Харківський національний університет радіоелектроніки
(повне найменування вищого навчального закладу)

Факультет *інфокомунікацій*
(повна назва)

Кафедра *інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*

на тему *Дослідження ефективності системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача*

Виконав:
студент *2* курсу, групи *ІМІм-22-3*
Федорченко О. М.
(прізвище, ініціали)
Спеціальність
172 "Телекомунікації та радіотехніка"
(код і повна назва спеціальності)
Освітня програма
Інформаційно-мережна інженерія
(повна назва освітньої програми)
Керівник *доц. Костромицький А.І.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ *Безрук В.М.*
(підпис) (прізвище, ініціали)

Харків - 2024 рік

Не містить відомостей, заборонених до відкритого публікування

Студент _____

Керівник_____

Харківський національний університет радіоелектроніки
(повне найменування вищого навчального закладу)

Факультет *інфокомунікацій*

Кафедра *інформаційно-мережної інженерії*

Рівень вищої освіти *другий (магістерський)*

Спеціальність *172 «Телекомунікації і радіотехніка»*

Тип програми *освітньо-наукова*

Освітня програма *Інформаційно-мережна інженерія*

ЗАТВЕРДЖУЮ

Зав.кафедри _____
(Підпис)

“ *18* ” *березня* 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Федорченко Олександру Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Дослідження ефективності системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача*

затверджені наказом ВНЗ від “*18* ” *березня* 2024 року № *232 Ст.*

2. Строк подання студентом роботи *12 червня 2024 року*

3. Вихідні дані до роботи *Дослідити ефективність системи перевірки виконання практичного завдання в локальному віртуальному оточенні користувача. Вдосконалити архітектуру системи. Апробація досліджень – підготувати та опублікувати тези доповіді на науковій конференції*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Вступ

1.Загальний огляд

2. Поточна архітектура

3. Шляхи вдосконалення архітектури

4. Практична реалізація поліпшень

5. Дослідження ефективності системи

Висновки

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

Слайди у форматі Power Point (назва та мета роботи, актуальність, дослідження архітектури, вдосконалення, висновки)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	*доц. Костромицький А.І.*	*18.03.24*	

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ.*	*18.03.24*	
2	*Підбір літератури за темою роботи.*	*19.03-23.03.24*	
3	*Виконання розділу 1*	*24.03-14.04.24*	
4	*Виконання розділу 2*	*15.04-24.04.24*	
5	*Виконання розділу 3*	*25.04-05.05.24*	
6	*Виконання розділу 4*	*06.05-17.05.24*	
7	*Виконання розділу 5*	*18.05-01.06.24*	
8	*Оформлення презентаційного матеріалу,*	*02.06-12.06.24*	
	підготовка до захисту у ЕК		

Дата видачі завдання *18 березня 2024 р.*

Студент _____ *Федорченко О. М.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Костромицький А.І.*
(підпис) (прізвище та ініціали

РЕФЕРАТ

Пояснювальна записка: 49 с., 13 рис., 6 джерел

Об'єкт роботи – система автоматичної перевірки виконання практичних завдань в локальному серидовищі учня.

Мета роботи – дослідження ефективності архітектури системи для автоматичної перевірки виконання практичних завдань, виконаних студентом в локальному серидовищі.

Досліджена поточна архітектура рішення та запропоновані та впроваджені поліпшення. Вдосконалення архітектури позитивно сприяло на якість та швидкість доставки програмного коду серверного додатка.

PYTHON, MOODLE, LMS, LTI, СИСТЕМА АВТОМАТИЧНОЇ ПЕРЕВІРКИ, CICD, TERRAFORM, ANSIBLE

ABSTRACT

Explanatory note: 49 p., 13 fig., 6 sources.

The object of the work is a system for automatically checking the completion of practical tasks in a student's local environment.

Purpose - to study the effectiveness of the system architecture for automatic verification of practical tasks performed by a student in a local serialization.

The current architecture of the solution was studied and improvements were proposed and implemented. Improvements in the architecture had a positive impact on the quality and speed of delivery of the server application code.

PYTHON, GOLANG, MOODLE, LMS, LTI, AUTOMATIC TESTING SYSTEM, CICD, TERRAFORM, ANSIBLE

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

LMS (Learning Management System) – система керування навчанням;

LTI (Learning Tools Interoperability) – специфікація освітніх технологій;

API (Application Programming Interface) – прикладний програмний інтерфейс;

HTTP (Hyper Text Transfer Protocol) – протокол передачі даних;

HTTPS (Hyper Text Transfer Protocol Secure) – захищений протокол передачі даних;

VPN (Virtual Private Network) – віртуальна приватна мережа;

IaC (Infrastructure as Code) – інфраструктура як код;

AWS (Amazon Web Services) – провайдер хмарних рішень;

YAML (YAML Ain't Markup Language) – формат серіалізації даних;

CICD (Continuous Integration and Continuous Delivery/Deployment) – неперервна інтеграція та неперервена доставка/розгортання;

OIDC (OpenID Connect) – протокол аутентифікації користувача сторонніми сервісами;

PEP 8 (Python Enchancement Proposal 8) – настанова щодо стилю оформлення коду;

EC2 (Elastic Compute Cloud) – сервіс обчислювальних потужностей в хмарі;

VPC (Virtual Private Cloud) – віртуальна приватна хмара;

ELB (Elastic Load Balancing) – сервіс розподілення трафіка у хмарі;

SSH (Secure Shell) – мережевий протоколо віддаленого доступа;

ВСТУП

Мета кваліфікаційної роботи – дослідити ефективність системи перевірки виконання практичного завдання в локальному віртуальному оточенні користувача. Вдосконалити архітектуру системи.

