БД – база даних;

CMS – Content Management System (система управління контентом);

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

PaaS – Platform as a Service (платформа як послуга);

IaaS – Infrastructure as a Service (інфраструктура як послуга);

CLI – Command Line Interface (інтерфейс командного рядка);

IaC – Infrastructure as Code (інфраструктура як код);

ВСТУП

Хмарні технології відкривають перед підприємствами нові горизонти масштабованості, гнучкості та ефективності. Вони дозволяють бізнесам оптимізувати свої витрати, поліпшити доступність та надійність своїх сервісів, та швидко адаптуватися до змінних ринкових умов. Однак, як і будь-яка інноваційна технологія, хмарні обчислення пропонують не У сучасному світі, де цифрова трансформація набуває все більшого значення, важливість хмарних технологій у бізнес-екосистемі стрімко зростає. Серед них особливу роль відіграють платформи, такі як Amazon Web Services (AWS), які надають потужні можливості для управління, зберігання та обробки великих обсягів даних. Ця кваліфікаційна робота фокусується на моделюванні процесу інтеграції систем управління контентом в AWS, досліджуючи можливості та виклики, що виникають у контексті розгортання інфраструктури у хмарному середовищі.

тільки переваги, але й виклики, зокрема у сферах безпеки даних, відповідності нормативним вимогам, та управління залежностями від провайдерів послуг.

Актуальність дослідження полягає у зростаючій ролі технологій хмарних обчислень у сучасному світі інформаційних технологій. Хмарні рішення відіграють важливу роль у тому, як підприємства зберігають і обробляють свої дані, пропонуючи альтернативу традиційним on-premises (внутрішнім) обчисленням. Це особливо актуально з огляду на постійний розвиток обох підходів, що свідчить про динамічний характер сучасного IT-світу. Таким чином питання створення моделей та підходів для взаємодії з хмарними середовищами та автоматизація створення інфраструктур для програмних продуктів стає актуальним на сьогоднішній день.

Метою дослідження є підвищення ефективності, гнучкості та масштабованості процесу створення хмарних інфраструктур за допомогою підходів інфраструктури як коду та контейнерної оркестрації.

Об'єкт дослідження – процес автоматичного розгортання інфраструктури у хмарному середовищі.

Методи дослідження: моделі хмарних обчислень, UML діаграми, підхід інфраструктури як коду, контейнерна оркестрація, засоби програмного забезпечення Terraform, Docker-compose, мови програмування Python.

Предмет дослідження – методи та моделі організації автоматизованого створення інфраструктур на основі ресурсів постачальника хмарних послуг в залежності від специфікацій архітектури інфраструктури.

Завдання дослідження

1. Виконати аналіз сучасного стану хмарних технологій та моделей хмарних обчислень.
2. Проаналізувати підходи та методології взаємодії з хмарними середовищами.
3. Розробити нову (чи модифікувати існуючу) модель для процесу розгортання інфраструктури у хмарі для програмного продукту на прикладі системи управління контентом для веб-сайту.
4. Реалізувати частину розробленої моделі за допомогою підходу інфраструктури як коду та контейнерної оркестрації.

Практичне значення одержаних результатів роботи: розроблена модель процесу інтеграції системи управління контенту для репрезентації можливості впровадження автоматизації для розгортання інфраструктури у хмарному середовищі. Дана модель спрямована на розширення переліку концепцій взаємодії з хмарними платформами, що дає змогу впроваджувати більш ефективні підходи для планування розробки архітектур програмного забезпечення.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ РОЗВИТКУ ТЕХНОЛОГІЇ ХМАРНИХ ОБЧИСЛЕНЬ

Розвиток інформаційних технологій в останні десятиліття запроваджує розмаїття способів, за допомогою яких підприємства зберігають та обробляють свої дані. Два основних підходи, які відзначаються серед бізнес-середовищ, це технологія cloud (хмарні рішення) та технологія on-premises (внутрішні обчислення). Постійний розвиток обох цих підходів свідчить про неспокійну динаміку сучасного IT-світу.

1.1. Моделі хмарних обчислень

Сучасний стан розвитку технології хмарних рішень вражає своєю еволюцією. Хмарні послуги надають можливість доступу до обчислювальних ресурсів, зберігання даних та програмного забезпечення через Інтернет. Це дозволяє підприємствам знизити витрати на обладнання та обслуговування, а також швидко масштабувати свої операції. Технологія cloud змінила підхід до IT-інфраструктури, забезпечуючи більшу гнучкість та мобільність в управлінні ресурсами.

Згідно з дослідженням Foundry’s 2023 Cloud Computing research[1] на сьогоднішній день популярність хмарних обчислень становить 52% до 48%, які використовують локальні середовища (on-premises) (див. рис. 1.1) [1].

Відстежуючи тенденцію збільшення популярності технології хмарних обчислень до 2025 року кількість інфраструктур, які використовують хмарні обчислення перевищить 60%[1], тобто дві третіх усіх інфраструктур буде знаходитись у хмарі, або ж частково використовувати хмарні технології.

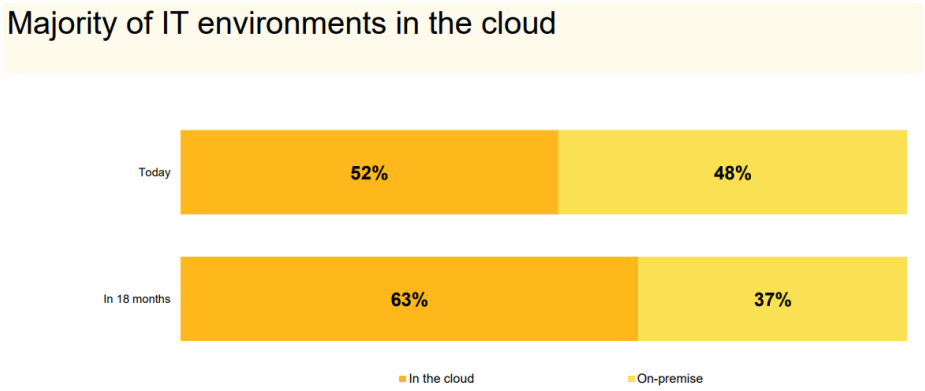

Рис. 1.1. – Відношення середовищ у хмарі до локальних.

У такому контексті, аналіз сучасного стану розвитку технології cloud стає надзвичайно актуальним, оскільки він сприяє зрозумінню тенденцій та вибору найкращого підходу для конкретних бізнес-завдань і вимог.

Таким чином доцільно провести аналіз переваг і недоліків моделей, які базуються на хмарних рішеннях [2]

Переваги хмарних інфраструктур:

- Надзвичайна масштабованість: хмарна інфраструктура дозволяє безперервно змінювати обсяг ресурсів залежно від потреб, що забезпечує найкращу продуктивність та ефективність.
- Значна економія коштів: замість витрат на капіталовкладення в обладнання та його обслуговування, хмарні рішення дозволяють оплачувати лише використані ресурси, що спрощує бюджетування, оскільки у своїй більшості постачальники хмарних рішень користуються моделлю оплати за використання.
- Розширена доступність і гнучкість: можливість доступу до хмарних послуг з будь-якого географічного місця з підключенням до Інтернету забезпечує продуктивну співпрацю та роботу на відстані.
- Автоматичні оновлення: постачальники хмарних послуг відповідають за оновлення інфраструктури та безпеки, таким чином знімаючи з користувача

зобов’язання що-до оновлення програмного забезпечення, яке надають постачальники.

- Просте аварійне відновлення: хмарні рішення включають до списку послуг резервне копіювання та аварійне відновлення, тим самим забезпечуючи резервування копій та високу доступність, що знижує час простою та гарантує неперервну роботу бізнесу.

Проте, існують потенційні недоліки, які слід враховувати:

- Питання безпеки і конфіденційності даних: надання конфіденційних даних постачальнику хмарних послуг може викликати занепокоєння стосовно безпеки та конфіденційності.
- Питання відповідності: організації повинні гарантувати, що вони дотримуються правил та галузевих стандартів при використанні хмарних послуг.
- Прив'язка до постачальника: перехід від одного постачальника хмарних послуг до іншого може бути складним і дорогим, оскільки організації можуть стати залежними від певних функцій або послуг, які пропонує певний постачальник.

Архітектура хмарних обчислень репрезентується у вигляді постачання сервісів, які представлені трьома моделями хмарних обчислень[3]:

SaaS (Software as a Service - програмне забезпечення як послуга):

SaaS - це модель, в якій програмне забезпечення розглядається як послуга, доступна через Інтернет. Користувачі отримують доступ до програмних додатків через веб-браузер, і вони можуть використовувати це програмне забезпечення без необхідності встановлювати та підтримувати його на власних серверах. Прикладами SaaS-рішень є Google Workspace (раніше G Suite), Microsoft 365 (раніше Office 365), Salesforce, Dropbox тощо.

Переваги SaaS включають простоту використання, автоматичне оновлення та масштабованість. Користувачам не потрібно витрачати час на налаштування та обслуговування інфраструктури або програмного забезпечення.

PaaS (Platform as a Service - платформа як послуга):

PaaS - це модель, в якій надаються середовища розробки та інструменти для створення, розгортання та управління програмними додатками. PaaS надає розробникам доступ до платформи, на якій вони можуть створювати власні додатки, не хвилюючись про інфраструктуру.

Прикладами PaaS-послуг є Microsoft Azure App Service, Google App Engine, Amazon Web Services Elastic Beanstalk тощо. Розробники використовують PaaS для розробки, тестування та розгортання своїх додатків, маючи можливість легко масштабувати та керувати ними.

IaaS (Infrastructure as a Service - інфраструктура як послуга):

IaaS - це модель, в якій надаються віртуальні обчислювальні ресурси, такі як віртуальні машини, сховища даних та мережеві ресурси через Інтернет. Користувачі можуть орендувати ці ресурси та використовувати їх для будь-яких обчислювальних завдань.

Прикладами IaaS-послуг є Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, і DigitalOcean. Інфраструктура як послуга надає велику гнучкість і контроль над обчислювальною інфраструктурою, дозволяючи користувачам розгортати власні операційні системи та додатки на віртуальних машинах.

У відповідності до темпів розвитку хмарних технологій, а також запровадження нових можливостей, які надаються постачальниками хмарних сервісів виникає потреба у розробці нових концепцій та методів взаємодії з хмарним середовищем, оскільки взаємодія з хмарними технологіями супроводжується наступними комплексними аспектами[4]:

1. Безпека: забезпечення безпеки даних і дотримання відповідних стандартів є однією з найбільших турбот при розгортанні інфраструктури у хмарі. Важливо забезпечити захист від несанкціонованого доступу, атак, витоків

даних і забезпечити конфіденційність та цілісність інформації. Прикладами з налаштувань безпеки можуть бути:

• Доступ та ідентифікація: необхідно налагодити механізми для впевненості в тому, що тільки авторизовані користувачі мають доступ до ресурсів.

• Захист даних: забезпечення конфіденційності та цілісності даних шляхом шифрування та контролю доступу.

• Захист від атак: захист від різних видів атак, таких як DDoS, SQL-ін'єкції, кросс-сайтові атаки тощо.

2. Відмовостійкість: інфраструктура у хмарі має бути відмовостійкою, тобто здатною працювати навіть в разі виникнення проблем або збоїв. Це може потребувати дублювання ресурсів у різних географічних регіонах і резервного зберігання даних. Зазвичай під час проектування інфраструктури у хмарі відносно відмовостійкості враховують наступне:

• Резервне копіювання: ротреба в регулярному резервному копіюванні даних та налаштування резервних серверів для швидкого відновлення.

• Географічна розподіленість: розгортання ресурсів в різних географічних регіонах для зменшення впливу можливих проблем.

3. Масштабованість: під час побудови інфраструктури важливо враховувати можливість масштабування, оскільки хмарні середовища надають можливість легко змінювати кількість ресурсів у залежності від потреб. До масштамобованості відносять наступні пункти:

• Горизонтальне масштабування: здатність збільшувати обсяг ресурсів, додавання нових серверів або контейнерів для підвищення продуктивності.

• Автоматизація: використання відповідних інструментів для автоматичного масштабування відповідно до попиту.

4. Управління: управління інфраструктурою у хмарі може бути складним завданням, оскільки ресурси можуть розгортатися і знестачуватися динамічно.

Важливо мати ефективні інструменти моніторингу для виявлення проблем та відстеження продуктивності, інструменти керування і автоматизації операцій, що у змозі виконувати ефективне розподілення ресурсів, контроль витрат і оптимізація.

5. Вартість: витрати на інфраструктуру у хмарі можуть швидко зростати, якщо не контролювати їх. Необхідно ретельно аналізувати та оптимізувати витрати на ресурси у хмарному середовищі.

6. Міграція: перенесення існуючих додатків і даних у хмару може бути складним завданням, яке вимагає аналізу і планування, тобто вибору ресурсів для міграції, перевірки сумісності додатків та даних.

7. Відповідність стандартам: деякі галузі і регіони можуть вимагати відповідності певним стандартам і правилам. Важливо бути усвідомленим стосовно вимог і забезпечити відповідність.

8. Доступність: інфраструктура у хмарі зазвичай залежить від доступності Інтернету і сервісів хмарних постачальників. В разі відмови в цьому доступі можуть виникнути проблеми з доступом до ресурсів. Для запровадження належного рівня доступності може проводитись встановлення резервних мереж і засобів доступу для забезпечення доступності в разі відмови основних мереж і сервісів.

1.2. Аналіз методів і технологій взаємодії з хмарним середовищем

Зі збільшенням популярності хмарних технологій і попиту на них, з’явилася велика кількість постачальників хмарних обчислень, а також методів взаємодії з хмарним середовищем і способів розгортання інфраструктур, які базуються як на монолітній архітектурі програмного забезпечення, так і на мікросервісній архітектурі. Станом на 2023 рік найбільш популярними постачальниками хмарних технологій[5] є:

- Amazon Web Services (AWS) - це популярна хмарна платформа, яку надає американська компанія Amazon. AWS пропонує широкий спектр обчислювальних, зберігання, мережевих, баз даних та інших послуг у форматі хмарних обчислень. Ця платформа дозволяє підприємствам, розробникам та іншим користувачам здійснювати різноманітні обчислювальні операції без необхідності інвестувати у власну апаратну інфраструктуру.

AWS також надає можливості для безпеки та захисту даних, включаючи інструменти для шифрування, ідентифікації та контролю доступу.

AWS має регіональні центри обробки даних по всьому світу, що дозволяє користувачам вибирати регіон для розташування своїх даних з урахуванням вимог до відмовостійкості та швидкості доступу.

- Google Cloud Platform (GCP) - це хмарна платформа, розроблена і надана компанією Google для надання різних послуг обчислення, зберігання даних, аналітики, машинного навчання та інфраструктури для розробників, підприємств та дослідницьких організацій. GCP пропонує різні послуги та інструменти, які допомагають користувачам розробляти та впроваджувати хмарні додатки та обчислювальні рішення.

GCP також надає послуги безпеки, моніторингу та керування інфраструктурою, а також дозволяє користувачам розгортати додатки та послуги в різних географічних регіонах і масштабувати їх згідно з потребами.

- Microsoft Azure, також відомий просто як Azure, це хмарна платформа та набір хмарних послуг, які надаються компанією Microsoft.

Azure надає користувачам доступ до обчислювальних ресурсів, зберігання даних, баз даних, інтернету речей, аналітики, штучного інтелекту, послуги DevOps та багатьох інших хмарних послуг та інструментів для розробки та розгортання додатків. Microsoft Azure також надає можливості для моніторингу, безпеки та управління інфраструктурою.

- IBM Cloud - це хмарна платформа, розроблена і надана компанією IBM для надання різних хмарних послуг та інфраструктури для розробників, підприємств та організацій. IBM Cloud пропонує набір інструментів та послуг, що охоплюють обчислювання, зберігання даних, аналітику, штучний інтелект, блокчейн, Інтернет речей (IoT) та інші технології.
- Oracle Cloud - це хмарна платформа і набір хмарних послуг, розроблений компанією Oracle, однією з найбільших та найвідоміших компаній у сфері реляційних баз даних та корпоративного програмного забезпечення. Oracle Cloud надає різноманітні послуги та інфраструктуру для розробки, розгортання та управління додатками та даними в хмарі.

Oracle Cloud також підтримує інші технології, такі як блокчейн, Інтернет речей, машинне навчання та багато інших. Вона популярна серед великих корпорацій і підприємств, які вже використовують реляційні бази даних Oracle та інші продукти Oracle.

- Alibaba Cloud, також відомий як Aliyun, - це хмарна платформа, розроблена і надана компанією Alibaba Group, однією з найбільших та найуспішніших компаній електронної комерції та технологій в Китаї. Alibaba Cloud надає широкий спектр хмарних послуг та інфраструктури для розробки та розгортання додатків та послуг в хмарі.
- Alibaba Cloud також надає можливості для моніторингу, безпеки та управління інфраструктурою. Платформа використовується в Китаї та по всьому світу, і її велика активність в Китаї робить її важливим гравцем на глобальному ринку хмарних послуг.

Відсоткове співвідношення по використанню послугами постачальників хмарних технологій зазначеними вище надається в таблиці 1[5].

Таблиця 1 – відсоткове співвідношення постачальників хмарних обчислень

Постачальник	Доля на ринку постачальників
Amazon Web Services	34%
Microsoft Azure	22%
Google Cloud Platform	9.5%
Alibaba Cloud	6%
Oracle Cloud	2%

Кожен з цих постачальників хмарних обчислень надає можливості для застосування своїх послуг за моделями SaaS, PaaS та IaaS.

В залежності від обраної архітектури побудови інфраструктури (монолітної або ж мікросервісної) існує велика кількість методів для взаємодії з хмарним середовищем, які можна застосовувати під час розгортання інфраструктури у хмарі:

Веб-консоль (портал): більшість хмарних постачальників пропонують веб-заснований інтерфейс управління або портал, де ви можете вручну створювати та налаштовувати ресурси. Цей інтерфейс використовується для створення віртуальних машин, баз даних, сховищ та інших служб. Даний метод зазвичай забирає багато часу, адже прирівнюється до ручного керування ресурсами у хмарі.

Інтерфейс командного рядка (CLI): хмарні постачальники надають інструменти командного рядка (наприклад, AWS CLI, Azure CLI, gcloud), які дозволяють взаємодіяти з хмарними послугами через командний рядок. Цей метод можна автоматизувати та використовувати для автоматизації завдань та розгортання.

Інфраструктура як код (IaC)[6]: IaC - це популярний метод розгортання інфраструктури в хмарі. Він передбачає написання коду для визначення та надання інфраструктурних ресурсів. Поширені інструменти IaC включають Terraform, AWS CloudFormation, Azure Resource Manager (ARM) шаблони та

Google Cloud Deployment Manager. IaC забезпечує повторюваність, контроль версій та можливість управління інфраструктурою як кодом.

Оркестрація контейнерів[7]: якщо є потреба у розгортанні додатків в контейнерах (наприклад, Docker), то для цього існують платформи оркестрації контейнерів, такі як Kubernetes, Docker Swarm або хмарні служби, такі як Amazon ECS та Azure Kubernetes Service (AKS), для управління контейнеризованими додатками в хмарі.