В теперішній час освіта зазнала значної цифрової трансформації, яка прискорилася завдяки глобальному переходу до дистанційного навчання. Ця трансформація полягає не лише в перенесенні традиційного навчального контенту в онлайн, але й у переосмисленні того, як ми викладаємо, вчимося та оцінюємо навички в цифрову епоху. Хоча системи управління навчанням (LMS) відіграють ключову роль у сприянні цьому переходу, пропонуючи платформу для доставки контенту і теоретичного тестування, вони не вирішують одну важливу задачу: ефективне оцінювання практичних навичок. У дисциплінах, де практичний досвід має вирішальне значення, традиційні LMS намагаються забезпечити точне і змістовне оцінювання здібностей учнів.

Суть проблеми полягає у складності оцінювання практичних завдань дистанційно. Традиційні платформи LMS призначені для управління та надання контенту, проведення дискусій та адміністрування вікторин. Однак їм бракує механізмів для ефективного оцінювання практичних навичок, які необхідні в багатьох галузях.

Ця прогалина в можливостях стає все більш очевидною, оскільки освіта продовжує переходити в онлайн, що підкреслює потребу в інструментах, які можуть точно і ефективно оцінювати ці навички у віддаленому середовищі. Щоб заповнити цю прогалину, в роботі представлено вдосконалення прототипу системи, призначеної для автоматизації перевірки практичних завдань, що виконуються в локальному віртуальному середовищі користувача. Перевага такої системи в тому що завдяки використанню локального оточення, значно скорочуються експлуатаційні витрати на підтримку роботи системи. Така система легко інтегрується з існуючими LMS через протокол

Learning Tools Interoperability (LTI), пропонуючи масштабоване та інтегроване рішення для підвищення ефективності та доступності дистанційного навчання.

1 ЗАГАЛЬНИЙ ОГЛЯД РІШЕННЯ НА БАЗІ LMS ТА LTI

1.1 Опис LMS

Система управління навчанням (LMS) – це потужний програмний ресурс для навчання та професійного розвитку, який навчальні установи будь-якого направлення можуть включити в навчальні поцеси. Багато платформ LMS розміщуються в хмарі, що дозволяє отримати до них віддалений доступ. LMS поєднує в собі управління базами даних у цифровому середовищі для управління навчальними програмами, навчальними матеріалами, інструментами оцінювання, плануванням курсів, дотриманням нормативних вимог тощо. Програмне забезпечення LMS може допомогти контролювати навчання, розвиток і підготовку, дозволяючи встановити чіткі очікування для учасників навчального процесу щодо їхнього прогресу.

LMS – це програмне забезпечення, яке допомагає створювати, керувати, організовувати та надавати учням навчальні матеріали в режимі онлайн. Учнями можуть бути школярі, учні професійно-технічних навчальних закладів, студенти коледжів, університетів або працівники організацій будь-якого типу. Тобто будь-хто, хто зацікавлений у навчанні впродовж життя та доступі до навчальних матеріалів на пристрої, переважно через Інтернет..

Звичайне розгортання LMS складається з чотирьох типів компонентів: програмних додатків і сервісів, апаратного забезпечення, що складається з серверних комп'ютерів і сховищ даних, мережевих підключень до Інтернету та інших організаційних систем, а також користувацьких інтерфейсів для інструкторів, учасників та адміністраторів.

Рисунок 1.1 – Загальна діаграма LMS

1.2 Базові функції LMS

Skillbuilder (2019) перелічує десять базових функцій, які мають бути присутніми в будь-якому LMS і можуть бути основою для порівняння продуктів:

1) Простота використання – хороший інтерфейс LMS є інтуїтивно зрозумілим і зручним для користувача, ким би він не був. Він має бути швидким у вивченні. Зрештою, навряд чи окремі особи та заклади хочуть витрачати свій час, енергію та ресурси на серію курсів про те, як пройти курс, тому легкість використання – це обов'язкова функція LMS для всіх;

2) Інтеграція – сумісність з іншими функціями та іншим програмним забезпеченням зменшує складність та час на випробування, оскільки інформацією з LMS можна ділитися з різними гілками навчального процесу;

3) Управління контентом – як частина LMS, ця функція надає авторам засоби для управління контентом і спільної роботи в одному централізованому місці. Це також гарантує відповідність LMS стандартам електронного навчання та сумісність із сучасними веб-технологіями. Окрім власного контенту, бажано, щоб LMS приймала курси від сторонніх постачальників;

4) Підтримка мобільного навчання – з подальшим розвитком мобільних технологій люди стають більш невіддільними від своїх мобільних телефонів, ніж від персональних комп'ютерів. У всьому світі 16-24-річні люди підключаються до інтернету через мобільні телефони загалом на 4,1 години на день порівняно з 3,3 годинами через ПК, а 25-34-річні користуються інтернетом через телефон (3,45 години) трохи більше, ніж через ПК (3,37) (Clement, 2019). Враховуючи, що сучасне покоління працівників отримує доступ до онлайн-контенту з мобільних телефонів більше, ніж з ПК, модулі LMS повинні мати мобільні версії, інакше вони ризикують бути проігнорованими;