Безсерверні обчислення (Serverless Computing)[8]: це підхід до розробки та розгортання додатків, де розробники можуть виконувати свій код безпосередньо в хмарному середовищі, іншими словами, без явної необхідності управляти віртуальними серверами або інфраструктурою. У цьому випадку функцію керування інфраструктури бере на себе постачальник хмарних обчислень

Платформа як послуга (PaaS): хмарні постачальники пропонують служби за моделлю PaaS, такі як AWS Elastic Beanstalk, Azure App Service та Google App Engine. Ці платформи надають підготовлене середовище для розгортання додатків, абстрагуючись від значної частини управління основною інфраструктурою.

Постійна інтеграція/постійне розгортання (CI/CD) [9]: інструкції для виконання CI/CD автоматизують процес розгортання, інтегруючи його у розробницький процес програмування. Інструменти, такі як Jenkins, Travis CI та CircleCI, можуть бути використані для створення, тестування та автоматичного розгортання ваших додатків в хмарі.

Гібридні та багатохмарні розгортання: деякі організації використовують комбінацію внутрішнього та кількох хмарних постачальників. Інструменти, такі як HashiCorp Terraform та Kubernetes, можуть допомогти управляти гібридними та багатохмарними розгортаннями.

API та SDK: постачальники хмарних послуг надають API та набори розробника (SDK), які дозволяють програмістам взаємодіяти з послугами та

розгортати ресурси програмно. Це корисно для створення власних інструментів або інтеграції розгортання в хмарі в існуючі програми.

Таким чином під час обрання методу розгортання потрібно врахувати такі чинники, як архітектура, потреби у масштабуванні, бюджет та рівень контролю та автоматизації, які необхідні для інфраструктури. У багатьох випадках може використовуватися комбінація цих методів для вирішення різних завдань розгортання.

1.3. Постановка задачі дослідження

Для простоти дослідження методів для процесу розгортання інфраструктури у хмарі візьмемо за приклад створення середовища для роботи веб-сайту. Базова схема того, як працює веб-сайт зображено на рисунку 1.2[10].

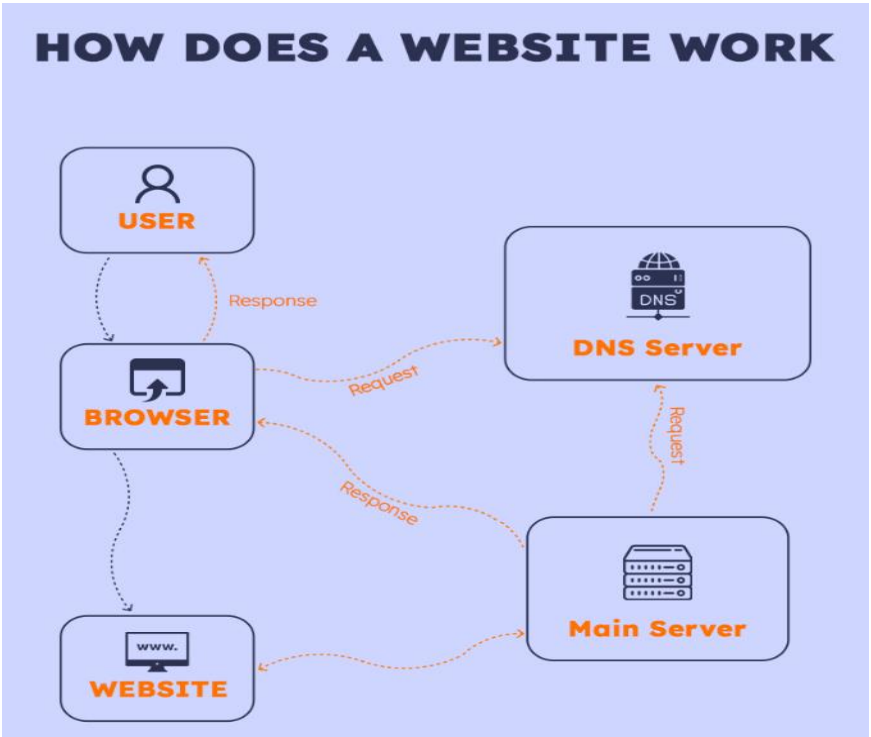

Рис. 1.2. – Схема роботи веб-сайту.

З точки зору розгортання інфраструктури для веб-сайту у хмарному середовищі потрібно врахувати наступні компоненти[11]:

- Веб-сервіс (web service) - тип програмного забезпечення або сервісу, який дозволяє взаємодіяти з функціоналом через Інтернет або інші комп'ютерні мережі. Ці сервіси широко використовуються для передачі даних між різними програмами, які працюють на різних пристроях або в різних місцях. Головна мета веб-сервісів — надання можливості виконувати певні функції або отримувати дані з інших систем за допомогою мережевих протоколів, наприклад, HTTP. Вони служать для інтеграції різноманітних систем і додатків, дозволяючи обмінюватися інформацією та функціональністю через мережу. Одним з відомих прикладів використання веб-сервісів є RESTful API, який застосовує HTTP-методи, такі як GET, POST, PUT, DELETE, для взаємодії з онлайн-ресурсами.
- Веб-сайт (website) - набір веб-сторінок, які зберігаються на веб-сервері і доступні для перегляду через Інтернет. Веб-сайт може містити текст, зображення, відео, аудіо, інші мультимедійні вміст, посилання на інші веб-сторінки та інші інтерактивні елементи.
- Доменне ім'я (domain name) - унікальне іменування, яке використовується для ідентифікації ресурсів (у тому числі і веб-сайтів) в Інтернеті. Доменні імена використовуються для заміни числових IP-адрес, що ідентифікують конкретні сервери або комп'ютери в мережі.
- Головний сервер - безпосередньо сервер, на якому зберігаються всі файли веб-сайту, на ньому розташовується веб-сервіс для обробки запитів до веб-сайту.
- База даних - організована колекція даних, яка зберігається на сервері і використовується для зберігання і управління інформацією, яка відображається або використовується на веб-сайті. База даних грає важливу роль в різних аспектах функціонування веб-сайту, таких як зберігання користувальницької інформації, контенту, налаштувань та багато інших.

Оскільки за приклад взято побудову інфраструктури для роботи веб-сайту, то згідно статистики[12] станом на 2023 з понад 200 мільйонів активних веб-сайтів у світі близько 80 мільйонів з них використовують системи управління контентом (CMS), при цьому дві третіх з усіх сайтів у світі користуються технологію системи управління контентом більшою чи меншою мірою, тому під час розгортання інфраструктури у хмарі є сенс урахувати також процес установки системи управління контентом на сервер.

Система управління контентом - це програмне забезпечення або платформа, яка дозволяє користувачам створювати, редагувати, оновлювати та організовувати вміст на веб-сайті без необхідності глибоких знань в галузі програмування або веб-розробки. CMS спрощує процес створення та керування веб-сайтами, забезпечуючи інтерфейс для редагування контенту, який може використовувати навіть людина без технічних навичок.

Основні характеристики CMS включають:

- Система керування контентом: CMS дозволяє керувати текстом, зображеннями, відео, аудіо та іншими елементами веб-сайту.
- Шаблони і дизайн: є можливість встановлення готових дизайнів або створити власні, щоб налаштувати зовнішній вигляд веб-сайту.
- Розділення прав доступу: CMS дозволяє призначати різні рівні доступу для користувачів, обмежуючи їхні можливості редагування та публікації контенту.
- Управління ресурсами: завантаження та керування різними ресурсами, такими як зображення та документи.
- Пошук та навігація: CMS забезпечує можливість пошуку контенту на сайті та створення зручної навігації.

Популярні приклади CMS включають Joomla, Drupal, Magento, WordPress, який є найбільш поширеним серед систем управління контентом, оскільки 43.1%

веб-сайтів, що використовують CMS користуються саме WordPress[12], та багато інших.

Таким чином основними задачами для дослідження ж наступні пункти:

- розробити нову (чи модифікувати існуючу) модель для процесу розгортання інфраструктури у хмарі для програмного продукту на прикладі системи управління контентом для веб-сайту.
- реалізувати частину розробленої моделі за допомогою підходу інфраструктури як коду та контейнерної оркестрації.

Висновки за розділом 1

Завершуючи аналіз сучасного стану розвитку технології хмарних обчислень, можна підкреслити, що ця галузь переживає період значного зростання та еволюції. Хмарні технології змінюють спосіб, яким підприємства здійснюють обробку та зберігання даних, пропонуючи надзвичайну масштабованість, економію коштів, розширену доступність, автоматичні оновлення та просте аварійне відновлення.

За даними Foundry's 2023 Cloud Computing research, на сьогоднішній день популярність хмарних обчислень становить 52%, що перевищує використання локальних середовищ (48%). Очікується, що до 2025 року кількість інфраструктур, які використовують хмарні обчислення, зросте до 60%. Це підкреслює важливість зрозуміння тенденцій у хмарних технологіях та вибору найкращого підходу для конкретних бізнес-завдань.

Проте, існують і певні виклики та недоліки, з якими стикаються підприємства при впровадженні хмарних рішень, включаючи питання безпеки та конфіденційності даних, відповідність нормативним вимогам, прив'язку до конкретного постачальника та складнощі міграції.

Архітектура хмарних обчислень представлена трьома основними моделями: SaaS (Software as a Service), PaaS (Platform as a Service) та IaaS (Infrastructure as a Service), кожна з яких має свої унікальні переваги та специфікації. Популярність хмарних технологій призвела до розробки нових концепцій та методів взаємодії з хмарним середовищем, включаючи безсерверні обчислення (Serverless Computing), оркестрацію контейнерів, постійну інтеграцію/постійне розгортання (CI/CD) та інші.

Хмарні технології також відіграють важливу роль у розвитку веб-сайтів, зокрема у розгортанні інфраструктур для їх підтримки. Системи управління контентом (CMS), як WordPress, стали популярними інструментами для створення та управління веб-сайтами. Використання CMS у поєднанні з хмарними технологіями забезпечує гнучкість, масштабованість та ефективність управління веб-ресурсами.

Загалом, хмарні обчислення продовжують впливати на IT-індустрію, пропонуючи підприємствам нові можливості для зростання та інновацій. Важливо враховувати як переваги, так і потенційні виклики, які вони представляють, для розробки стратегій, які найкраще відповідають потребам та цілям організації.

РОЗДІЛ 2
МОДЕЛЮВАННЯ ПРОЦЕСУ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ У ХМАРІ

2.1. Загальна модель процесу автоматизації розгортання інфраструктури у хмарі

З урахуванням компонентів, перелічених у розділі 1, які повинні бути встановлені та коректно налаштовані пропонується модель (див. рис. 2.1) для репрезентації процесу автоматичного розгортання інфраструктури для системи управління контентом у хмарі.

Дана модель розроблена у нотації IDEF0 (частина серії IDEF), оскільки ця нотація є однією з найпопулярніших для моделювання рішень, систем та процесів[26]. Вона широко використовується у бізнес-аналізі та інженерії для проектування програмного забезпечення.

Розроблена модель має наступні характеристики відповідно до IDEF0 нотації:

- Структура діаграми: модель складаються з ієрархічно організованих вузлів, які представляють функції або процеси в системі.
- Блоки: кожна функція або процес відображається у вигляді блоку з унікальним номером. Блоки мають назви, які коротко описують функцію.

Батьківська діаграма на рисунку 2.1 має два батьківські блоки, які репрезентують основні процеси, які мають бути реалізовані для даної моделі, а саме:

- розгорнути інфраструктуру – основний батьківський блок, що відповідає за процес розгортання інфраструктури у хмарі відповідно до наданих специфікацій та параметрів. Цей блок виконує як розгортання системи управління контентом і бази даних, так і базового сервера та мережі.

- оновити інфраструктуру – додатковий батьківський блок, який не є напряму залежним від блоку «розгорнути інфраструктуру», проте взаємодіє з результатом, який отримується на виході роботи блоку «розгорнути інфраструктуру», і є важливим компонентом для даної моделі.

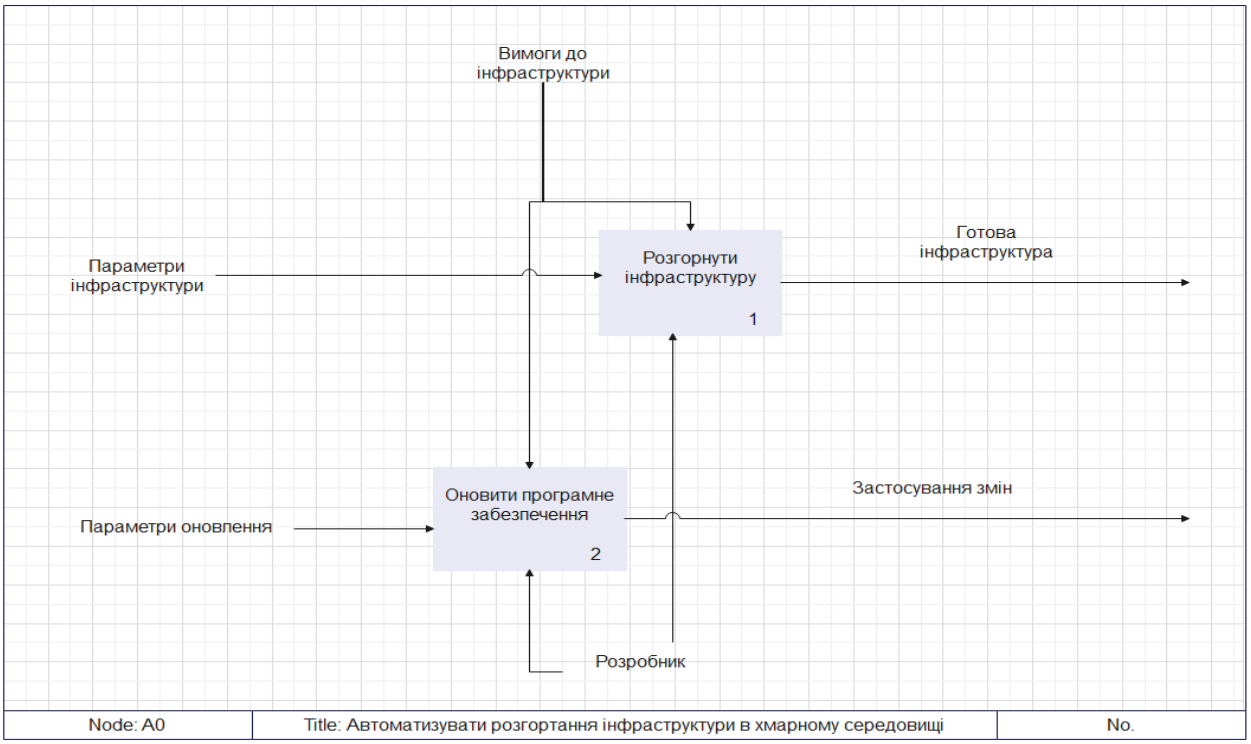

Рис. 2.1. – IDEF0 діаграма процесу автоматизації розгортання інфраструктури у хмарі.

Стрілки:

Входи: показують, що потрібно для виконання завдання блоку. Вони входять зліва в блок.

Для батьківського блоку 1 «розгорнути інфраструктуру» входом є параметри для інфраструктури, і у відповідності до наданих параметрів на вході розгортається інфраструктура[26]. Цими параметрами можуть бути версії системи управління контентом, версії бази даних, специфікація мережі, IP адреси, специфікації резервного копіювання, домен, який буде використовуватися для доступу до веб-сайту, операційна система для сервера,

кількість оперативної пам’яті на сервері, кількість дискового простору та інші параметри, які мають бути враховані.

Для батьківського блоку 2 «оновити інфраструктуру» входом є параметри для компонентів, які потрібно оновити в інфраструктурі. Цими параметрами можуть бути версії системи управління контентом, бази даних, оновлення програмного забезпечення на сервері, а також інші параметри, оновлення яких потрібно контролювати.

Виходи: показують, що блок виробляє або створює. Вони виходять справа з блоку.

Виходом для блоку 1 є готова інфраструктура, створена у відповідності до наданих параметрів. Тобто на вході отримуються де-які параметри, а на виході результатом є інфраструктура, створена згідно цих параметрів.

Виходом для блоку 2 є застосування змін (оновлень) у інфраструктурі, тобто результатом виконання цього блоку є процес оновлення компонентів відповідно до вхідних параметрів.

Керівні стрілки: показують умови або обмеження, які керують виконанням функції. Вони входять зверху в блок.

Для блоків 1 і 2 умови та обмеження вводяться безпосередньо в сам блок і не вказуються безпосередньо як параметри входу. Такими умовами можуть бути наприклад обмеження постачальника хмарних послуг на програмне забезпечення базового сервера, обмеження на IP адреси для мережі, обмеження на версії бази даних та системи управління контентом тощо.

Механізми: показують, якими ресурсами або засобами виконується функція. Зазвичай такими ресурсами є людські ресурси або ж обладнання. Вони входять знизу в блок.

Основним механізмом для блоку 1 та 2 батьківської діаграми є розробник, який використовує відповідні інструменти розробки, потрібні для реалізації функціоналу батьківських блоків.

Декомпозиція: кожен батьківський блок розкладається на діаграму нижчого рівня, що дає змогу аналізувати модель на більш детальному рівні. Це створює ієрархічну структуру з багатьма рівнями деталізації.

Розглянемо кожен батьківський блок більш детально у вигляді дочірніх IDEF0 діаграм.

Для батьківського блоку 1 «розгорнути інфраструктуру» дочірня діаграма зображена на рисунку 2.2.

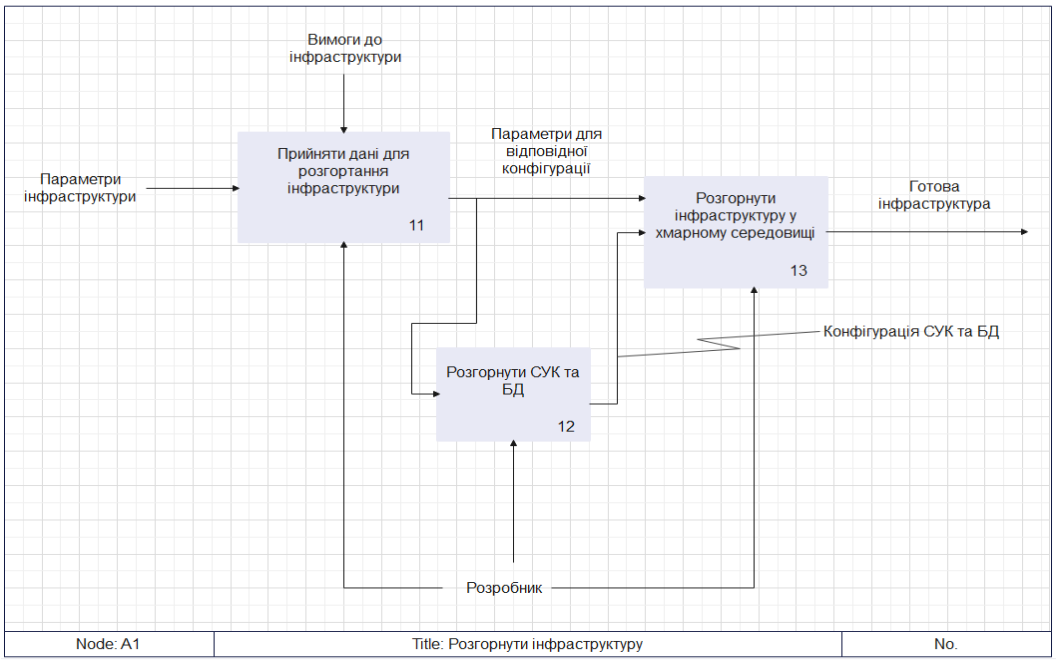

Рис. 2.2. – IDEF0 діаграма процесу розгортання інфраструктури у хмарі.

Діаграма A1 (див. рис. 2.2) складається з вузлів, які репрезентують собою під-процеси, що деталізують батьківський блок 1 «розгорнути інфраструктуру» з батьківської діаграми A0 (див. рис. 2.1).

Кожен процес відображений у вигляді окремого блоку, і має свою назву у відповідності до основної задачі, яка має бути виконана у рамках даного блоку.

Блок 11 - "Прийняти дані для розгортання інфраструктури", є під-процесом, для батьківського блоку 1 «розгорнути інфраструктуру» діаграми A0 (див. рис. 2.1) який забезпечує обробку вхідних даних для подальшого розгортання інфраструктури. Він відображає собою де-який інтерфейс для

взаємодії з користувачем. Головним завданням даного блоку є прийом всіх потрібних параметрів, згідно яких має бути створена інфраструктура. Далі отримані параметри мають бути перенаправлені як входи до блоків 12 та 13. Також блок виступає тригером для початку процесів у блоках 12 та 13 і має керівну стрілку «Вимоги до інфраструктури», оскільки на рівні блоку 11 виставляються обмеження щодо вхідних параметрів.

Блок 12 - "Розгорнути СУК та БД"[26], це під-процес розгортання системи управління контентом (СУК) та бази даних відповідно до специфікацій. Входом для даного блоку є вихід з блоку 11, наприклад версії системи управління контентом та бази даних, а на виході ми отримуємо конфігурації, згідно яких буде виконана установка СУК та БД.

Блок 13 - "Розгорнути інфраструктуру у хмарному середовищі"[26], це кінцевий та найважливіший під-процес, який відповідає за розгортання інфраструктури. Входом для даного блоку є виходи з блоків 11, тобто специфікація мережі, IP адреси, операційна система тощо, та блоку 12, що надає потрібну конфігурацію для установки системи управління контентом та бази даних для неї.

Для батьківського блоку 2 «оновити інфраструктуру»[26] дочірня діаграма зображена на рисунку 2.3.

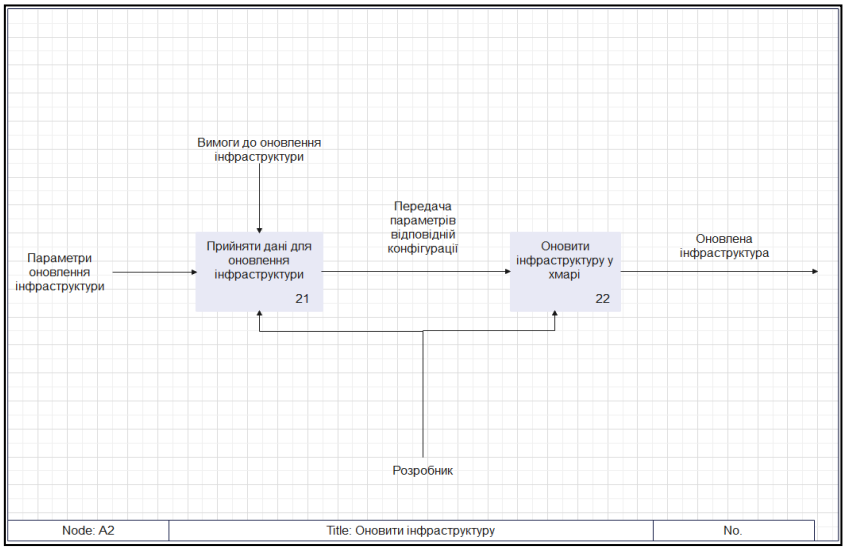

Рис. 2.3. – IDEF0 діаграма процесу оновлення інфраструктури у хмарі.

Блок 21 для цієї діаграми виконує аналогічну функцію, як і блок 11 з діаграми A1 (див. рис. 2.2), тобто є інтерфейсом, за допомогою якого користувач має змогу виконати потрібні оновлення[26]. Входом для цього блоку є параметри для компонентів інфраструктури, що потребують оновлення (такими параметрами можуть бути оновлення для операційної системи, пакетів програмного забезпечення сервера, бази даних або системи управління контентом). Виходом цього блоку є передача параметрів для блоку 22.

Блок 22 це ще один дочірній блок для блоку 2 діаграми A0 (див. рис. 2.1). Цей блок є репрезентацією самого процесу оновлення компонентів інфраструктури у відповідності до того, які параметри були надані для блоку 21[26].

Таким чином ми маємо базову модель для автоматизації інтеграції інфраструктури для системи управління контентом у хмарне середовище. Базуючись на даній моделі є можливість для створення автоматизації у відповідності до вимог, а також до модернізації архітектури на основі описаних діаграм. Також дана модель може бути основою для розробки детального плану різних імплементацій, які можуть включати розробку модулів для інтеграції з хмарними сервісами. Це дозволить підвищити гнучкість та масштабованість всієї інфраструктури для системи управління контентом, забезпечуючи більш ефективне управління ресурсами в хмарному середовищі, у порівнянні з ручною установкою.

2.2. Деталізація моделі процесу автоматизації розгортання інфраструктури у хмарі

На основі загальної базової моделі, розробленої у нотації IDEF0 (див. рис. 3) можна конкретизувати кожен етап автоматизації. Для цього розроблено

відповідну UML діаграму класів (див. рис. 2.4) [27] та діаграму діяльності (див. рис. 2.5), що надає можливості підібрати інструментарій для реалізації програмного продукту на основі базової моделі, а також проаналізованих підходів з розділу 1.

Діаграма класів (див. рис. 2.4), являє собою структуру класів та методів, що використовуються для інтеграції, оновлення та управління компонентами інфраструктури для системи управління контентом. В цій діаграмі класів системою управління контентом виступає найпопулярніша – WordPress. База даних – MySQL, оскільки це одна з найбільш використовуваних баз даних, а також вона має сумісність з WordPress.

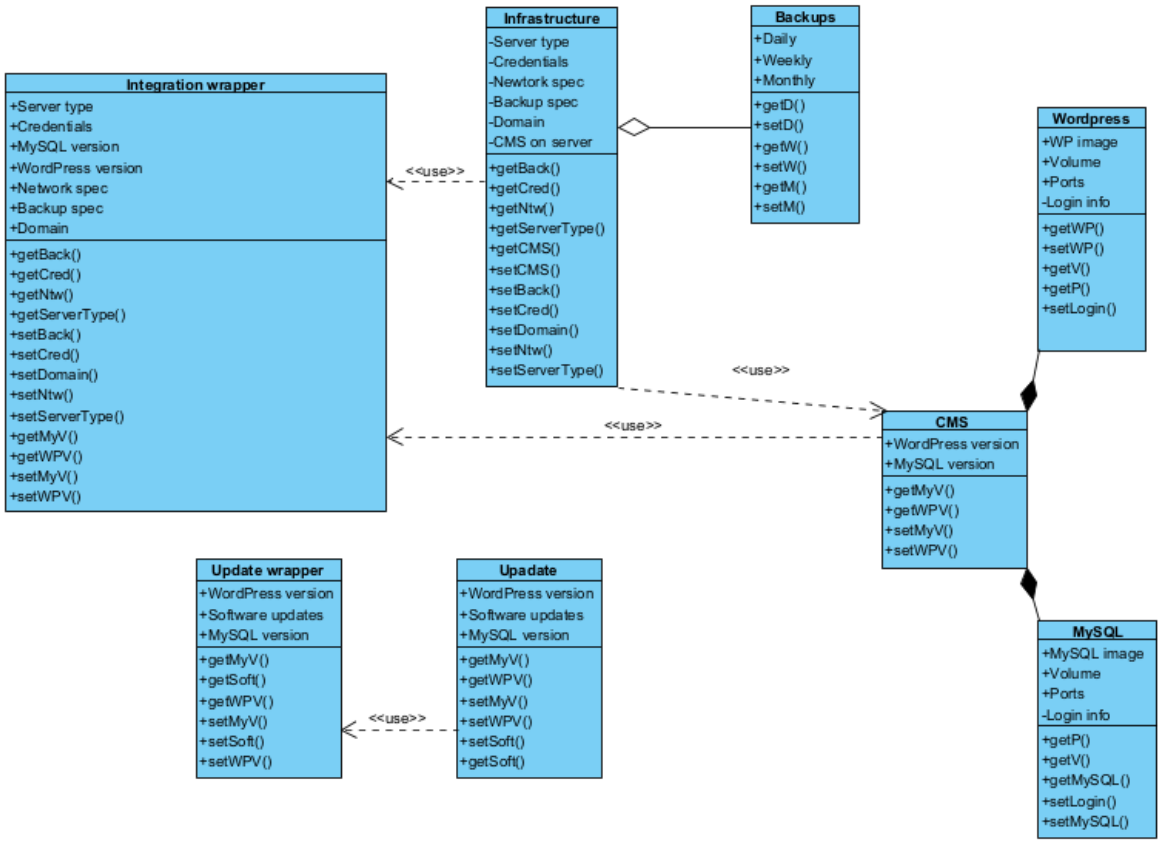

Рис. 2.4. – UML діаграма класів процесу розгортання інфраструктури для CMS.

Клас Integration Wrapper[27]

Цей клас слугує як обгортка для інтеграції різних компонентів системи. Він має наступні атрибути:

- Server type: тип головного сервера. Може включати операційну систему, кількість оперативної пам’яті, дискового простору тощо.
- Credentials: дані, які потрібні будуть для аутентифікації до сервісів, які цього потребують (ім'я користувача, пароль тощо)
- MySQL version: версія бази даних MySQL.
- WordPress version: версія системи управління контентом WordPress.
- Network spec: характеристики мережі, можливо, IP-адреси, DNS налаштування тощо.
- Backup spec: специфікації для резервного копіювання, можливо, частота, розташування тощо.
- Domain: доменне ім'я, асоційоване з сервером або системою управління контентом.

Методи, що входять до складу цього класу, дозволяють отримувати та встановлювати вищезазначені атрибути (get і set методи). Ці методи можуть бути використані для конфігурації сервера, наприклад, для зміни версії MySQL або WordPress, зміни домену, або специфікації резервних копій.

Зв'язки:

Асоціація з класом Infrastructure показує, що Integration Wrapper використовує або може впливати на ресурси, які керуються Infrastructure. Оскільки Integration Wrapper репрезентує собою інтерфейс користувача, то після того, як параметрі надаються до класу Integration Wrapper він надає відповідні параметри класу Infrastructure.

Асоціація з CMS означає, що Integration Wrapper ініціює відповідні параметри надані користувачем класу CMS.

Клас Update Wrapper[27]

Інтерфейс користувача для передачі параметрів у класі, відповідальні установку оновлень.

Атрибути:

- WordPress version: версія системи управління контентом.
- Software updates: які пакети програмного забезпечення вимагають оновлення.
- MySQL version: версія бази даних.

Методи:

get/set для кожного атрибута, для того щоб отримувати та встановлювати значення відповідно.

Зв'язки:

Асоціація з класом Updates, що вказує на передачу параметрів до цього класу.

Клас Infrastructure[27]

Відображає загальну інфраструктуру і включає інформацію, яка є загальною для всієї інфраструктури, таку як серверні ресурси та CMS.

Атрибути:

- Server type: тип головного сервера. Може включати операційну систему, кількість оперативної пам’яті, дискового простору тощо.
- Credentials: дані, які потрібні будуть для аутентифікації до сервісів, які цього потребують (ім'я користувача, пароль тощо).
- Network spec: характеристики мережі, можливо, IP-адреси, DNS налаштування тощо.
- Backup spec: специфікації для резервного копіювання, можливо, частота, розташування тощо.
- Domain: доменне ім'я, асоційоване з сервером або системою управління контентом.

- CMS on server: конфігурація системи управління контентом та бази даних на головному сервері.

Методи:

get/set для кожного атрибута для можливості приймати та встановлювати відповідні параметри.

Зв'язки:

Пов’язаний відношенням агрегації з класом Buckups, оскільки цей клас можна вважати окремою частиною класу Infrastructure.

Класи WordPress, MySQL і CMS[27]

Ці класи представляють компоненти, специфічні для WordPress та MySQL, з методами для управління системою управління WordPress. Вони містять інформацію про версії пакетів відповідних компонентів, об'єм пам’яті, порти, та інформацію для входу, а також методи для отримання та встановлення цих атрибутів. Вони представлені як композиції для класу CMS, оскільки система управління контентом та база даних самі по собі є двома окремими компонентами інфраструктури, але працюють в тісній парі, тому пов’язані між собою через клас CMS.

Клас Backups[27]

Клас Backups управляє резервними копіями інфраструктури. Він включає атрибути для різних типів резервного копіювання та відповідні методи для їх встановлення. Це дозволяє автоматизувати процес створення резервних копій на щоденній, щотижневій або щомісячній основі.

Клас Update[27]

Цей клас зосереджений на оновленні версій WordPress та MySQL, їх компонентів, або ж програмного забезпечення цільового сервера. Він містять методи для отримання поточних версій та встановлення оновлень. Це дозволяє системі підтримувати актуальність програмного забезпечення, що є критично важливим для безпеки та функціональності всієї інфраструктури.

Загалом, ця діаграма класів представляє систему управління інфраструктурою для системи управління контентом, яка автоматизує багато аспектів розгортання, управління та оновлення серверів, які використовують WordPress та MySQL. Взаємодія між класами показує, як можна модульно розділити різні аспекти управління інфраструктурою, роблячи систему гнучкою і легкою для розширення або зміни.

Для забезпечення зрозумілості та стандартизації процесів, а також для ідентифікації місць потенційного поліпшення в інфраструктурних операціях та процесах управління конфігурацією розроблено UML діаграму діяльності (див. рис. 2.5).

В контексті наданої діаграми зображуються наступні характеристики процесу автоматизації:

Визначення робочого процесу: діаграма ілюструє конкретний процес налаштування та оновлення конфігурації системи, що включає валідацію параметрів, їх встановлення, оновлення конфігурації, з'єднання з хмарною інфраструктурою та створення екземплярів та CMS.

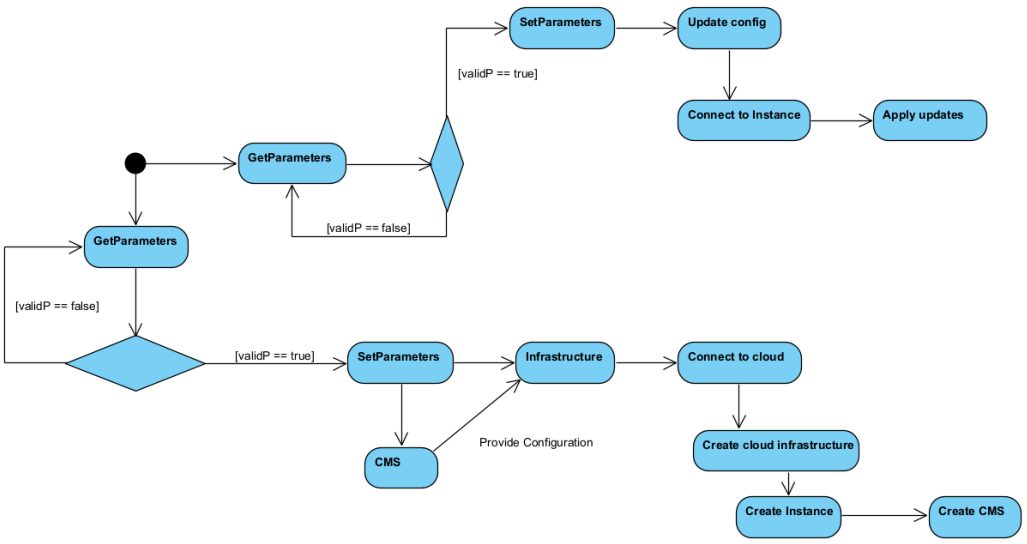

Рис. 2.5. – UML діаграма діяльності процесу розгортання інфраструктури для CMS.

Управління конфігурацією: показує, як система або додаток можуть бути налаштовані або оновлені, зокрема процедуру перевірки та встановлення конфігураційних параметрів.

Взаємодія між компонентами: діаграма пояснює, як різні компоненти системи (наприклад, CMS, хмарна інфраструктура) взаємодіють між собою.

Підтримка розгортання: слугує путівником для процесів розгортання, вказуючи на кроки потрібні для створення хмарної інфраструктури.

Роз'яснення логіки прийняття рішень: відповідно до діаграми можна виділити умови, які визначають потоки процесу автоматизації для різних випадків, наприклад, що робити, якщо параметри невалідні.

На діаграмі (див. рис. 2.5) зображено процес управління конфігурацією для автоматизації процесу розгортання інфраструктури у хмарі. Процес розпочинається з кроку "GetParameters", де здійснюється збір параметрів конфігурації.

Якщо параметри невалідні (isValidP == false), то процес повторюється. Якщо параметри валідні, вони передаються до кроку "SetParameters", де вони встановлюються.

Аналогічним чином діє крок "GetParameters" в контенсті процесу оновлення компонентів інфраструктури, проте різниця між ними полягає у функціональному навантаженні та відмінності параметрів, які можуть бути передані.

У нижній частині діаграми, після "SetParameters", є крок "Infrastructure" і "Connect to cloud", які вказують на з'єднання з хмарною інфраструктурою та її створення.

Завершальні кроки включають "Create cloud infrastructure" і "Create Instance", після чого відбувається створення інфраструктури та CMS.

У верхній частині діаграми, після встановлення параметрів, наступні кроки включають "Update config" і "Connect to Instance", конфігурація системи оновлюється, і здійснюється з'єднання з інстансом (екземпляром застосунку або сервісу).

Далі виконується крок "Apply updates", що виконує застосування оновлень конфігурацій, компонентів інфраструктури тощо.

Загалом, діаграма зображує процес, який включає збір параметрів, їх валідацію, встановлення та оновлення конфігурацій та самої інфраструктури, з'єднання з хмарним середовищем, і створення самої інфраструктури та системи управління контентом, що важливо для розгортання та управління застосунками у хмарному середовищі.

2.3. Підбір програмного забезпечення для реалізації моделі

В рамках даної кваліфікаційної роботи планується реалізувати основну частину розробленої моделі, яка буде включати:

- Інтерфейс користувача для спрощення взаємодії з основним функціоналом програми.
- Розгортання інфраструктури, що включає створення головного сервера, мережі, заходів безпеки для доступу на сервер.
- Розгортання системи управління контентом та бази даних на головному сервері.