5) Підтримка змішаного навчання – вміст LMS має підтримувати змішані методи навчання, такі як відео та аудіо, оскільки змішане навчання може забезпечити кращі результати порівняно з одним способом навчання;

6) Тестування та оцінювання – вбудоване тестування та оцінювання дозволяє контролювати цілі, завдання та результати онлайн-навчання, які є найбільш важливими для навчальних закладів. Результати можна збирати, оцінювати і порівнювати з іншими учнями або з ключовими показниками ефективності;

7) Звітування та відстеження – звіти на макрорівні дозволяють оцінити ефективність контенту. Вони включають дані про кількість користувачів, курсів, груп у навчанні, відсоток завершення курсів і успішність проходження тестів, серед іншого. Рішення для відстеження відстежують прогрес розвитку кожного учасника протягом усього курсу;

8) Безпека – має першорядне значення, оскільки LMS містить не лише персональні дані учнів, а й власне навчальні матеріали. Це включає в себе парольну автентифікацію, блокування IP-адрес та шифрування;

9) Кастомізація та брендування – налаштування зовнішнього вигляду контенту для відображення образів бренду є бажаною опцією для багатьох клієнтів. Дозвіл клієнтам використовувати назву свого бренду в навчальних модулях допомагає користувачам ідентифікувати себе ближче до бренду;

10) Електронна комерція – функції електронної комерції, що використовуються для оплати, дозволяють клієнтам купувати програмне забезпечення через Інтернет, використовуючи безпечні платіжні засоби.

1.3 Інтеграція LMS та LTI

Інтероперабельність навчальних інструментів (Learning Tools Interoperability (LTI) - це стандарт, розроблений Глобальним навчальним консорціумом IMS, який дозволяє різним навчальним програмам безперешкодно підключатися та взаємодіяти. Він функціонує як міст між системами управління навчанням (LMS), освітніми платформами та іншими зовнішніми інструментами, сприяючи їхній сумісності та обміну інформацією в безпечний, стандартизований спосіб.

Інтеграція LTI означає процес з'єднання та забезпечення зв'язку між різними додатками та платформами електронного навчання у стандартизований спосіб.

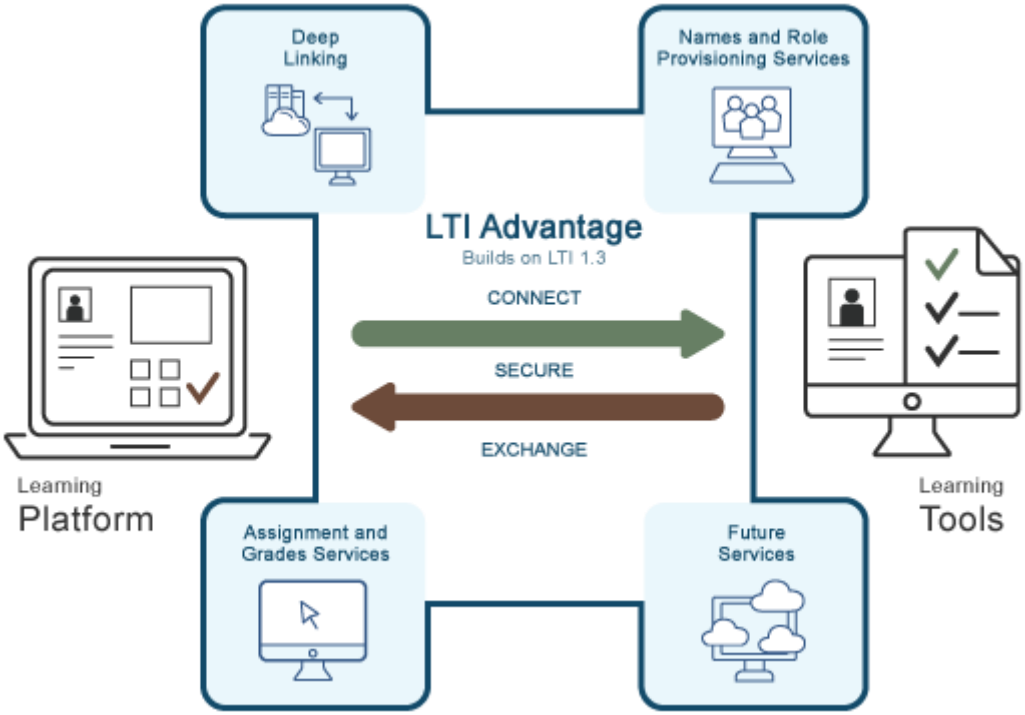

Рисунок 1.3 – Інтеграція LTI

У той же час інтеграція дозволяє системі управління навчанням (LMS) запускати зовнішні інструменти та сервіси таким чином, щоб забезпечити цим інструментам безпечний доступ до заздалегідь визначеного набору даних користувача. Ця інтеграція дозволяє студентам і викладачам взаємодіяти з різноманітними навчальними інструментами з основної LMS без необхідності окремого входу для кожного інструменту.