Для реалізації програмного виробу на основі розробленої моделі в першу чергу потрібно обрати хмарне середовище, яке буде базовим для створення архітектури. З урахуванням сьогоднішніх тенденцій та популярності постачальників хмарних послуг (див. табл. 1, розділ 1) пропонується обрати платформу Amazon Web Services (AWS) як постачальника хмарних послух, і за допомогою ресурсів, які надаються для використання створити архітектуру.

Amazon Web Services має власну документацію, використання якої може допомогти почати працювати з цією платформою, а також проаналізувати її ресурси для обрання найбільш підходящих з точки зору поставленої задачі[13].

У загальному щоб почати користуватися Amazon Web Services (AWS), потрібно виконати кілька кроків:

Реєстрація Аккаунту AWS:

- Спочатку потрібно створити аккаунт AWS. Це можна зробити, відвідавши офіційний сайт AWS і вибравши опцію "Створити AWS аккаунт".
- В процесі реєстрації потрібно ввести свої контактні дані, інформацію про платіжну картку та обрати тарифний план. Це обов’язкова процедура, оскільки не зважаючи на те що багато сервісів AWS є безкоштовними, існують сервіси за які знімається оплата, хоча їх використання не обов’язкове.

Вивчення AWS Management Console:

- Після створення аккаунта, отримується доступ до AWS Management Console, який є головним інтерфейсом для управління AWS сервісами.
- Важливо провести час, досліджуючи консоль та різні доступні сервіси.

Вибір та налаштування Сервісів:

- AWS пропонує широкий спектр хмарних сервісів, таких як Amazon EC2 для віртуальних серверів, Amazon S3 для сховища даних, AWS Lambda для безсерверних обчислень та багато інших.

Використання SDK та API:

- AWS надає SDK (Software Development Kit) для різних мов програмування, що дозволяє легко інтегрувати AWS сервіси в ваші додатки.
- Також можна взаємодіяти з сервісами AWS через API, що дає можливість автоматизувати різні задачі.

Навчання та підтримка:

- AWS пропонує різноманітні ресурси для навчання, включаючи документацію, туторіали, вебінари та курси.

- Також доступна підтримка через форуми AWS та офіційну службу підтримки.

Безпека та Управління:

- Важливо налаштувати безпеку для аккаунту, включаючи налаштування правил доступу, шифрування даних тощо.
- AWS також надає інструменти для моніторингу та управління ресурсами.

Експериментування з Free Tier:

- AWS пропонує "Free Tier" для нових користувачів, що дозволяє використовувати певні сервіси безкоштовно протягом обмеженого часу.

Ці кроки є основними для ефективного використання AWS для хмарних обчислювальних потреб.

З урахуванням вище зазначених кроків мною було створено обліковий запис в Amazon Web Services під номером 913078705715, який буде використовуватись як базова платформа для розгортання інфраструктури.

Основними сервісами AWS, з якими планується взаємодія для реалізації моделі є наступні:

1. Amazon EC2 (Elastic Compute Cloud)[14] є однією з ключових послуг в рамках Amazon Web Services (AWS), яка надає масштабовані обчислювальні рішення в хмарі. Основні аспекти та можливості EC2:

Основні Властивості

- Віртуальні Сервери в Хмарі (Instances):
- EC2 дозволяє користувачам запускати віртуальні сервери, які називаються "instances". Ці сервери можна налаштувати відповідно до потреб користувача, вибираючи з різних конфігурацій апаратного забезпечення.

Гнучкість і Масштабування:

- EC2 забезпечує гнучкість у виборі конфігурацій (CPU, пам'ять, сховище), що дозволяє користувачам оптимізувати вартість та продуктивність.

- Масштабування ресурсів може бути автоматичним або вручну, в залежності від змін у навантаженні.

Безпека:

- EC2 надає розширені можливості для управління безпекою. Користувачі можуть використовувати групи безпеки та мережеві ACL для контролю доступу до instances.

Зберігання Даних:

- Для зберігання даних можна використовувати Amazon Elastic Block Store (EBS), що надає персистентне блокове сховище для EC2 instances.

Інтеграція з Іншими AWS Сервісами:

- EC2 легко інтегрується з іншими сервісами AWS, такими як Amazon S3, RDS, VPC та інші, що надає потужні можливості для розробки повномасштабних рішень.

Типи Instances

- Загального призначення: добре збалансовані характеристики CPU та пам'яті. Підходять для веб-серверів, додатків середнього розміру.
- Обчислювально оптимізовані: підходять для задач, що вимагають високої продуктивності обчислень, наприклад, наукових моделювань.
- Оптимізовані по пам’яті: ідеальні для задач, які вимагають великого обсягу пам'яті, наприклад, бази даних великого розміру.
- Оптимізоване по сховищу: для задач, які потребують високої швидкості читання/запису на диск, таких як NoSQL бази даних.

Використання сервісу:

- Веб-сервери / Хостинг додатків: запуск і управління веб-серверами, додатками.
- Обробка баз даних: використання для різних типів баз даних, включаючи транзакційні бази даних і великі бази даних.

- Наукове моделювання та великі обчислення: запуск обчислювально інтенсивних додатків.
- Резервне копіювання та сховище: використання EC2 для резервного копіювання та зберігання даних.
- EC2 надає потужні та гнучкі можливості для різноманітних обчислювальних завдань, роблячи його ключовим компонентом багатьох хмарних рішень.

Таким чином проаналізувавши можливості сервісу EC2, обираємо його як основний сервіс для розгортання головного сервера, на якому буде розташовано система управління контентом та база даних.

2. Amazon Virtual Private Cloud (VPC) є сервісом в Amazon Web Services (AWS), який дозволяє користувачам запускати ресурси AWS у віртуальній приватній мережі. Ця мережа віртуально відокремлена від інших віртуальних мереж в AWS. Основні аспекти та можливості VPC:

Ізоляція ресурсів:

- VPC дозволяє користувачам запускати AWS ресурси в ізольованій частині облачної мережі AWS, забезпечуючи контроль над мережевим середовищем.

Індивідуальна IP-адресація:

- В межах VPC, користувачі можуть вибирати свої власні діапазони IP-адрес (CIDR block), налаштовуючи мережеве середовище за своїми потребами.

Підмережі:

- В межах VPC можна створювати підмережі, які можуть бути публічними чи приватними. Це дозволяє розділяти різні типи ресурсів (наприклад, веб-сервери та бази даних) в різних сегментах мережі.

Інтернет-шлюзи та NAT:

- Інтернет-шлюзи дозволяють ресурсам у публічних підмережах з'єднуватися з інтернетом, а NAT (Network Address Translation) дозволяє

ресурсам у приватних підмережах доступ до інтернету, не надаючи інтернету доступу до цих ресурсів.

Безпека:

- VPC надає групи безпеки та списки контролю доступу (ACL), що дозволяє детально контролювати вхідний та вихідний мережевий трафік.

Приклади використання сервісу:

- Запуск веб-додатків: VPC може використовуватись для ізоляції та захисту веб-додатків у хмарі, розділяючи різні частини застосунків у різних підмережах.
- Розміщення баз даних: бази даних можуть бути розміщені у приватних підмережах з обмеженим доступом для підвищення безпеки.

Таким чином використання Amazon Virtual Private Cloud дозволить нам створити свою мережу та підмережі для розміщення компонентів архітектури та надання змоги відповідним ресурсам взаємодіяти з мережею Інтернет.

3. Amazon Identity and Access Management (IAM)[16] є ключовим компонентом в Amazon Web Services (AWS), що дозволяє ефективно керувати доступом до AWS ресурсів. IAM надає гнучкі інструменти для управління дозволами користувачів, груп, ролей та політик безпеки. Основні аспекти IAM:

Користувачі та групи:

- IAM дозволяє створювати окремі облікові записи для користувачів, які потребують доступу до AWS ресурсів.
- Користувачі можуть бути організовані у групи для спрощення управління дозволами.

Ролі та Політики:

- Ролі в IAM дозволяють делегувати дозволи на різні ресурси без необхідності створювати кілька облікових записів.
- Політики безпеки, які можуть бути прикріплені до користувачів, груп, та ролей, визначають дозволи на доступ до ресурсів AWS.

Багаторівнева аутентифікація (MFA):

- IAM підтримує використання MFA для додаткового рівня безпеки при доступі до AWS ресурсів.

Інтеграція з іншими AWS сервісами:

- IAM інтегрується з більшістю сервісів AWS, дозволяючи керувати доступом до цих сервісів на основі встановлених дозволів.

Журнали історії:

- IAM забезпечує детальні журнали доступу, що дозволяють відстежувати, хто, коли та як використовував AWS ресурси.

Приклади використання:

- Контроль доступу до ресурсів: наприклад, обмеження доступу до певних компонентів Amazon S3 або управління доступом до EC2 instances.
- Управління ролями для додатків: дозволяє додаткам, що працюють на EC2, отримувати доступ до інших AWS сервісів, якщо це необхідно для їх функціонування.
- Федерація користувачів: інтеграція з існуючими системами ідентифікації, що дозволяє користувачам використовувати їхні існуючі облікові записи для доступу до AWS ресурсів.
- Дотримання політик безпеки: створення та виконання політик безпеки, що вимагаються в організації або відповідно до вимог законодавства.

Безпека:

IAM є важливим інструментом для забезпечення безпеки в AWS, оскільки він дозволяє точно контролювати, хто має доступ до ресурсів і яким чином цей доступ використовується.

Отже, обравши постачальником хмарних обчислень платформу Amazon Web Services, я обрав сервіси Amazon EC2, Amazon Virtual Private Cloud та Amazon Identity and Access Management як основні для використання при реалізації розробленої моделі.

Використання вище описаних сервісів може відбуватися за допомогою засобів взаємодії з Amazon Web Services вручну, проте оскільки реалізація моделі інтеграції системи управління контентом передбачає автоматизацію процесу створення інфраструктури пропонується обрати програмне забезпечення, яке може виконувати розгортання інфраструктури самостійно.

Процес автоматизації розгортання інфраструктури пропонується реалізувати за допомогою методології інфраструктури як коду з залученням інструментарію програмного забезпечення Terraform[17]

Terraform – це інструмент для управління інфраструктурою як код (Infrastructure as Code, IaC), розроблений HashiCorp. Він дозволяє користувачам автоматизувати створення, зміну та управління інфраструктурою в різних сервісах облачних та локальних провайдерів. Terraform використовує декларативний підхід, де користувачі описують бажаний стан інфраструктури за допомогою конфігураційних файлів, а Terraform забезпечує реалізацію цього стану.

Основні властивості та можливості Terraform:

- Декларативні конфігурації: Terraform використовує конфігураційні файли (зазвичай у форматі HCL – HashiCorp Configuration Language), що описують компоненти інфраструктури, такі як віртуальні машини, мережі, сховища тощо.
- Підтримка багатьох провайдерів: Terraform може взаємодіяти з широким спектром провайдерів інфраструктури, включаючи Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, VMware, OpenStack та багато інших.
- Ідемпотентність: Terraform забезпечує ідемпотентність, що означає, що багаторазове застосування однієї і тієї ж конфігурації не впливатиме на стан інфраструктури, якщо цей стан вже досягнуто.

- Управління станом: Terraform веде облік поточного стану інфраструктури і використовує цю інформацію для планування змін, мінімізуючи ризик помилок при оновленні інфраструктури.
- Модульність: Terraform дозволяє створювати модульні конфігурації, що спрощують повторне використання коду та керування складними інфраструктурами.

Приклади використання:

- Автоматичне створення та управління компонентами інфраструктури, такими як сервери, мережі, бази даних.
- Управління складними та/або багатошаровими інфраструктурами для великих систем або додатків.
- Інтеграція з DevOps практиками для автоматизації процесів розгортання та забезпечення послідовності середовищ.
- Управління ресурсами у гібридних або багатохмарних середовищах, що включають ресурси з різних облачних провайдерів.

Переваги:

- Консистентність та передбачуваність: забезпечує однакове розгортання інфраструктури незалежно від середовища.
- Зниження ризику помилок: мінімізує людські помилки завдяки автоматизації процесів.
- Простота управління: спрощує управління складними інфраструктурами.

Terraform є могутнім інструментом для автоматизування та ефективного управління інфраструктурою, як у хмарних, так і в локальних середовищах.

Оскільки Terraform має можливість взаємодіяти з платформою Amazon Web Services, то планується використовувати його функціонал для реалізації розгортання інфраструктури у хмарному середовищі.

Інструментарію Terraform цілком і повністю достатньо для того, щоб реалізувати основну частину розробленої моделі, а саме розгортання інфраструктури, системи управління контентом та бази даних. Проте у такому випадку система управління контентом, база даних та інші компоненти, такі як веб-сервіс, мають бути установлені безпосередньо на головний сервер що робить написання програми більш комплексним, а також робить інфраструктуру менш гнучкою та масштабованою в цілому. Для вирішення питання швидкості розгортання компонентів системи управління контентом, масштабованості та гнучкості пропонується використати підхід контейнерної оркестрації, та розгортати систему управління контентом, базу даних та інші необхідні компоненти в ізольованих середовищах. Таким чином пропонується використання інструментарія Docker, який призначений для управління контейнерами як ізольованими середовищами з застосунками. Оркестратором пропонується обрати docker-compose, яка є частиною технології Docker, оскільки це надасть можливість більш централізованого керування. Системою управління контентом, яку потрібно інсталювати обрано WordPress, база даних – MySQL.

Розгортання WordPress та MySQL як Docker контейнерів (з використанням docker-compose) та безпосередньо на сервері (традиційний спосіб) відрізняються за кількома ключовими аспектами. Основні відмінності та переваги використання docker-compose.

Розгортання без Docker (Традиційний Спосіб)

- Пряма інсталяція: WordPress та MySQL встановлюються безпосередньо на хост-систему.
- Залежності: всі залежності (наприклад, PHP, Apache/Nginx веб сервсів) також встановлюються на головний сервер.
- Конфігурація: налаштування сервера, бази даних, та веб-сервера вимагає ручного втручання. В контексті автоматичного розгортання через

Terraform цей аспект збільшує комплексність програми а також вимагає додаткового контролю.

- Залежність від системи: сумісність та версії пакетів залежать від конкретної ОС головного сервера.

Розгортання з Docker Compose

- Контейнеризація: WordPress та MySQL розгортаються у вигляді контейнерів.
- Ізоляція: кожен контейнер має власне ізольоване середовище, що зменшує конфлікти залежностей.
- Легкість конфігурації: файл конфігурації оркестратора дозволяє визначити конфігурацію для кожного сервісу у єдиному файлі.
- Портативність: контейнери можуть легко переміщатися між різними системами.

Переваги Docker Compose

- Консистентність середовища: контейнери забезпечують однакове середовище в розробці, тестуванні та виробництві.
- Ізоляція ресурсів: кожен контейнер використовує власні ресурси та залежності, що знижує ризик конфліктів.
- Спрощене розгортання: Із docker-compose всі потрібні компоненти системи управління контентом і бази даних включно можуть бути розгорнуті за одну команду.
- Легкість масштабування: можливість легко масштабувати сервіси, збільшуючи кількість контейнерів.
- Спрощене оновлення та розгортання: оновлення та розгортання можуть бути виконані швидше та ефективніше.
- Легкість відновлення після збоїв: У випадку збою, контейнер може бути швидко перезапущений.

- Мережева конфігурація: Docker дозволяє легко налаштувати мережеві взаємодії між контейнерами.

Використання Docker та docker-compose для розгортання WordPress та MySQL надає значні переваги у термінах портативності, ізоляції, консистентності середовища та спрощення управління інфраструктурою. Це особливо корисно в розробці та тестуванні, де потрібна швидка настройка та ізоляція середовища. Враховуючи ці аспекти, а також можливості інструментарію Terraform, ми можемо автоматизувати процес розгортання інфраструктури у хмарному середовищі Amazon Web Services, і установки на головному сервері системи управління контентом WordPress та бази даних MySQL як ізольовних контейнерів під оркестрацією docker-compose.

Останнім аспектом, який потрібно врахувати під час реалізації програмного продукту – це інтерфейс користувача, який дозволить інтерактивно передавати необхідні параметри до Terraform. Суть полягає в тому, що без подібного інтерфейсу програму потрібно запускати безпосередньо за допомогою засобів Terraform, які можуть бути не зрозумілими користувачеві, який прагне використовувати функціонал програми для розгортання інфраструктури.

Для реалізації користувацього інтерфейсу пропонується використання мови програмування Python.

Python є потужною мовою програмування, яка підтримує створення широкого спектру застосунків, включаючи інтерфейси для взаємодії з інструментами управління інфраструктурою, такими як Terraform. Python відомий своєю простотою, читабельністю коду та широкою підтримкою бібліотек, що робить його відмінним вибором для автоматизації та інтеграції різних систем.

Інтеграція Python з Terraform

Для інтеграції Python з Terraform можна використовувати бібліотеку python_terraform[18]. Ця бібліотека дозволяє використовувати Python для

управління і взаємодії з Terraform, дозволяючи створювати, змінювати та управляти інфраструктурою через Terraform за допомогою Python скриптів.

Висновки за розділом 2

Таким чином для реалізації розробленої моделі процесу розгортання інфраструктури для системи управління контентом було обрано хмарну платформу Amazon Web Services (AWS) через її гнучкість, розмаїття сервісів та можливості, а також через найбільш поширене використання з поміж інших поставників хмарних обчислень. Основними сервісами AWS, що будуть використані у програмі, є Amazon EC2, Amazon Virtual Private Cloud (VPC) та Amazon Identity and Access Management (IAM).

Реалізація також включає використання Terraform – інструменту для управління інфраструктурою як коду, що дозволяє автоматизувати створення, зміну та управління інфраструктурою. Terraform був обраний через його декларативний підхід, ідемпотентність, модульність та підтримку багатьох провайдерів інфраструктури.

Для вирішення питань швидкості розгортання, масштабованості та гнучкості системи було вирішено використовувати підхід контейнерної оркестрації, розгортаючи WordPress та MySQL у Docker контейнерах із застосуванням Docker Compose як оркестратора. Це забезпечить консистентність середовищ, ізоляцію ресурсів, спрощення розгортання та легкість масштабування.

Останнім етапом стане розроблення інтерфейсу користувача на мові програмування Python з використанням бібліотеки python_terraform. Цей інтерфейс дозволяє інтерактивно передавати необхідні параметри до Terraform, забезпечуючи автоматизацію процесу розгортання інфраструктури в AWS та інсталяції системи управління контентом та бази даних.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОЇ МОДЕЛІ НА ОСНОВІ ОБРАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В ході розробки програми було створено UML діаграму розгортання (див. рис. 3.1), на якій продемонстровано, яким чином програму було розроблено, які конфігураційні файли вона містить, та яка інфраструктура розгортається. Програмний код кожного конфігураційного файлу зображено в додатку Г.