На практиці LTI може виглядати так: викладач має бажання інтегрувати зовнішню платформу для відеолекцій у свою LMS. Замість того, щоб вимагати від студентів виходити з LMS і окремо входити на цю платформу, LTI дозволяє інтегрувати цей зовнішній інструмент безпосередньо в LMS. Студенти можуть отримувати доступ до контенту відеолекцій і взаємодіяти з ним, не виходячи з середовища LMS, а їхню взаємодію з матеріалом можна відстежувати і передавати в LMS. Це створює безперебійний, уніфікований навчальний процес, підвищуючи зацікавленість і зменшуючи потенційні технічні труднощі.

Інтероперабельність засобів навчання (LTI) – важливий компонент електронного навчання, що забезпечує покращену інтеграцію, обмін даними, зручність для користувачів, можливості для навчання та економічну ефективність. Руйнуючи бар'єри між різними інструментами та платформами електронного навчання, інтероперабельність відіграє важливу роль у створенні безперешкодного, цікавого та ефективного досвіду онлайн-навчання. Оскільки сектор електронного навчання продовжує розвиватися і розширюватися, розуміння ключових концепцій, таких як LTI, стає все більш важливим для викладачів, адміністраторів і професіоналів у галузі електронного навчання. Завдяки своєму потенціалу трансформувати та покращувати цифрове навчальне середовище, LTI продовжуватиме відігравати центральну роль у майбутньому електронного навчання.

2 ПОТОЧНА АРХІТЕКТУРА ДОДАТКУ

2.1 Використання змішаної архітектури для додатку

Проєкт базується на використанні відкритої системи управління навчанням (LMS), яка вже є повноцінним і готовим до застосування продуктом, а саме Moodle. Це готове рішення дозволяє заощадити час та ресурси. На схемі (рис. 2.1) видно основні компоненти проєкту.

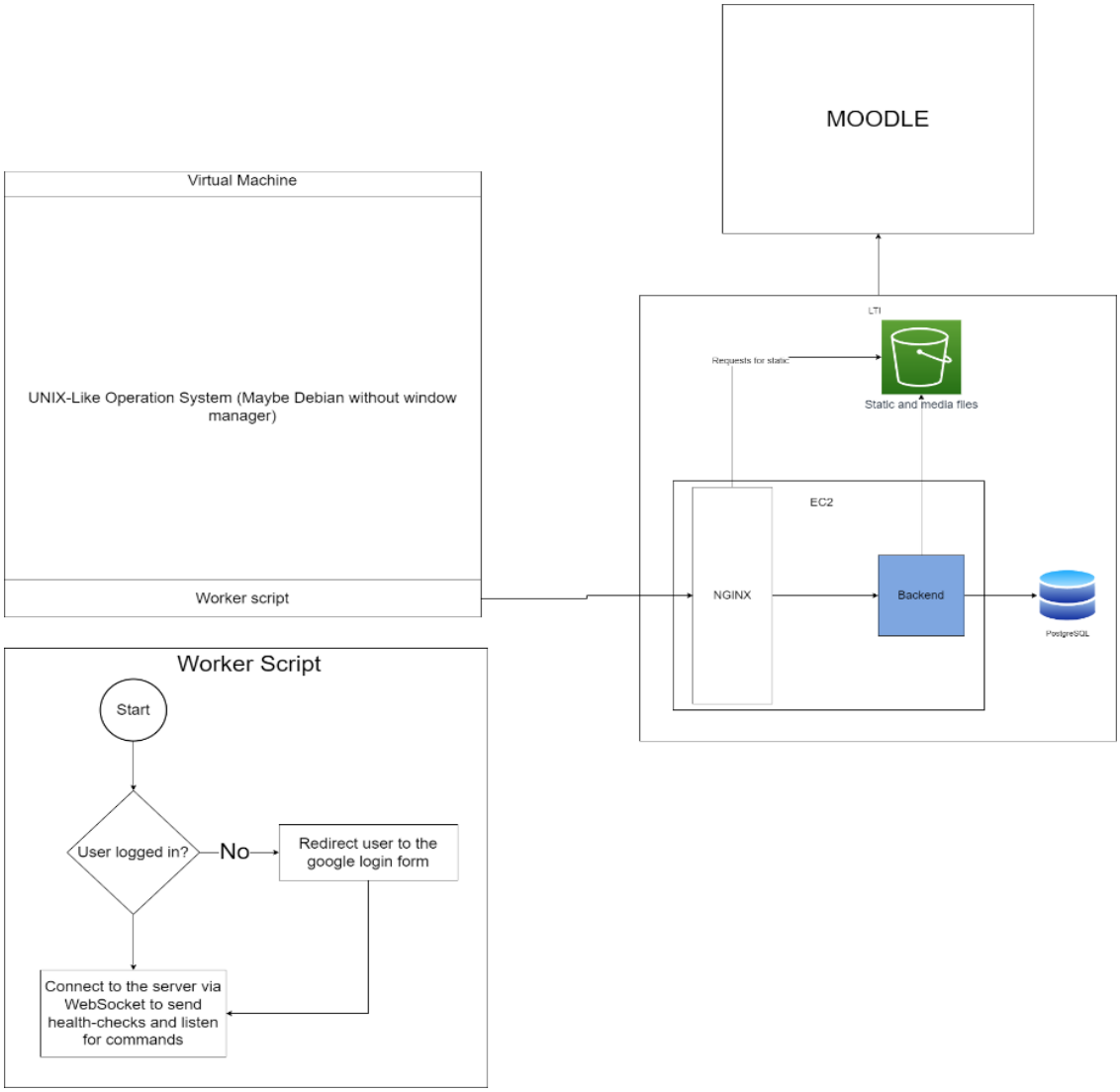

Рисунок 2.1 – Архітектура проєкту

Перший компонент системи – це монолітна LMS, Moodle. Ця система відповідає за створення завдань, розміщення лекційних матеріалів та надає широкий спектр функцій, властивих системам управління навчанням.