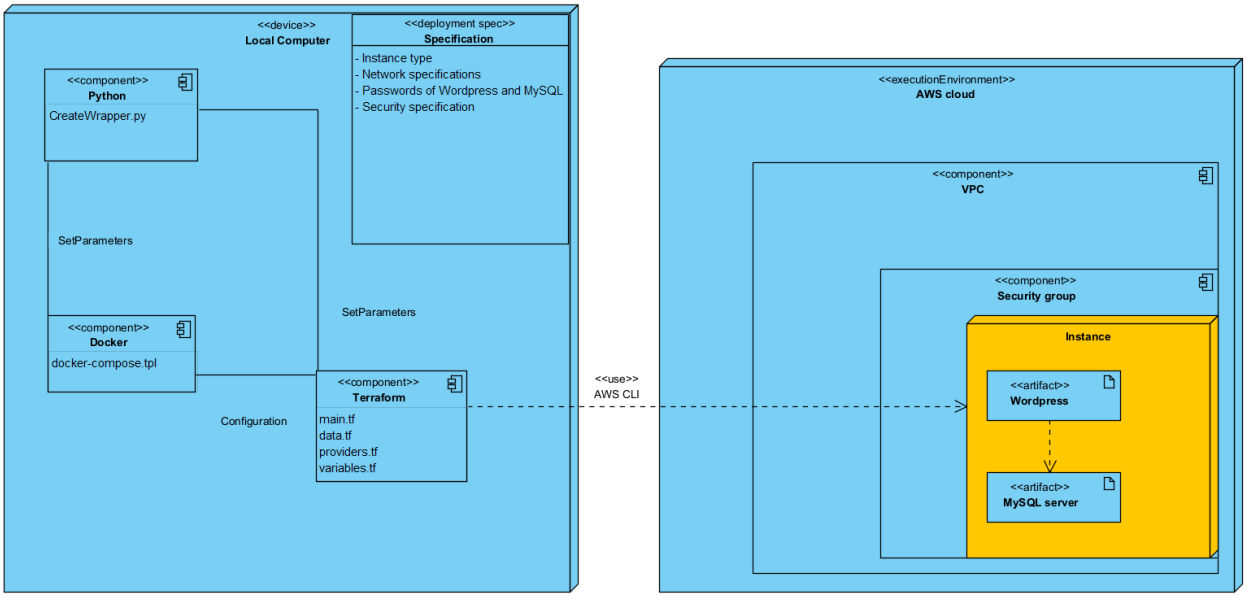

Рис. 3.1. – UML діаграма розгортання інфраструктури для CMS в платформі AWS.

Зліва на діаграмі вказані локальні компоненти, які використовуються для створення та конфігурації інфраструктури. Справа показано, як ці компоненти розгортаються в хмарному середовищі AWS.

Local Computer це девайс, на якому проводиться розробка програми, запуск та тестування. Для цього використовувався ноутбук Lenovo ideapad 330S-15IKB GTX1050, з процесором Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz, з типом 64-розрядної операційної системи Windows 11, процесор на базі архітектури x64. Середовищем, де створено конфігураційні файли обрано

Microsoft Visual Studio. Запуск програми та конфігурація доступу до хмарної платформи проводиться на базі Ubuntu 20.04.5 WSL.

Параметри, які було визначено як специфікації розгортання включають:

- Тип екземпляру (тип сервера згідно визначення AWS[19]. Якщо параметр пропускається, то застосовується тип сервера “t2.micro”)
- Мережеві специфікації (IP адреса мережі та публічної підмережі, що передаються за методом безкласової маршрутизації, за замовчуванням має значення 10.0.0.0/16 для всієї мережі, 10.0.10.0/24 для підмережі, імена для регіонів, де розгортається мережа, за замовчуванням один регіон eu-central-1a)
- Паролі для Wordpress та MySQL, ім’я бази даних для WordPress.
- Специфікації безпеки (IP адреси, з яких планується SSH підключення до головного сервера, передаються за методом безкласової маршрутизації, за замовчуванням має значення 0.0.0.0/0, ім’я для публічного SSH ключа, який буде розміщений на сервері, ім’я для групи безпеки - Security Group [20], IP адреси, з яких планується підключення до бази даних, передаються за методом безкласової маршрутизації, за замовчуванням має значення 0.0.0.0/0).

Локальні конфігураційні файли (див. рис. 3.1) включають:

1. CreateWrapper.py: цей скрипт на мові програмування Python використовує модуль python_terraform, щоб автоматизувати процес розгортання інфраструктури в AWS через Terraform. Код виконує наступний функціонал:

- Збір вхідних даних користувача: скрипт запитує у користувача ряд параметрів для налаштування інфраструктури, таких як CIDR блок VPC, CIDR публічних підмереж, назву групи безпеки, діапазон IP-адрес для SSH-з'єднання, назву SSH ключа, тип екземпляра сервера тощо. Якщо користувач нічого не вводить, використовуються значення за замовчуванням.
- Створення словника змінних: всі зібрані дані зберігаються в словнику variables, де ключами є назви параметрів, а значеннями - введені користувачем дані.

- Ініціалізація об'єкта Terraform: створюється об'єкт tf класу Terraform з модуля python_terraform, який вказує на робочий каталог (де знаходяться файли Terraform) та передає словник змінних.
- Планування змін (Terraform Plan): виконується команда plan, яка показує, які зміни будуть внесені в інфраструктуру. Вивід команди перевіряється на наявність помилок і виводиться на екран.
- Застосування змін (Terraform Apply): виконується команда apply, яка фактично вносить зміни в інфраструктуру. Команда автоматично затверджує зміни (без запиту підтвердження від користувача) і знову ж таки, вивід перевіряється на наявність помилок.

Якщо виникають помилки під час виконання команд plan або apply, скрипт виводить повідомлення про помилку та детальну інформацію про неї.

Візуальне зображення інтерфейсу продемонстроване на рисунку 3.2.

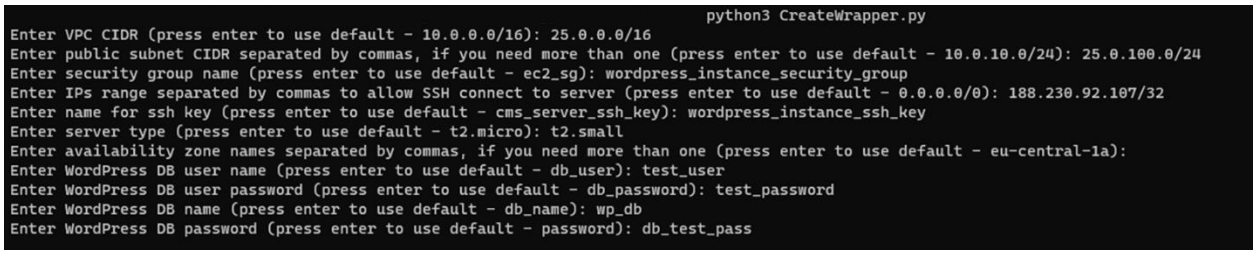

Рис. 3.2. – Інтерфейс користувача розробленої програми.

Приклад частини виводу інтерфейсу при успішному виконанні програми зображено на рисунку 3.3.

У разі, якщо параметри вводяться не вірно, або існують інші проблеми, які заважають виконанню програми, в консоль виводиться конкретна помилка, чому не вдається розгорнути інфраструктуру. Приклад помилки виконання зображено на рисунку 3.4.

```
# module.vpc.aws_vpc.this[0] will be created
+ resource "aws_vpc" "this" {
    + arn                                  = (known after apply)
    + cidr_block                           = "25.0.0.0/16"
    + default_network_acl_id               = (known after apply)
    + default_route_table_id               = (known after apply)
    + default_security_group_id            = (known after apply)
    + dhcp_options_id                      = (known after apply)
    + enable_dns_hostnames                 = true
    + enable_dns_support                   = true
    + enable_network_address_usage_metrics = (known after apply)
    + id                                   = (known after apply)
    + instance_tenancy                     = "default"
    + ipv6_association_id                  = (known after apply)
    + ipv6_cidr_block                      = (known after apply)
    + ipv6_cidr_block_network_border_group = (known after apply)
    + main_route_table_id                  = (known after apply)
    + owner_id                             = (known after apply)
    + tags                                 = {
        + "Name"      = "cms-vpc"
        + "Terraform" = "true"
      }
    + tags_all                             = {
        + "Name"      = "cms-vpc"
        + "Terraform" = "true"
      }
  }
```

Рис. 3.3. – Приклад частини виводу при успішному виконанні програми.

```
Enter VPC CIDR (press enter to use default - 10.0.0.0/16):
Enter public subnet CIDR separated by commas, if you need more than one (press enter to use default - 10.0.10.0/24):
Enter security group name (press enter to use default - ec2_sg):
Enter IPs range separated by commas to allow SSH connect to server (press enter to use default - 0.0.0.0/0):
Enter name for ssh key (press enter to use default - cms_server_ssh_key):
Enter server type (press enter to use default - t2.micro): t3.micro
Enter availability zone names separated by commas, if you need more than one (press enter to use default - eu-central-1a): test
Enter WordPress DB user name (press enter to use default - db_user):
Enter WordPress DB user password (press enter to use default - db_password):
Enter WordPress DB name (press enter to use default - db_name):
Enter WordPress DB password (press enter to use default - password):
Error in Terraform plan execution.

Error in Terraform apply execution.

Error: creating EC2 Subnet: InvalidParameterValue: Value (test) for parameter availabilityZoneId is invalid. Subnets can currently
az3.
       status code: 400, request id: cb6f0203-4847-4879-9df0-28cddeb97b1b

  with module.vpc.aws_subnet.public[0],
  on .terraform/modules/vpc/main.tf line 97, in resource "aws_subnet" "public":
  97: resource "aws_subnet" "public" {
```

Рис. 3.4. – Приклад частини виводу при помилці виконання програми.

docker-compose.tpl: це файл конфігурації docker-compose, який використовується для створення та запуску контейнерів Docker для двох сервісів: WordPress (системи управління контентом, та всіх необхідних для її роботи додаткових компонентів) і MySQL (бази даних). Зазвичай для docker-compose використовується формат файлу YAML, проте оскільки ця конфігурація використовується в комбінації з Terraform, то обрано формат tpl, тобто шаблон.

Це дозволяє змінювати параметри для конфігурації оркестратора а також взаємодіяти з Terraform. Конфігураційний файл складається з наступних частин:

Версія: визначає версію синтаксису файлу docker-compose. Тут використовується версія '3.8'. Версія синтаксису за документацією docker-compos повинна визначатися обов’язково[21]

Сервіси: ця секція визначає два сервіси: wordpress і mysql.

WordPress:

- image: wordpress:latest - використовує останню версію образу WordPress з Docker Hub (репозиторій для образів, на основі яких створюються контейнери [22]).
- ports: - "80:80" – направляє порт 80 із контейнера на порт 80 головного сервера, щоб WordPress був доступний через веб-браузер.
- environment - визначає змінні середовища для конфігурації WordPress, такі як дані для підключення до бази даних MySQL.
- volumes: - wordpress:/var/www/html - використовує том для зберігання даних WordPress, щоб дані зберігались навіть після зупинки або видалення контейнера.

MySQL:

- image: mysql:5.7 - використовує версію 5.7 образу MySQL з Docker Hub.
- ports: - "3306:3306" – направляє порт 3306 із контейнера на порт 3306 головного сервера, щоб MySQL був доступний через будь-який клієнт.
- environment - встановлює змінні середовища для MySQL, включаючи пароль root-користувача та дані для створення нової бази даних та користувача.
- volumes: - mysql:/var/lib/mysql - використовує том для зберігання даних MySQL, щоб дані зберігались після зупинки або видалення контейнера.

Томи: ця секція визначає томи, які використовуються сервісами. Тут вони названі wordpress і mysql. Таким чином дані, якими оперує система управління

контентом та база даних зберігаються безпосередньо на сервері, що запобігає втраті інформації у разі переривання роботи контейнерів.

Файл використовує змінні середовища (наприклад, ${wordpress_db_user}), які визначаються при запуску інтерфейсу користувача CreateWrapper.py та передаються до шаблону конфігурації docker-compose.

У результаті при успішному розгортанні всієї інфраструктури, головний інтерфейс є доступним за протоколом HTTP та IP сервера, де пропонуються кроки для остаточної конфігурації системи управління контентом (див. рис. 3.5).

Остаточна конфігурація має проводитись самим користувачем, оскільки вона вимагає вводу імені головного адміністратора та поштової адреси.

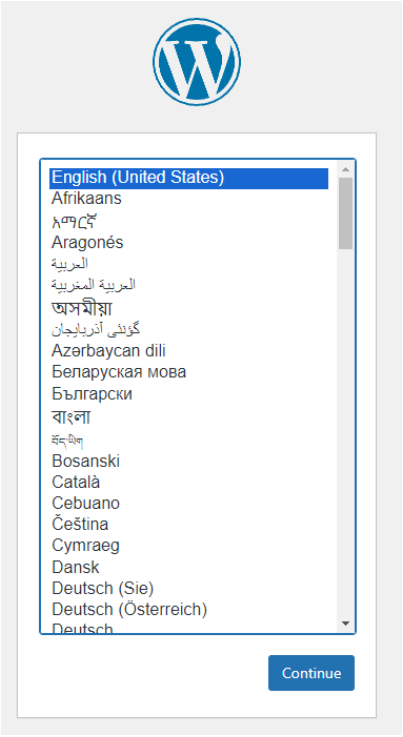

Рис. 3.5. – Сторінка конфігурації системи управління контентом для адміністратора.

Після цієї конфігурації сторінка розробника стає доступною, і система управління контентом вважається готовою до експлуатації (див. рис. 3.6).

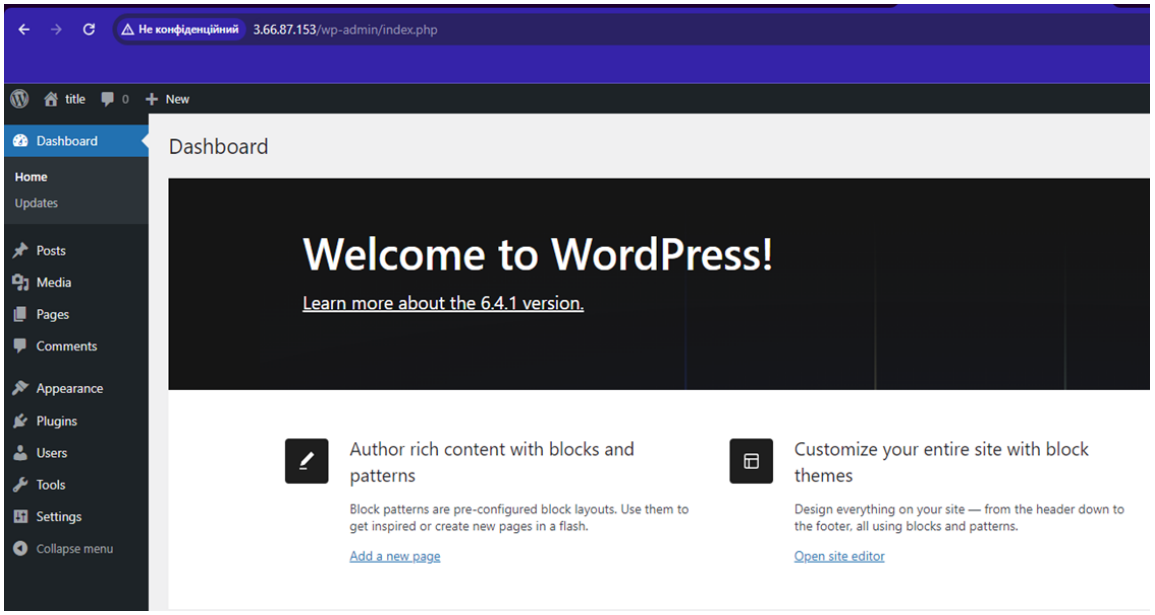

Рис. 3.6. – Інтерфейс розробника в системі управління контентом.

Конфігураційні файли Terraform: main.tf, data.tf, providers.tf, variables.tf є основою створення інфраструктури.

В першу чергу, для того щоб Terraform мав змогу взаємодіяти з AWS було створено відповідного користувача за допомогою IAM сервісу, та створено ключі доступу до ресурсів AWS через інтерфейс командного рядка для AWS – AWS CLI[23]. Таким чином Terraform використовуючи AWS CLI матиме змогу змінювати інфраструктуру використовуючи дані створеного користувача. Цей користувач повинен мати права доступу на взаємодію з необхідними ресурсами хмарного середовища.

Файл providers.tf: містить короткий фрагмент коду, який є частиною конфігурації для Terraform. Цей конфігураційний файл включає наступне:

- Provider: У Terraform, provider використовується для визначення платформи або сервісу, на якому буде розгортатися інфраструктура. У даному випадку, використовується провайдер aws, який вказує на те, що Terraform буде управляти ресурсами в Amazon Web Services (AWS). Ключі ж доступу з міркувань безпеки не прописуються безпосередньо в коді програми. Для цього в

середовищі Ubuntu 20.04.5 WSL, яке використовується для запуску програми, було виконано конфігурацію AWS CLI, де були вказані потрібні ключі (див. рис. 3.7). Terraform із вказаним провайдером AWS має змогу самостійно перевіряти конфігурацію AWS CLI, таким чином використовуючи ключі доступу до хмарного середовища.

```
AWS Access Key ID [****************526W]:
AWS Secret Access Key [****************aIqe]:
Default region name [eu-central-1]:
Default output format [json]:
```

Рис. 3.7. – Конфігурація доступу до AWS за допомогою AWS CLI.

Конфігурація провайдера: в середині блоку провайдера є параметри конфігурації, які визначають, як Terraform повинен взаємодіяти з AWS. В даному випадку, вказаний тільки один параметр: region = "eu-central-1". Цей параметр вказує регіон AWS, де будуть розгортатися ресурси[24]. eu-central-1 означає Центральний регіон Європи (Франкфурт).

Цей код не включає в себе визначення самих ресурсів, які будуть створені. Він лише налаштовує Terraform для роботи з AWS у вказаному регіоні.

Файл variables.tf: Цей код є частиною конфігурації Terraform, інструменту для управління інфраструктурою як код (Infrastructure as Code). Він описує оголошення змінних, які можуть бути використані в інших частинах Terraform-конфігурації. Кожна змінна має наступні визначення:

- vpc_cidr: встановлює CIDR-блок для віртуальної приватної хмари (VPC) в AWS. Значення "10.0.0.0/16" означає, що для VPC використовується діапазон IP-адрес з префіксом /16 у приватному діапазоні 10.x.x.x. Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- public_subnets_cidr: визначає список CIDR-блоків для публічних підмереж. "10.0.10.0/24" означає одну підмережу з діапазоном /24 у VPC. Це

значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.

- sg_name: встановлює назву для групи безпеки, яка може бути використана для екземплярів EC2. У цьому випадку, назва групи безпеки - "ec2_sg". Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- ssh_connect_ips: визначає список IP-адрес або CIDR-блоків, які мають дозвіл на SSH-підключення. "0.0.0.0/0" означає, що дозволено підключення з будь-якої IP-адреси. Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- db_connect_ips: визначає список IP-адрес або CIDR-блоків, які мають дозвіл на підключення до бази даних. "0.0.0.0/0" означає, що дозволено підключення з будь-якої IP-адреси. Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- ssh_key_name: вказує назву SSH ключа, який використовується для доступу до екземплярів EC2. У цьому випадку, назва ключа - "cms_server_ssh_key". Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- instance_type: визначає тип екземпляра EC2, який буде використовуватися. "t2.micro" - це невеликий і економічний тип екземпляра в AWS. Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- azs: визначає список зон доступності (Availability Zones), які будуть використовуватися. "eu-central-1a" - це одна з зон у регіоні Центральної Європи (Франкфурт). Це значення встановлюється за умовчування, якщо користувач не ввів власне значення за допомогою інтерфейсу.
- wordpress_db_user, wordpress_db_password, wordpress_db_name: визначають відповідно користувача, пароль та назву бази даних для WordPress.

- mysql_root_password: встановлює пароль для користувача root бази даних MySQL.

Ці змінні використовуються в інших частинах Terraform-конфігурації для динамічного управління ресурсами без необхідності використання змінних напряму.

Доступ до бази даних може відбуватися з будь-якого клієнта, якщо IP користувача знаходить в списку дозволених (див. рис. 3.8) та з іменем користувача та паролем, який був наданий під час розгортання інфраструктури.

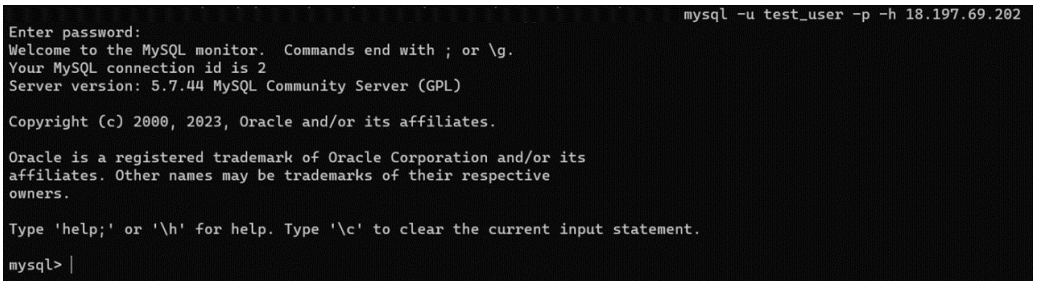

Рис. 3.8. – Успішне підключення до бази даних.

На рисунку 3.9 зображено інтерфейс MySQL через командний рядок, де можна побачити, що в для доступу в MySQL сервіс обраній базі даних wp_db (ім’я бази вводиться користувачем під час розгортання інфраструктури) вже є таблиці для системи управління контентом, завантажені за замовчуванням, що свідчить про коректність розгортання інфраструктури.

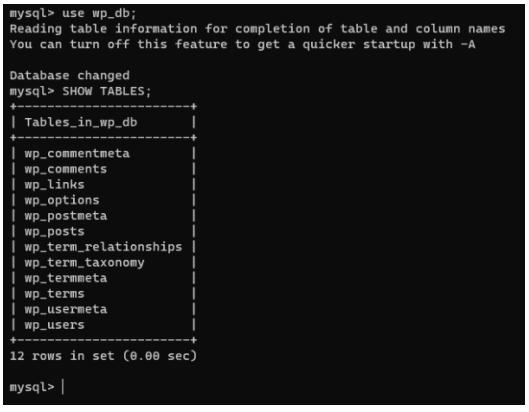

Рис. 3.9. – Інтерфейс командного рядка MySQL.

Файл data.tf: цей код є частиною конфігурації TerraformУ коді представлені два блоки даних: template_file і aws_ami. Вони не визначають, які ресурси будуть створені в рамках інфраструктури, проте визначають додаткові дані, які будуть використовуватися Terraform.

- template_file "docker_compose": цей блок визначає шаблон файлу, який Terraform буде обробляти.
- template - Вказує на шлях до файлу шаблону. ${path.module} вказує на поточний каталог модуля Terraform, і docker-compose.tpl це назва файлу шаблону.
- vars - Цей розділ містить змінні, які передаються в шаблон. У цьому випадку, це змінні для налаштування WordPress та MySQL, такі як ім'я користувача бази даних, пароль, назва бази даних, та пароль root для MySQL.
- aws_ami "this" - цей блок використовується для пошуку AMI (Amazon Machine Image) в AWS.
- most_recent = true - означає, що Terraform повинен шукати найновіший доступний образ.
- filter - ці фільтри використовуються для уточнення пошуку AMI. Перший фільтр шукає образи з ім'ям, що відповідає шаблону "amzn2-ami-hvm-*", що є образами Amazon Linux 2 з підтримкою HVM (Hardware Virtual Machine). Другий фільтр вибирає тільки образи, які знаходяться у стані "available".
- owners = ["amazon"] - означає, що шукаються тільки образи, які належать Amazon. Таким чином ми матимемо на сервері операційну систему Amazon Linux 2, і базова конфігурація сервера обирається автоматично на основі шаблонів створених самим AWS.

Загалом, цей код Terraform використовується для налаштування інфраструктури, зокрема, для обробки шаблону файлу конфігурації та пошуку підходящих образів AMI у AWS.

Файл main.tf:

Цей код Terraform описує конфігурацію інфраструктури в AWS для запуску веб-сервера з використанням Docker та Docker Compose. Він складається з кількох частин, кожна з яких відповідає за окремий аспект інфраструктури:

Модуль VPC (Virtual Private Cloud):

Використовується для створення VPC в AWS, використовуючи зовнішній модуль Terraform terraform-aws-modules/vpc/aws[25]. Змінна name встановлює назву VPC, а cidr визначає CIDR-блок VPC (визначається змінною var.vpc_cidr). Змінні azs та public_subnets визначають зони доступності та CIDR-блоки для публічних підмереж. tags використовуються для маркування VPC в AWS.

На рисунку 3.10 зображено створену мережу та підмережу у хмарному середовищі, згідно переданих параметрів.

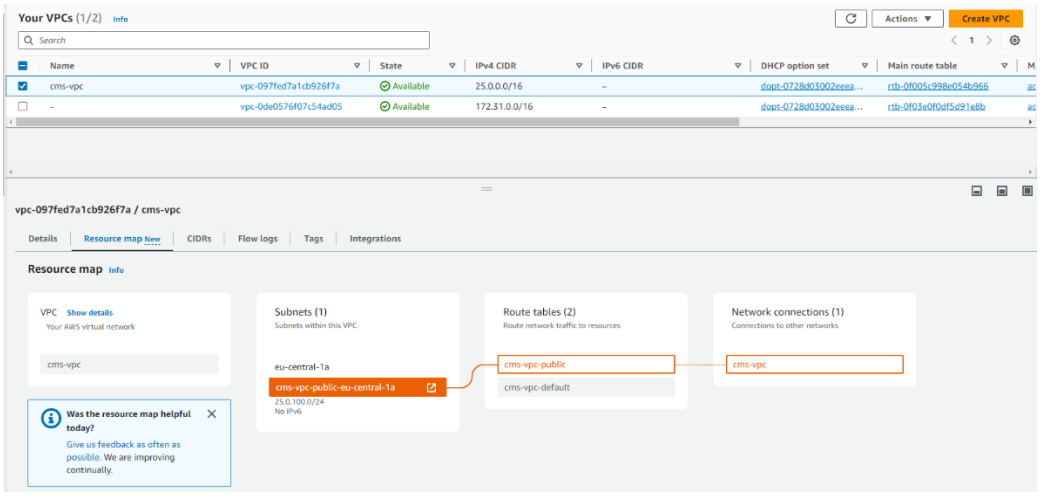

Рис. 3.10. – Створена мережа та підмережа в AWS.

Ресурс AWS Security Group "ec2_sg":

- Створює групу безпеки для контролю доступу до екземплярів EC2.
- Визначає правила для вхідного трафіку (ingress) для портів 80 (HTTP), 443 (HTTPS), 22 (SSH) та 3306 для доступу до бази даних(див. рис. 3.11). Встановлює правило для вихідного трафіку (egress) для всіх портів та адрес.

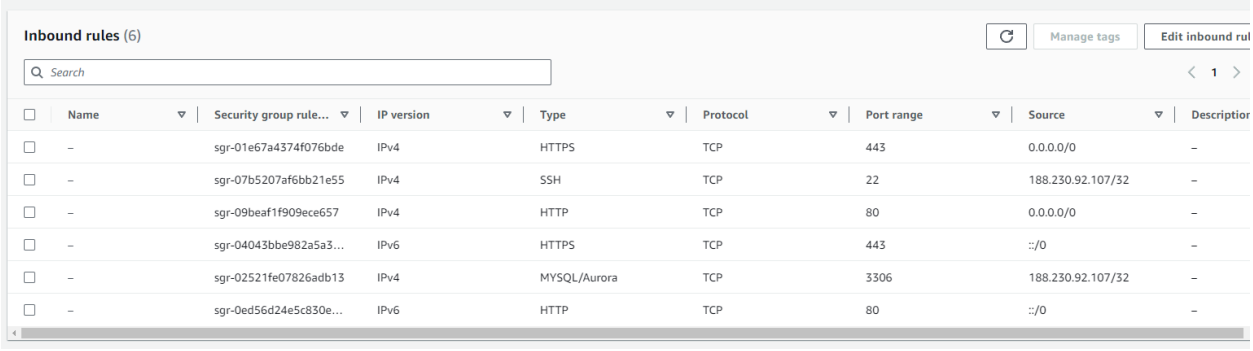

Рис. 3.11. – Група безпеки для сервера.

Ресурс "local_file" для файлу Docker Compose:

Створює локальний файл зі згенерованим вмістом Docker Compose конфігурації. Суть полягає в тому, що через шаблон docker-compose.tpl створюється файл docker-compose.yaml, який можна запустити локально для випробування конфігурації оркестратора локально, і цей де файл передається наголовний сервер у інфраструктурі.

На рисунку 3.12 зображено контейнери Docker з базовою інформацією про них. Статус UP свідчить про успішний запуск та роботу системи управління контентом та бази даних в окремих контейнерах.

```
[ec2-user@ip-10-0-10-43 ~]$ docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS                                                  NAMES
020dac2cf8e2   mysql:5.7          "docker-entrypoint.s…"   7 minutes ago   Up 7 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   ec2-user-mysql-1
3c9fb53a352a   wordpress:latest   "docker-entrypoint.s…"   18 minutes ago  Up 7 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp                      ec2-user-wordpress-1
```

Рис. 3.12. – Контейнери системи управління контентом та бази даних.

Ресурс "aws_key_pair" для SSH ключа:

Створює пару ключів SSH в AWS для доступу до екземплярів EC2. У даному випадку безпосередньо для цієї реалізації SSH ключі було згенеровано локально за допомогою функціоналу Ubuntu 20.04.5 WSL.

Ресурс "aws_instance" для екземпляра EC2 "web":

Використовує AMI (Amazon Machine Image), визначене в блоку data.aws_ami.this. Встановлює тип екземпляра, ключ SSH, ID підмережі, ID групи безпеки та активує публічну IP-адресу. Таким чином створюється головний сервер.

Містить два блоки provisioner:

- Перший блок встановлює Docker і Docker Compose на екземплярі EC2.
- Другий блок переміщує файл docker-compose.yaml на головний сервер та запускає Docker Compose.

Для налаштування з'єднання через SSH для конфігурації екземпляра використовується блок connection.

Базове описання створеного сервера можна побачити на рисунку 3.13. Згідно рисунку видно, що серверу присвоюється публічна IP адреса та DNS ім’я, за якими можна звертатися до сервісі на сервері. Так, доступ до системи управління контентом може відбуватися через веб-браузер та IP адресою сервера, доступ до бази даних може відбуватися через клієнт бази даних із вказівкою IP сервера як головного розміщення бази даних. Також на зображенні видно, що сервер знаходиться у вказаному через інтерфейс регіоні, має вказаний тип сервера тощо. Таким чином можна сказати, що програма успішно виконує призначену функцію.

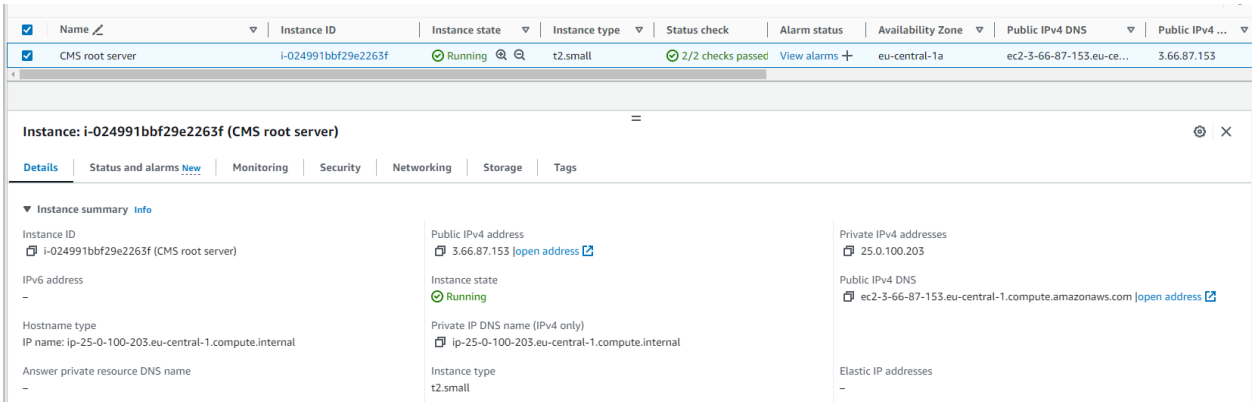

Рис. 3.13. Контейнери системи управління контентом та бази даних.

Ця сукупність конфігурацій Terraform є основною в програмі, що дозволяє автоматизувати процес створення і налаштування інфраструктури в AWS для системи управління контентом та бази даних.

Висновки за розділом 3

У розділі 3 було описано процес розробки програмного продукту для розгортання інфраструктури системи управління контентом (CMS) на платформі Amazon Web Services (AWS). Основні аспекти, описані у цьому розділі, включають:

Створення UML діаграми розгортання: діаграма демонструє процес розробки, конфігураційні файли та інфраструктуру, яка розгортається.

Конфігураційні параметри розгортання: описуються специфікації, такі як тип екземпляра, мережеві специфікації, паролі для Wordpress та MySQL, та специфікації безпеки.

Локальні конфігураційні файли: розглядаються файли, такі як CreateWrapper.py, який використовує Python та Terraform для автоматизації процесу розгортання.

Конфігурація Docker: використовується файл конфігурації docker-compose для створення та запуску контейнерів Docker для WordPress та MySQL.

Файли Terraform: описуються основні конфігураційні файли Terraform, такі як main.tf, data.tf, providers.tf, variables.tf, які використовуються для створення та налаштування інфраструктури в AWS.

Запуск інфраструктури: Описується процес створення інфраструктури, включаючи налаштування VPC, ресурсів безпеки, локальних файлів та екземплярів EC2.

Загалом, цей розділ детально описує процес розробки та розгортання програмного продукту для CMS на AWS, включаючи всі аспекти від планування до реалізації, з акцентом на використання таких інструментів, як Docker, Python, та Terraform.

ВИСНОВКИ

У цій кваліфікаційній роботі представлено глибоке дослідження хмарних технологій, розробку моделі процесу інтеграції системи управління контентом у хмарне середовище та розробку програмного продукту, спрямованого на розгортання інфраструктури системи управління контентом у хмарному сервісі Amazon Web Services (AWS). В роботі були використані сучасні технології та методології для створення ефективного та масштабованого рішення. Розглянуті основні аспекти включають:

Створення діаграм для представлення моделі: це надало змогу візуалізувати структуру та взаємозв'язки між компонентами системи, що спрощує розуміння та подальшу розробку на основі створеної моделі.

Детальний опис конфігураційних параметрів розгортання: описано важливі параметри, які включають типи екземплярів, мережеві налаштування, а також параметри безпеки для Wordpress та MySQL, що забезпечує гнучкість та відповідність специфічним потребам користувачів.

Робота з локальними конфігураційними файлами: описано використання файлів, таких як CreateWrapper.py, який за допомогою Python та Terraform дозволяє автоматизувати процес розгортання, знижуючи потенційні людські помилки та підвищуючи ефективність процесу.

Конфігурація Docker: використання Docker дозволило створювати та управляти контейнерами для WordPress та MySQL, забезпечуючи високий рівень ізоляції та стандартизації середовища.

Файли Terraform: детально описано використання Terraform для створення та налаштування інфраструктури в AWS, що демонструє гнучкість та масштабованість рішення.

Запуск інфраструктури: описано процес створення інфраструктури, що включає налаштування VPC, ресурсів безпеки, локальних файлів, та екземплярів EC2, що є ключовим для успішного розгортання системи.

На додаток можна зазначити, що в ході виконання даної кваліфікаційної роботи мої розробки були представлені на науково-технічних конференціях КМНТ-2022 та КМНТ-2023.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cloud computing study 2023: веб-сайт. URL: https://foundryco.com/tools-for-marketers/research-cloud-computing/ (дата звернення: 10.10.2023).

2. Cloud vs. On-Premises: Pros, Cons, and Use Cases: веб-сайт. URL: https://www.datamation.com/cloud/cloud-vs-on-premises-pros-cons-and-use-cases/ (дата звернення: 10.10.2023).

3. Cloud Deployment Models: веб-сайт. URL: https://www.geeksforgeeks.org/cloud-deployment-models/?ref=lbp (дата звернення: 10.10.2023).

4. Hazzaa N. Alshareef: Current Development, Challenges and Future Trends in Cloud Computing: A Survey. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 14, No. 3, 2023. P. 329-338: URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://thesai.org/downloads/volume14no3/paper_37-current_development_challenges_and_future_trends.pdf (дата звернення: 12.10.2023).

5. Top 10 Cloud Computing Companies of 2023: веб-сайт. URL: https://www.knowledgehut.com/blog/cloud-computing/top-cloud-computing-companies (дата звернення: 13.10.2023).

6. How infrastructure as code (IaC) manages complex infrastructures: веб-сайт. URL: https://www.atlassian.com/microservices/cloud-computing/infrastructure-as-code (дата звернення: 15.10.2023).

7. What is container orchestration?: веб-сайт. URL: https://www.ibm.com/topics/container-orchestration (дата звернення: 15.10.2023).

8. What is serverless?: веб-сайт. URL: https://www.ibm.com/topics/serverless (дата звернення: 16.10.2023).

9. What is CI/CD?: веб-сайт. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (дата звернення: 17.10.2023).

10. How does a website work?: веб-сайт. URL: https://www.hostitsmart.com/manage/knowledgebase/205/how-does-a-website-work.html (дата звернення: 17.10.2023).

11. Domain Email and Website Hosting Articles: веб-сайт. URL: https://www.doteasy.com/domain-email-and-website-hosting-articles/how-do-websites-work (дата звернення: 18.10.2023).

12. Exploring the Latest CMS Statistics and Its Impact on Content Management: веб-сайт. URL: https://techjury.net/blog/cms-statistics/ (дата звернення: 18.10.2023).

13. Welcome to AWS Documentation: веб-сайт. URL: https://docs.aws.amazon.com/ (дата звернення: 20.10.2023).

14. Amazon Elastic Compute Cloud Documentation: веб-сайт. URL: https://docs.aws.amazon.com/ec2/?icmpid=docs_homepage_featuredsvcs (дата звернення: 20.10.2023).