Другий компонент системи - це додаток, який буде виконувати роль автоматизованого перевіряльника завдань. Цей компонент функціонує як окремий сервіс, оскільки інтеграція такого механізму в існуючу систему з великою кодовою базою є складним завданням.

Третій компонент – це клієнтське програмне забезпечення. Це локальне середовище учня, на якому розгорнуто віртуальну машину зі спеціально написаним скриптом, що забезпечує синхронізацію між локальним середовищем та сервісом перевірки домашніх завдань.

Четвертий компонент є продовженням третього. Це скрипт, який відповідає за функціонал синхронізації між локальним середовищем учня та сервісом перевірки.

2.2 Опис серверної частини додатку

Сервер працює на стабільній та рекомендовані системі Debian Linux. На цьому сервері працює програма, яка обробляє всі запити від користувачів. У ролі користувачів виступають веб-браузер учня та його віртуальна машина (worker). Клієнти спілкуються з сервером за допомогою вебсокетів (WebSocket), що дозволяє підтримувати постійне двостороннє з'єднання.

Для взаємодії клієнтів і сервера необхідно розробити прикладний програмний інтерфейс (API), що є набором чітко визначених методів для взаємодії між сервісом і програмою, яка його використовує.

Сервер відкриває порт для вхідних підключень через HTTP, що є прийнятним під час розробки в локальному середовищі, але небезпечним для взаємодії з користувачем через глобальну мережу, оскільки може призвести до витоку даних. Тому з'єднання далі буде захищене за допомогою HTTPS.

Для цього було налаштовано окремий публічний сервер з Nginx у ролі зворотного проксі-сервера. Це дозволило швидко налаштувати HTTPS для нашого сервера і перенести його в захищену мережу VPN. Крім того, у майбутньому це дасть змогу підвищити надійність системи, налаштувавши обмеження для відкритих з'єднань з однієї IP-адреси та використовуючи балансувальник навантаження (LoadBalancer) для розподілу навантаження.

3 ШЛЯХИ ВДОСКОНАЛЕННЯ АРХІТЕКТУРИ

3.1 Опис інфраструктури як коду. Переваги та недоліки

Інфраструктура як Код (Infrastructure as Code, IaC) – це практика управління та налаштування інфраструктури через машинозчитувані файли, а не фізичні апаратні налаштування чи інтерактивні налаштування конфігурації. Основна ідея полягає в тому, що інфраструктура стає кодом, який можна версіонувати, перевіряти, перевикористовувати та автоматизувати.

Одним із найпоширеніших інструментів для реалізації IaC є Terraform, створений компанією HashiCorp. Terraform дозволяє користувачам визначати інфраструктуру у вигляді конфігураційних файлів, які можна застосувати для створення та керування інфраструктурою у різних хмарних сервісах, таких як AWS, Azure, Google Cloud Platform та інших.

Переваги Інфраструктури як Коду:

1) Автоматизація та повторюваність – замість того, щоб вручну налаштовувати інфраструктуру кожного разу, IaC дозволяє автоматизувати ці процеси; це забезпечує стабільність і повторюваність, оскільки один і той самий код створює однакове середовище кожного разу, з Terraform можна автоматично створити віртуальні машини, налаштувати мережеві конфігурації та розгорнути додатки з однієї конфігурації;

2) Швидке розгортання – IaC дозволяє швидко розгорнути складні середовища за короткий час, це особливо корисно для великих організацій, де потрібно швидко масштабувати ресурси; З Terraform можна розгортати цілі середовища натисканням однієї кнопки або за допомогою однієї команди;

3) Контроль версій та історія змін – використання систем контролю версій, таких як Git, дозволяє відстежувати всі зміни в конфігурації інфраструктури, це забезпечує повний аудит і дозволяє легко повертатися до

попередніх версій у разі помилок; Конфігураційні файли Terraform можуть зберігатися в репозиторії Git, що дозволяє відстежувати всі зміни;

4) Консистентність середовищ – IaC забезпечує, що різні середовища (розробка, тестування, продуктивність) мають однакову конфігурацію, що знижує ризик непередбачуваних помилок; використовуючи один і той самий Terraform код, можна створити однакові середовища для різних етапів розробки.

Недоліки Інфраструктури як Коду:

1) Крива навчання – впровадження IaC вимагає певного рівня знань і досвіду, що може бути складним для новачків, вивчення нових інструментів і підходів потребує часу; terraform має свою специфічну синтаксис і логіку, яку потрібно вивчити перед використанням

2) Керування станом – інструменти IaC, такі як Terraform, зберігають стан інфраструктури, що може викликати проблеми, якщо стан не синхронізований між різними користувачами або середовищами; неправильне управління файлом стану Terraform може призвести до непередбачуваних змін в інфраструктурі;

3) Залежність від інструментів – використання конкретного інструмента для IaC може створити залежність від цього інструмента, якщо інструмент змінюється або припиняє підтримку, це може створити проблеми;

4) Складність управління великими інфраструктурами – керування дуже великою інфраструктурою через код може стати складним і заплутаним, потрібно ретельно планувати та організовувати конфігураційні файли.