15. Amazon Virtual Private Cloud Documentation: веб-сайт. URL: https://docs.aws.amazon.com/vpc/?icmpid=docs_homepage_featuredsvcs (дата звернення: 20.10.2023).

16. AWS Identity and Access Management Documentation: веб-сайт. URL: https://docs.aws.amazon.com/iam/ (дата звернення: 20.10.2023).

17. Terraform About the Docs: веб-сайт. URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 22.10.2023).

18. python-terraform 0.10.1: веб-сайт. URL: https://pypi.org/project/python-terraform/ (дата звернення: 24.10.2023).

19. Instance types: веб-сайт. URL: https://docs.aws.amazon.com/awsec2/latest/userguide/instance-types.html (дата звернення: 20.10.2023).

20. Amazon EC2 security groups for Linux instances: веб-сайт. URL: https://docs.aws.amazon.com/awsec2/latest/userguide/ec2-security-groups.html (дата звернення: 20.10.2023).

21. Docker Compose overview: веб-сайт. URL: https://docs.docker.com/compose/ (дата звернення: 25.10.2023).

22. Overview of Docker Hub: веб-сайт. URL: https://docs.docker.com/docker-hub/ (дата звернення: 25.10.2023).

23. What is the AWS Command Line Interface?: веб-сайт. URL: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html (дата звернення: 26.10.2023).

24. Regions, Availability Zones, and Local Zones: веб-сайт. URL: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html (дата звернення: 20.10.2023).

25. Terraform module which creates VPC resources on AWS: веб-сайт. URL: https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest (дата звернення: 28.10.2023).

26. Мірошниченко Д.О, Толстолузька О.Г, Петриченко М.О. Модель автоматизації розгортання інфраструктури для системи управління контентом в хмарному середовищі: збірник наукових праць Міжнародної науково-технічної конференції «Комп'ютерне моделювання у наукоємних технологіях». – Х. : ХНУ імені В.Н. Каразіна, 2022. – С. 144- 148.

27. Мірошниченко Д.О, Толстолузька О.Г. Аналіз підходів до розробки моделі автоматизації розгортання інфраструктури для системи управління контентом: збірник наукових праць Міжнародної науково-технічної конференції «Комп'ютерне моделювання у наукоємних технологіях». – Х. : ХНУ імені В.Н. Каразіна, 2023.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: **15 – Автоматизація та приладобудування**
Спеціальність: **151 – «Автоматизація та комп'ютерно-інтегровані технології»**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної та прикладної системотехніки
д.т.н., проф. Шматков С. І.

«08 » грудня 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мірошниченко Дмитро Олександрович
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Модель процесу інтеграції системи управління контентом в платформу Amazon Web Services»
керівник роботи Толстолузька Олена Геннадіївна, доктор технічних наук, с.н.с.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «10» листопада 2023 року № 4101-5/3197

2. Строк подання студентом роботи 28 .11.2023

3. Перелік питань, які потрібно розробити)
1) Аналіз проблеми та постановка задачі створення моделі процесу установки системи управління контентом в платформу Amazon Web Services
2) Аналіз методів для розгортання інфраструктури у хмарному середовищі
3) Розробка моделі процесу інтеграції системи управління контентом в хмарне середовище
4) Дослідження технологій для реалізації моделі
5) Створення програмного виробу на основі обраних технологій для реалізації моделі процесу установки системи управління контентом
6) Розробка рекомендацій щодо застосування розробленої моделі і програмного виробу

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Підбір літератури, створення бази для розробки моделі	08.12.2022-01.03.2023
2	Порівняння обраного підходу з аналогами, аналіз переваг і недоліків	02.03.2023-15.05.2023
3	Створення моделі	16.05.2023-20.06.2023
4	Аналіз технологій для реалізації моделі	21.06.2023-10.06.2023
5	Розробка програмного продукту на основі обраних технологій	10.06.2023-15.08.2023
6	Випробування програмного виробу, виправлення помилок.	16.08.2023-20.09.2023
7	Аналіз результатів випробувань	21.09.2023-30.09.2023
8	Розробка рекомендацій щодо застосування програмного виробу.	01.10.2023-10.10.2023
9	Підготовка до попереднього захисту роботи	11.10.2023-30.10.2023
10	Створення пояснювальної записки	2.11.2023-15.11.2023
11	Представлення кваліфікаційної роботи науковому керівнику	16.11.2023-28.11.2023

5. Дата видачі завдання 08.12.2022

Студент Д.О. Мірошниченко
ініціали, прізвище


підпис

Керівник роботи О.Г. Толстолузька
ініціали, прізвище


підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання

на розробку програмного виробу
«МОДЕЛЬ ПРОЦЕСУ ІНТЕГРАЦІЇ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ В ПЛАТФОРМУ AMAZON WEB SERVICES»

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва програмного виробу: модель процесу інтеграції системи управління контентом в платформу amazon web services 1.2. Галузь застосування: автоматизація та приладобудування
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології» 2.2. Завдання на кваліфікаційну роботу № 4101-5/3197 від “10” листопада 2023 року (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	3.1. Мета розробки програмного виробу: розробка моделі процесу інтеграції інфраструктури для системи управління контентом підвищення рівня ефективності, гнучкості та масштабованості процесу створення хмарних інфраструктур за допомогою підходів інфраструктури як коду та контейнерної оркестрації. 3.2. Призначення програмного виробу: репрезентації можливості впровадження автоматизації для розгортання інфраструктури у хмарному середовищі. Дана модель спрямована на розширення переліку концепцій взаємодії з хмарними платформами, що дає змогу впроваджувати більш ефективні підходи для планування розробки архітектур програмного забезпечення. 3.3. Вихідні дані для розробки: вихідними

	даними для розробки моделі слугує ручне розгортання інфраструктур для програмного продукту у хмарному середовищі.
4. Технічні вимогидо програмного виробу	4.1. Вимоги до функціональних характеристик: можливість автоматично розгортати інфраструктуру у хмарному середовищі згідно заданих параметрів. 4.2. Вимоги до надійності: забезпечення розгортання інфраструктури при різних параметрах, відмова до розгортання при передачі параметрів неправильного формату. 4.3. Вимоги до умов експлуатації: обліковий запис у платформі AWS, встановлення програмного забезпечення Terraform, Docker-compose та мови програмування Python з необхідними бібліотеками. 4.4. Вимоги до складу і параметрів технічнихзасобів: комп'ютер або ноутбук з 4 ГБ оперативної пам'яті, процесором не нижче Intel(R) Core(TM) i5-8250U. 4.5. Вимоги до інформаційної та програмної сумісності: підтримка ОС Linux або Windows10/11, підтримка Python 3, Terraform 1.6.4. 4.6. Вимоги до маркування та упаковки: відсутні. 4.7. Вимоги до транспортування і зберігання: відсутні. 4.8. Спеціальні вимоги: стабільний доступ до мережі Інтернет.
5. Вимоги до програмної документації.	Програмною документацією до виробу «модель процесу інтеграції системи управління контентом в платформу amazon web services» вважати: 1) Справжнє Технічне завдання на розробку програмного виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до кваліфікаційної роботи).

<table>
<tr><td></td><td colspan="2">3) Опис програмного виробу (представити в розділі пояснювальної записки до кваліфікаційної роботи).
4) Текст програми (представити в Додатку Г до пояснювальної записки до кваліфікаційної роботи).</td></tr>
<tr><td>6. Техніко-економічні показники</td><td colspan="2">Якісна оцінка: у порівнянні з ручним розгортанням аналогічної інфраструктури програмна реалізація не допускає помилок, які можуть бути спричинені людським фактором, оскільки виконання відбувається програмно, то на будь-якому етапі, де могли б виникати помилки та неточності при ручному розгортанні – вони анулюються. Проблеми виникають тільки в разі невалідних параметрів, обмежень постачальника хмарних послуг на ресурси або ж при нестабільному з’єднанні з мережею інтернет.
Кількісна оцінка: в межах визначеної архітектури, ручне розгортання інфраструктури займає приблизно 60 хвилин. Розгортання ж інфраструктури через програму займає до 6 хвилин при стабільному Інтернет з’єднанні, що означає, що реалізована програма в 10 разів швидша за ручне розгортання аналогічної інфраструктури.</td></tr>
<tr><td rowspan="4">7. Стадії і етапи розробки</td><td>Дата</td><td>Назва етапу</td></tr>
<tr><td>21.10.23 – 25.10.23</td><td>Розробка та опис загальної моделі для об'єкту практики у форматі IDEF0</td></tr>
<tr><td>25.10.23 – 01.11.23</td><td>Розробка, опис та деталізація моделі для об'єкту практики у форматі різних UML діаграм.</td></tr>
<tr><td>01.11.23 – 04.11.23</td><td>Дослідження та опис програмного забезпечення для реалізації розробленої моделі.</td></tr>
<tr><td></td><td>04.11.23 – 15.11.23</td><td>Програмна реалізація</td></tr>
</table>

	15.11.23 – 20.11.23	розробленої моделі на основі підібраного програмного забезпечення. Тестування та корегування програмного продукту.
8. Порядок контролю і приймання	В даному розділі повинні бути вказанізагальні вимоги до приймання розробленого програмного виробу наприклад: 1) Перевірка ходу розробки програмного виробу. Керівнику робіт виконувати 1 раз в 3 тижні. 2) Випробування програмного виробу відповідно до програми і методики випробувань провести на базі комп'ютерного класу. 3) Захист розробленого програмного виробу провести на засіданні атестаційної комісії. Пояснювальну записку надати на паперових носіях в одному примірнику, в електронному вигляді - на CD-диску в одному екземплярі.	

Виконавець:

студент групи КУ-61

Мірошниченко Д.О.

Замовник:

д. т. н., професор кафедри теоретичної та прикладної системотехніки

Толстолузька О.Г

Додаток В

Програма і методика випробувань програмного виробу

«Модель процесу інтеграції системи управління контентом в платформу amazon web services»

1 Об'єкт випробувань

1.1 Найменування випробуваного програмного виробу: модель процесу інтеграції системи управління контентом в платформу amazon web services.

1.2 Область його застосування: : автоматизація та приладобудування.

2. Мета випробувань

Перевірка відповідності функціональності програмної реалізації заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1 Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2 Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі локального комп’ютерного забезпечення Виробника протягом заявлених стадій та етапів розробки (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3.3 Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4 Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться Виконавцем та Замовником.

4. Вимоги до програми або програмного виробу

- Сумісність з ОС: програма повинна працювати на основних операційних системах: Windows, Linux.
- Надійність: забезпечити високу ступінь надійності та стабільності роботи програми, мінімізувати можливість збоїв і помилок.
- Захист від некоректних дій користувача: включити заходи безпеки для запобігання некоректних дій користувачів, валідація даних і обробка помилок.
- Розширюваність: програма має бути легко розширюваною, дозволяючи додавати нові функції та модулі без значного переписування існуючого коду.
- Ізоляція елементів програми: компоненти програми повинні бути ізольовані одне від одного, щоб зменшити їх взаємний вплив на роботу програми при редагуванні коду.
- Маркування, упаковка, транспортування, і зберігання: ці вимоги не застосовуються до програмного продукту.

5. Вимоги до програмної документації

Програмною документацією до виробу «модель процесу інтеграції системи управління контентом в платформу amazon web services» вважати:

1) Технічне завдання на розробку програмного виробу (представлено в Додатку Б до пояснювальної записки до кваліфікаційної роботи).

2) Програма і методика випробувань розробленого програмного виробу (представлена в Додатку В до пояснювальної записки до кваліфікаційної роботи).

3) Опис програмного виробу (представлено в розділі 3 пояснювальної записки до кваліфікаційної роботи).

4) Текст програми (представлений в Додатку Г до пояснювальної записки до кваліфікаційної роботи) ».

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідно встановити програмне забезпечення Terraform, Docker-compose та Python включно бібліотекою python_terraform. Випробовування проводяться в середовищі Ubuntu WSL 20.04.5.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

3. Перевірку комплектності складу технічних і програмних засобів.

4. Методику проведення перевірок на 1 етапі випробувань.

5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1) Перевірку відповідності технічних характеристик програми вимогам технічного завдання.

2) Перевірку ступеня виконання функціональних вимог до програми.

3) Методику проведення перевірок, що входять до переліку по 2 етапу випробувань

1) Програма працює відповідно до умов експлуатації операційних систем MS Windows, Linux, а також сумісних з ними.

2) Для роботи необхідний компілятор мови програмування python, версії не нижчої ніж 3.0, встановлене програмне забезпечення Terraform, обліковий запис в Amazon Web Services та інтерфейс командного рядка AWS CLI.

3) Порядок проведення випробувань:

1. Запуск програми здійснюється за допомогою команди «python3 CreateWrapper.py» у середовищі Ubuntu WSL 20.04.5
2. Після запуску програми вводяться параметри для інфраструктури. Якщо параметр не вказано – використовується значення за замовчуванням.
3. В середовищі Ubuntu WSL після виконання буде виведено результат виконання програми та час, за який було розгорнуто інфраструктуру.
4. Для перевірки інфраструктури потрібно перейти в консоль AWS у веб браузері та переглянути сервіси EC2 та VPC
5. Потім потрібно перейти в браузері за IP адресою створеного сервера (можна дізнатися в консолі користувача AWS) за протоколом HTTP.
6. У вкладці браузера з’явиться конфігураційна сторінка WordPress, що свідчитиме про успішне розгортання CMS та бази даних.
7. Для підключення до бази даних потрібно використати MySQL клієнт, ввести IP сервера як хоста та дані користувача (ім’я та пароль).

Для проведення випробувань пропонується тест 1 та тест 2.

Тест 1

1. перевірка виконання програми;

2. введення змінних для розгортання інфраструктури;

3. отримання виводу програми про успішне виконання, перевірка веб браузера та підключення до бази даних.

```
aws_instance.web (remote-exec):  ✓ Container ec2-user-mysql-1      Started1.2s
aws_instance.web (remote-exec):  ✓ Container ec2-user-wordpress-1  Started1.2s
aws_instance.web (remote-exec):
aws_instance.web: Creation complete after 3m28s [id=i-0302fc23c2e8a82db]

Apply complete! Resources: 13 added, 0 changed, 0 destroyed.
Total time for Terraform execution: 296.27 seconds
```

Рис. В.1. – Тест 1: вивід успішного виконання програми.

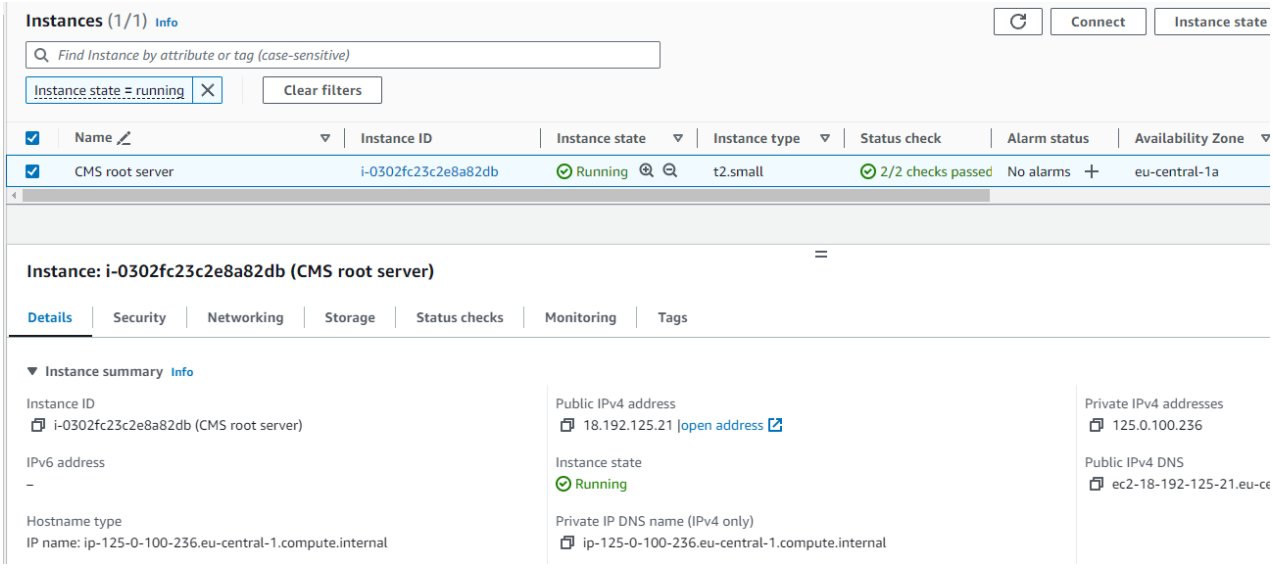

Рис. В.2. – Тест 1: інформація про створений сервер в AWS.

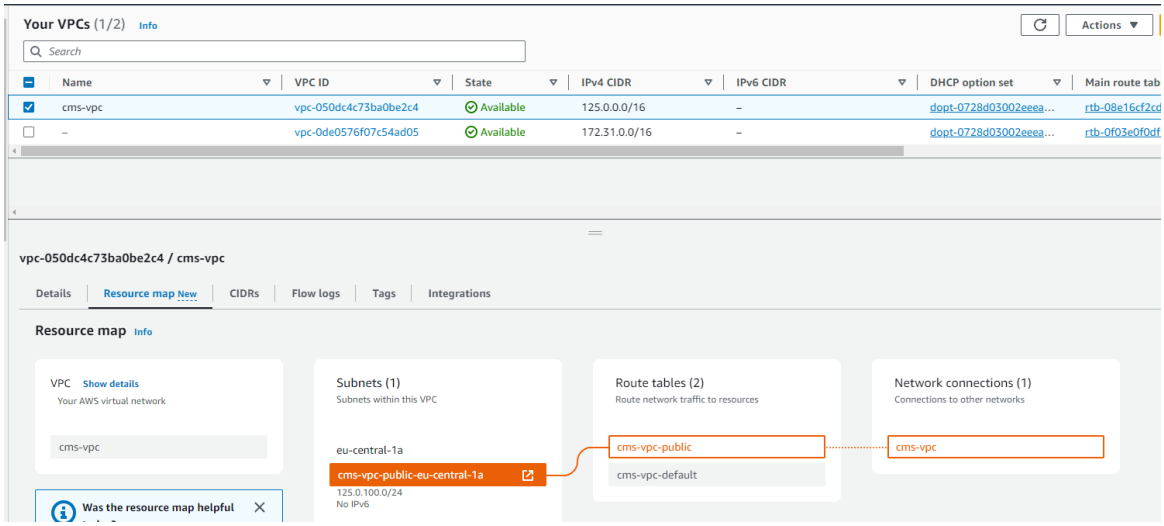

Рис. В.3. – Тест 1: інформація про створену мережу в AWS.

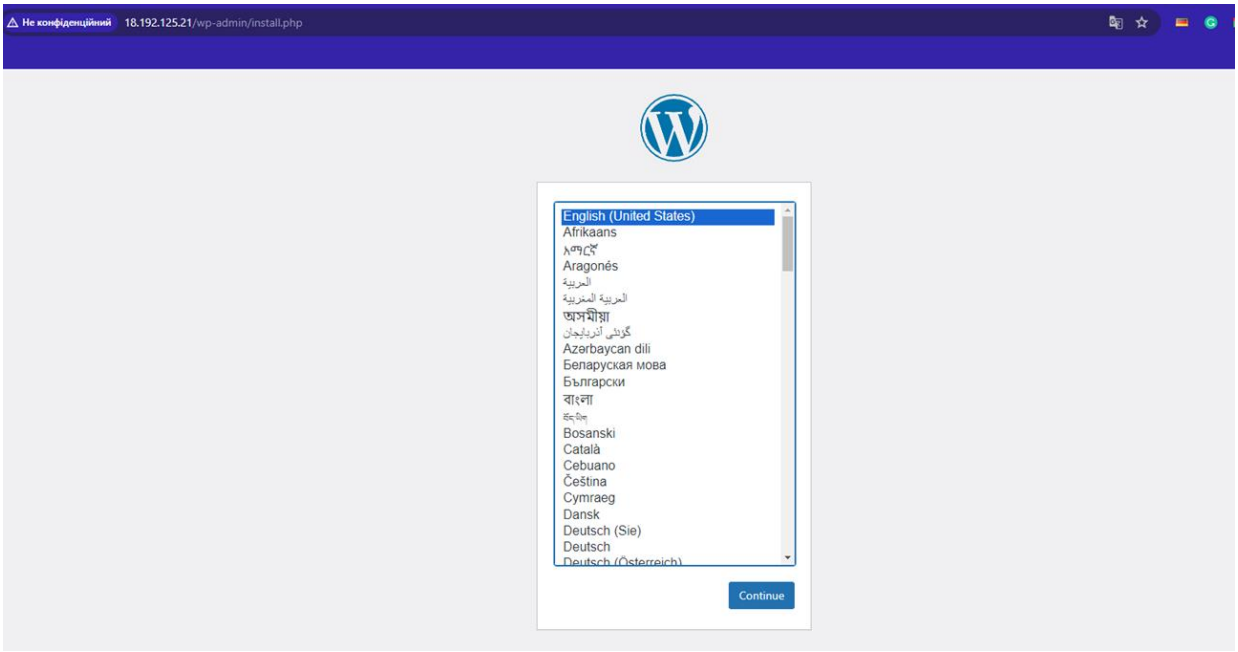

Рис. В.4. – Тест 1: веб-сторінка початкової конфігурації Wordpress.

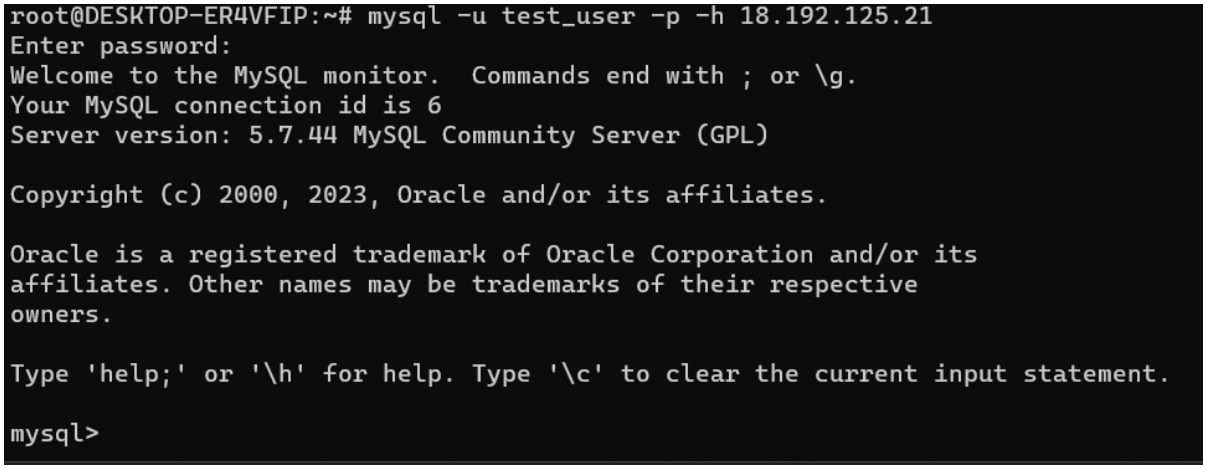

Рис. В.5. – Тест 1: успішне підключення до бази даних.

Тест 2

1. перевірка виконання програми;

2. введення даних, які можуть спровокувати помилку;

3. отримання виводу програми про неможливе виконання програми.

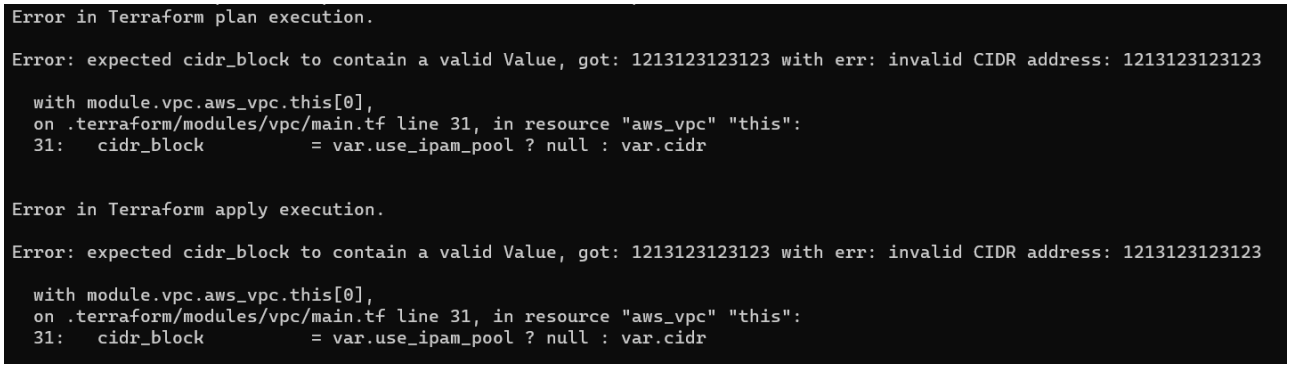

Рис. В.6. – Тест 2: приклад помилок при неуспішному виконанні програми.

Висновки: тест 1 успішно пройшов випробування і тест 2 успішно пройшов випробування. Випробування пройшло успішно.

Виконавець: студент групи КУ-61, Мірошниченко Д.О.

Додаток Г

ТЕКСТ ПРОГРАМИ

«Модель процесу інтеграції системи управління контентом в платформу amazon web services»

CreateWrapper.py

```python
#!/bin/python

import time
from python_terraform import Terraform

# Collect user inputs with default values
vpc_cidr = input('Enter VPC CIDR (press enter to use default - 10.0.0.0/16): ') or "10.0.0.0/16"
public_subnets_cidr = input('Enter public subnet CIDR separated by commas, if you need more than one (press enter to use default - 10.0.10.0/24):
sg_name = input('Enter security group name (press enter to use default - ec2_sg): ') or "ec2_sg"
ssh_connect_ips = input('Enter IPs range separated by commas to allow SSH connect to server (press enter to use default - 0.0.0.0/0): ') or "0.0.0
db_connect_ips = input('Enter IPs range separated by commas to allow database connect to server (press enter to use default - 0.0.0.0/0): ') or "0
ssh_key_name = input('Enter name for ssh key (press enter to use default - cms_server_ssh_key): ') or "cms_server_ssh_key"
instance_type = input('Enter server type (press enter to use default - t2.micro): ') or "t2.micro"
azs = input('Enter availability zone names separated by commas, if you need more than one (press enter to use default - eu-central-1a): ') or "eu-
wordpress_db_user = input('Enter WordPress DB user name (press enter to use default - db_user): ') or "db_user"
wordpress_db_password = input('Enter WordPress DB user password (press enter to use default - db_password): ') or "db_password"
wordpress_db_name = input('Enter WordPress DB name (press enter to use default - db_name): ') or "db_name"
mysql_root_password = input('Enter WordPress DB password (press enter to use default - password): ') or "password"

# Construct variables dictionary
variables = {
    'vpc_cidr': vpc_cidr,
    'public_subnets_cidr': public_subnets_cidr.split(','),
    'sg_name': sg_name,
    'ssh_connect_ips': ssh_connect_ips.split(','),
    'db_connect_ips': db_connect_ips.split(','),
    'ssh_key_name': ssh_key_name,
    'instance_type': instance_type,
    'azs': azs.split(','),
    'wordpress_db_user': wordpress_db_user,
    'wordpress_db_password': wordpress_db_password,
    'wordpress_db_name': wordpress_db_name,
    'mysql_root_password': mysql_root_password
}
tf = Terraform(working_dir='/mnt/c/Users/User/Desktop/Folders/6 курс/Диплом/infra', variables=variables)

# Terraform Plan
plan_output = tf.plan(refresh=True, capture_output=True)

if plan_output[0] == 2:  # Checking if the plan was successful
    formatted_plan_output = plan_output[1].strip().split('\n')  # Splitting the output into lines
    for line in formatted_plan_output:
        print(line)  # Printing each line individually for better readability
else:
    print("Error in Terraform plan execution.")
    print(plan_output[2])

start_time = time.time() #Start time of terraform execution

# Terraform Apply
apply_output = tf.apply(skip_plan=True, capture_output=True, auto_approve=True)

end_time = time.time()#Finish time of terraform execution

# Calculate total execution time
total_duration = end_time - start_time

if apply_output[0] == 0:  # Checking if the plan was successful
    formatted_apply_output = apply_output[1].strip().split('\n')  # Splitting the output into lines
    for line in formatted_apply_output:
        print(line)  # Printing each line individually for better readability
else:
    print("Error in Terraform apply execution.")
    print(apply_output[2])

print(f"Total time for Terraform execution: {total_duration:.2f} seconds")
```

Рис. Г.1. – файл CreateWrapper.py.

```
docker-compose.tpl
version: '3.8'

services:
  wordpress:
    image: wordpress:latest
    ports:
      - "80:80"
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_USER: ${wordpress_db_user}
      WORDPRESS_DB_PASSWORD: ${wordpress_db_password}
      WORDPRESS_DB_NAME: ${wordpress_db_name}
    volumes:
      - wordpress:/var/www/html

  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ${mysql_root_password}
      MYSQL_DATABASE: ${wordpress_db_name}
      MYSQL_USER: ${wordpress_db_user}
      MYSQL_PASSWORD: ${wordpress_db_password}
    volumes:
      - mysql:/var/lib/mysql

volumes:
  wordpress:
  mysql:
```

Рис. Г.2. – файл docker-compose.tpl.

```
data.tf > ...
# Data source for creating a Docker Compose file using a template
1 reference
data "template_file" "docker_compose" {
  # Specifies the template file path.
  # The file function reads the contents of a file, and path.module refers to the module's directory.
  template = file("${path.module}/docker-compose.tpl")

  # Variables to be substituted in the template file.
  vars = {
    wordpress_db_user     = var.wordpress_db_user     # Username for the WordPress database
    wordpress_db_password = var.wordpress_db_password # Password for the WordPress database
    wordpress_db_name     = var.wordpress_db_name     # Name of the WordPress database
    mysql_root_password   = var.mysql_root_password   # Root password for the MySQL database
  }
}

# Data source for querying the most recent Amazon Machine Image (AMI) that matches given criteria
1 reference
data "aws_ami" "this" {
  most_recent = true # Ensures the most recent AMI is selected

  # Filter to select AMIs based on their name.
  # Here, it selects Amazon Linux 2 AMIs that are HVM enabled.
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*"]
  }

  # Filter to ensure only available AMIs are considered.
  filter {
    name   = "state"
    values = ["available"]
  }

  owners = ["amazon"] # Restricts search to AMIs owned by Amazon
}
```

Рис. Г.3. – файл data.tf.

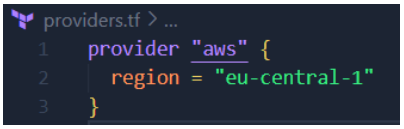

Рис. Г.4. – файл providers.tf.

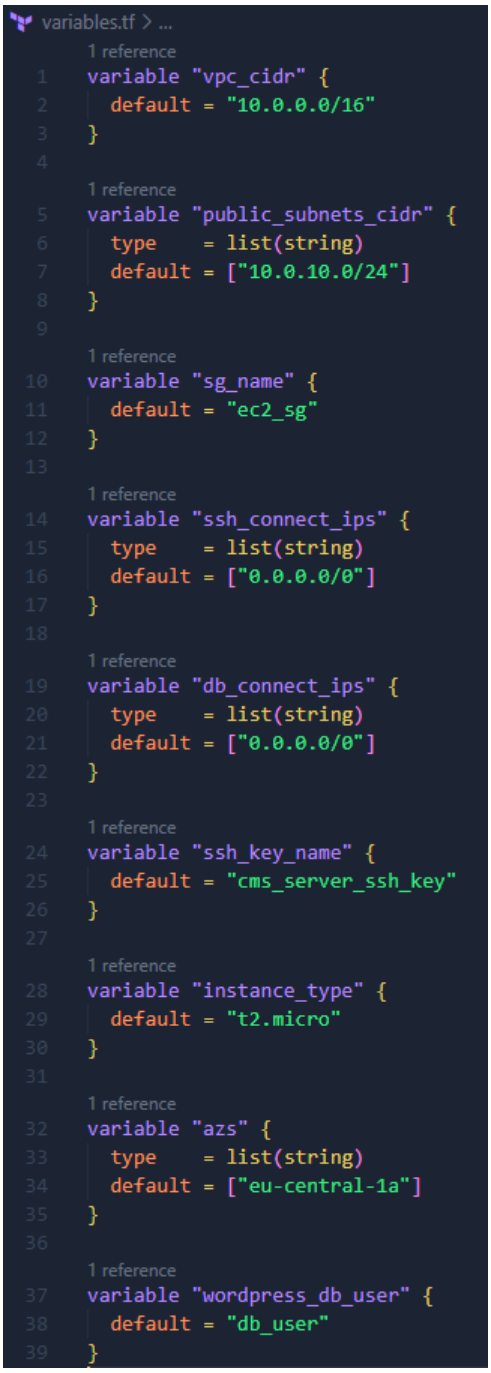

Рис. Г.5. - файл variables.tf.

main.tf > resource "aws_security_group" "ec2_sg"

```
# Module for creating a VPC (Virtual Private Cloud) in AWS
2 references
module "vpc" {
  source = "terraform-aws-modules/vpc/aws" # Specifies the Terraform registry source for the VPC module

  name = "cms-vpc"    # Name of the VPC
  cidr = var.vpc_cidr # CIDR block for the VPC, specified as a variable

  azs            = var.azs                # Availability zones for the VPC, specified as a variable
  public_subnets = var.public_subnets_cidr # CIDR blocks for the public subnets, specified as a variable

  tags = {
    Terraform = "true" # A tag to mark resources managed by Terraform
  }
}

# Resource for creating a security group in the above VPC
1 reference
resource "aws_security_group" "ec2_sg" {
  name        = var.sg_name             # Name of the security group, specified as a variable
  description = "Allow inbound traffic" # Description of the security group
  vpc_id      = module.vpc.vpc_id       # Associates the security group with the VPC created above

  # Ingress rules for allowing HTTP traffic
  ingress {
    from_port        = 80
    to_port          = 80
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"] # Allows traffic from all IP addresses
    ipv6_cidr_blocks = ["::/0"]      # Allows IPv6 traffic from all addresses
  }

  # Ingress rules for allowing HTTPS traffic
  ingress {
    from_port        = 443
    to_port          = 443
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"] # Allows traffic from all IP addresses
    ipv6_cidr_blocks = ["::/0"]      # Allows IPv6 traffic from all addresses
  }

  # Ingress rules for allowing SSH traffic
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = var.ssh_connect_ips # Allows SSH traffic from specified IP addresses
  }
```

Рис. Г.6. – файл main.tf, перший фрагмент.

```
    # Ingress rules for allowing Database connect
    ingress {
      from_port   = 3306
      to_port     = 3306
      protocol    = "tcp"
      cidr_blocks = var.db_connect_ips # Allows Database connect from specified IP addresses
    }
    # Egress rules to allow all outbound traffic
    egress {
      from_port        = 0
      to_port          = 0
      protocol         = "-1"          # Allows all protocols
      cidr_blocks      = ["0.0.0.0/0"] # Allows traffic to all IP addresses
      ipv6_cidr_blocks = ["::/0"]      # Allows IPv6 traffic to all addresses
    }

    tags = {
      Name = "allow_tls" # A tag for the security group
    }
  }

  # Resource for creating a local file for Docker Compose configuration
  1 reference
  resource "local_file" "docker_compose_file" {
    content  = data.template_file.docker_compose.rendered # Content of the Docker Compose file
    filename = "${path.module}/docker-compose.yaml"       # Location to save the Docker Compose file
  }

  # Resource for creating an AWS key pair for SSH access
  1 reference
  resource "aws_key_pair" "ssh_key" {
    key_name   = var.ssh_key_name              # Name of the key pair
    public_key = file("/root/.ssh/id_rsa.pub") # Public key for the SSH key pair
  }

  # Resource for creating an AWS EC2 instance
  1 reference
  resource "aws_instance" "web" {
    ami                         = data.aws_ami.this.id           # AMI for the instance
    instance_type               = var.instance_type             # Type of instance
    key_name                    = aws_key_pair.ssh_key.key_name  # SSH key pair name
    subnet_id                   = module.vpc.public_subnets[0]   # Subnet ID for the instance
    vpc_security_group_ids      = [aws_security_group.ec2_sg.id] # Security group for the instance
    associate_public_ip_address = true                           # Associate a public IP address

    tags = {
      Name = "CMS root server" # A tag for the instance
    }
```

```
  # Provisioner to transfer the Docker Compose file to the instance
  provisioner "file" {
    source      = local_file.docker_compose_file.filename
    destination = "/tmp/docker-compose.yaml"
  }

  # Provisioner to install and configure Docker and Docker Compose on the instance
  # Commands to install and start Docker, add user to Docker group, and install Docker Compose
  provisioner "remote-exec" {
    inline = [
      "set -e",
      "echo 'Installing Docker...'",
      "sudo yum install docker -y",
      "echo 'Starting Docker...'",
      "sudo systemctl start docker",
      "sudo systemctl enable docker",
      "echo 'Adding user to docker group...'",
      "sudo usermod -aG docker $USER",
      "echo 'Downloading Docker Compose...'",
      "sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose",
      "echo 'Setting permissions for Docker Compose...'",
      "sudo chmod +x /usr/local/bin/docker-compose"
    ]
  }

  # Provisioner to move Docker Compose file and start the Docker Compose application
  # Commands to move the Docker Compose file and start the application
  provisioner "remote-exec" {
    inline = [
      "echo 'Moving docker-compose.yaml...'",
      "sudo mv /tmp/docker-compose.yaml /home/ec2-user/docker-compose.yaml",
      "echo 'Starting Docker Compose...'",
      "cd /home/ec2-user",
      "docker-compose up -d"
    ]
  }

  # SSH connection configuration for the provisioners
  connection {
    type        = "ssh"
    user        = "ec2-user"
    private_key = file("/root/.ssh/id_rsa") # Update with your SSH key path
    host        = self.public_ip
  }
}
```

Рис. Г.7. – файл main.tf, другий та третій фрагмент.