У підсумку, Інфраструктура як Код є потужним підходом до управління інфраструктурою, забезпечуючи автоматизацію, повторюваність і консистентність. Використання інструментів, таких як Terraform, може значно покращити ефективність і надійність управління інфраструктурою. Однак, впровадження IaC вимагає часу для навчання та адаптації, а також ретельного управління станом і організацією конфігураційних файлів.

3.2 Опис конфігурації як коду. Переваги та недоліки

Конфігурація як Код (Configuration as Code, CaC) - це практика управління та налаштування систем і програмного забезпечення через кодовані файли, а не за допомогою ручного налаштування. Цей підхід дозволяє автоматизувати налаштування серверів, додатків та інших компонентів інфраструктури, забезпечуючи консистентність та повторюваність конфігурацій.

Одним із найпоширеніших інструментів для реалізації CaC є Ansible, створений компанією Red Hat. Ansible дозволяє описувати конфігурації у вигляді сценаріїв (playbooks) на основі YAML, які можна використовувати для автоматизації налаштувань серверів, розгортання додатків і оркестрації складних робочих процесів.

Переваги Конфігурації як Коду:

1) Автоматизація та повторюваність – Ansible дозволяє автоматизувати процеси налаштування та конфігурації, зменшуючи потребу в ручних операціях. Це забезпечує консистентність та повторюваність конфігурацій у всіх середовищах, наприклад, за допомогою Ansible можна автоматизувати налаштування веб-серверів, баз даних та інших сервісів, використовуючи один і той самий сценарій;

2) Швидкість та ефективність – автоматизація процесів конфігурації дозволяє значно зменшити час, необхідний для розгортання нових серверів або налаштування програмного забезпечення, це особливо корисно в середовищах з великою кількістю серверів, Ansible може одночасно налаштовувати сотні серверів, що значно прискорює процес розгортання;

3) Простота використання – Ansible використовує простий і зрозумілий синтаксис на основі YAML, що робить його легким для вивчення та використання, це знижує бар’єр входу для нових користувачів, наприклад,

конфігураційний файл Ansible виглядає як звичайний текстовий документ, що описує послідовність дій для налаштування сервера;

4) Ідемпотентність – один із ключових принципів Ansible означає, що повторне виконання одного й того ж сценарію не змінює стан системи, якщо він вже відповідає бажаному, це забезпечує стабільність і передбачуваність конфігурацій, Ansible перевіряє поточний стан системи перед внесенням змін, що запобігає небажаним модифікаціям.

Недоліки Конфігурації як Коду:

1) Масштабованість – Ansible підходить для автоматизації налаштувань у невеликих та середніх масштабах, але може стикатися з проблемами масштабованості у великих середовищах з тисячами серверів, це може вимагати додаткових інструментів для управління масштабованістю;

2) Управління станом – Ansible не зберігає стан інфраструктури, як це роблять деякі інші інструменти, це означає, що користувачам потрібно ретельно планувати сценарії, щоб забезпечити бажаний стан системи, відсутність централізованого збереження стану може ускладнити відстеження змін у великих проектах;

3) Відсутність зворотнього зв’язку у реальному часі – Ansible виконує завдання у послідовному режимі і може не надавати зворотний зв’язок у реальному часі, це може ускладнити налагодження і моніторинг у великих проектах, для отримання зворотного зв’язку в реальному часі можуть знадобитися додаткові інструменти моніторингу;

4) Крива навчання – хоча Ansible має відносно простий синтаксис, освоєння всіх можливостей та найкращих практик може вимагати часу і досвіду, це може бути складним для новачків, повне розуміння ідемпотентності та інших концепцій Ansible може потребувати додаткових зусиль.

В якості висновку можна зазначити, що Конфігурація як Код є потужним підходом до управління конфігураціями, який забезпечує автоматизацію,

повторюваність та стабільність. Використання інструментів, таких як Ansible, може значно покращити ефективність та надійність управління конфігураціями серверів та додатків. Однак, впровадження CaC вимагає часу для навчання та адаптації, а також ретельного планування та організації конфігураційних сценаріїв.

3.3 Інструменти поліпшення якості та безпеки програмного коду

Bandit і Flake8 – це популярні інструменти для аналізу коду на мові Python, які допомагають розробникам забезпечити високу якість та безпеку свого коду. Ці інструменти автоматично перевіряють код на відповідність певним стандартам і виявляють потенційні проблеми, що дозволяє розробникам швидко виправляти помилки та покращувати кодову базу.

Bandit - це інструмент для статичного аналізу коду, розроблений спеціально для виявлення вразливостей у Python-коді. Він аналізує файли вихідного коду і шукає загальні проблеми безпеки, такі як неправильне використання функцій або потенційні місця для атак.

До переваг Bandit можна віднести:

1) Виявлення вразливостей – Bandit спеціалізується на пошуку вразливостей безпеки, таких як SQL-ін’єкції, незахищені конфігурації та інші поширені проблеми, це допомагає запобігти експлуатації цих вразливостей у виробничому середовищі, наприклад, Bandit може виявити використання небезпечних методів або бібліотек, які можуть призвести до витоку даних;

2) Автоматизація безпеки – використання Bandit у процесі CI/CD дозволяє автоматично перевіряти кожен коміт на наявність вразливостей, що забезпечує безперервний контроль безпеки, це значно знижує ризик впровадження небезпечного коду у виробниче середовище;

3) Простота використання – Bandit легко інтегрується у процес розробки і має простий інтерфейс командного рядка. Він може аналізувати кодові бази різного розміру без значних налаштувань

Flake8 – це інструмент для статичного аналізу коду на Python, який поєднує функціональність кількох інших інструментів, таких як PyFlakes, pycodestyle та Ned Batchelder's McCabe script. Flake8 перевіряє код на відповідність стандартам стилю (PEP 8), виявляє синтаксичні помилки і складність коду.

Перевагі Flake8 наступні:

1) Відповідність стандартам стилю – Flake8 допомагає забезпечити, щоб код відповідав стилістичним стандартам, таким як PEP 8, це робить код більш читабельним і легшим для підтримки, як приклад можна виявити зайві пробіли, неправильне використання відступів та інші стилістичні помилки;

2) Виявлення помилок – виявляє синтаксичні та логічні помилки, які можуть приводить до несправностей у роботі програм, також можна виявити невикористовувані змінні, неправильне імпортування бібліотек та інші потенційні проблеми;

3) Аналіз складності коду – Flake8 включає перевірку складності коду за допомогою McCabe script, що дозволяє виявляти надмірно складні функції та методи, що допомагає розробникам оптимізувати та спрощувати код.

3.4 Вдосконалення системи доставки коду

Як система зберігання та доставки коду була використана Github та Github Actions з релізами додатку за допомогою тегів.

Теги у GitHub є зручним механізмом для маркування певних точок в історії репозиторію. Вони часто використовуються для позначення релізів програмного забезпечення, що дозволяє легко відстежувати різні версії додатка. Використання тегів у комбінації з автоматизованою доставкою

додатку при створенні тегу, має численні переваги для ефективного управління розгортанням програмного забезпечення.

Переваги використання тегів:

1) Чітка ідентифікація версій – теги дозволяють точно ідентифікувати конкретні версії програмного забезпечення, це особливо важливо для відстеження релізів, що спрощує підтримку та управління різними версіями додатка, наприклад, тег v1.0.0 може відповідати першій стабільній версії додатка, що дозволяє легко відновити цю версію у разі потреби;

2) Історія релізів – використання тегів створює чітку історію релізів, яка легко доступна всім учасникам проекту. Це допомагає у відстеженні змін між різними версіями та у визначенні, коли і які нові функції були додані, можна переглядати історію тегів у вкладці “Releases” на GitHub, що надає зручний інтерфейс для користувачів;

3) Зручність для користувачів – теги дозволяють користувачам легко знайти і завантажити певні версії програмного забезпечення, це зручно для користувачів, які хочуть використовувати стабільні релізи або повернутися до попередньої версії, користувачі можуть завантажувати певні версії додатка безпосередньо з репозиторію на GitHub.

Деплой додатку запропоновано змінити на тільки при створенні тегу, що дозволило:

1) Підвищити контроль якості – деплой додатку лише при створенні тегу, а не при кожному пуші у мастер гілку, забезпечує додатковий рівень контролю якості, що надає додаткову впевнитись, що тільки перевірені та готові до продуктивного середовища версії додатку потрапляють на сервер і це зменшує ризик впровадження сирого або некоректного коду у продуктивне середовище;

2) Сформувати чіткий процес релізів – використання тегів для деплою встановлює чіткий процес релізів, де кожен реліз повинен бути явним і маркованим, це забезпечує прозорість та передбачуваність у розгортанні нових версій, наприклад, новий функціонал може бути розроблений і

тестований у окремих гілках, а потім злитий у мастер гілку, лише після цього створюється тег, що запускає процес деплою;

3) Зменшення вірогідності випадкових деплоїв – автоматизація деплою при кожному пуші у мастер гілку може призвести до випадкових деплоїв, коли код ще не готовий для продуктивного середовища, використання тегів зменшує цю можливість, оскільки деплой запускається тільки після явного створення тегу, це дає можливість додатково перевірити код перед створенням тегу і запуском деплою;

4) Покращений контроль версій – використання тегів у деплої дає кращий контроль над версіями додатку, дозволяючи чітко визначати, яка саме версія розгорнута у продуктивному середовищі, у разі виникнення проблеми з новою версією, можна легко повернутись до попереднього тегу і розгорнути попередню стабільну версію.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПОЛІПШЕНЬ

4.1 Впровадження Інфраструктури як коду за допомогою Terraform

Створюємо конфігураційні файли Terraform, де визначаємо нашу інфраструктуру AWS у файлах .tf. Далі для більшої зручності всі конфігураційні файли відправляємо у репозіторій GitHub, Цей репозиторій містить конфігурації Terraform для налаштування інфраструктури AWS, включаючи VPC, підмережі, групи безпеки, екземпляри EC2 та ELB. На рис. 4.1 наведена структура розташування файлів Terraform у сховищі.

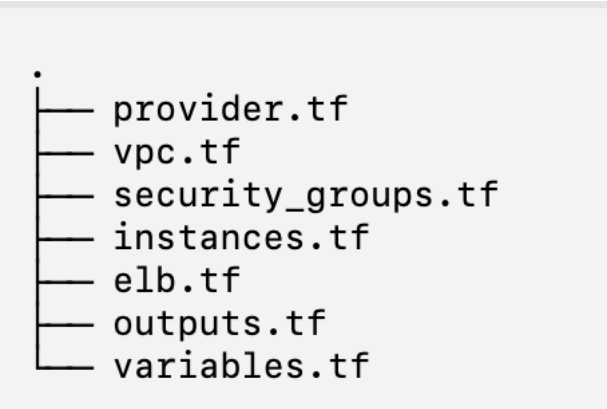

Рисунок 4.1 – Структура файлів Terraform у сховищі

На рис. 4.2 наведена конфігурація для створення екземпляра EC2 для серверу. Де визначаємо екземпляр EC2 під керуванням Ubuntu 20.04, використовуючи останню Ami id від Canonical та пару ключів для доступу по SSH.

```
resource "aws_instance" "web" {
  ami                    = data.aws_ami.ubuntu.id
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.main.id
  vpc_security_group_ids = [aws_security_group.web_sg.id]
  associate_public_ip_address = true

  key_name      = aws_key_pair.deploy.key_name

  tags = {
    Name = "web-server"
  }
}

resource "aws_key_pair" "deploy" {
  key_name   = "deployment_key"
  public_key = file("~/.ssh/id_rsa.pub")
}
```

Рисунок 4.2 – Конфігурація екземпляру сервера EC2

На рис. 4.3 наведена остаточна діаграма хмарної інфраструктури у AWS

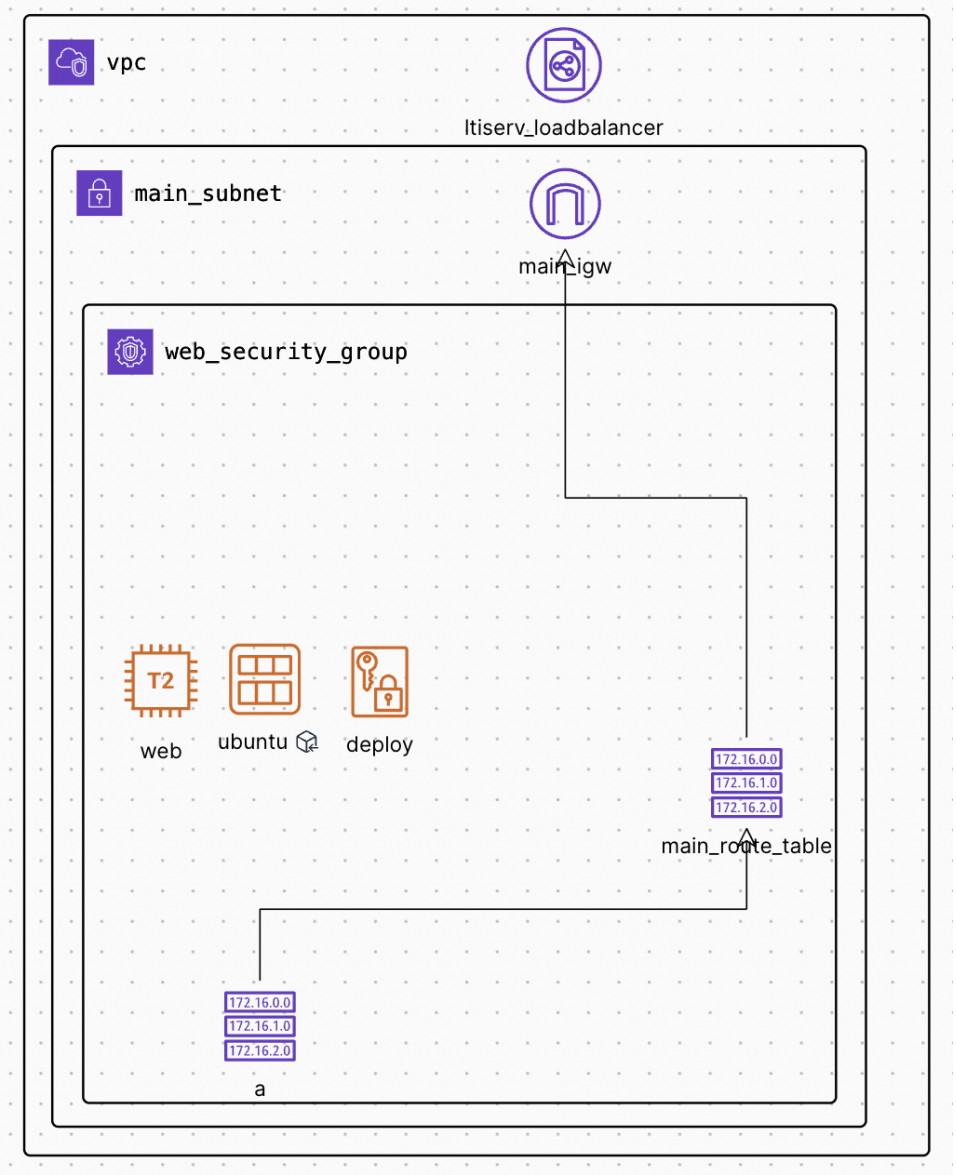

Рисунок 4.3 – Хмарна інфраструктура у AWS

4.2 Впровадження конфігурації як коду за допомогою Ansible

Ansible використовуємо для автоматизації конфігурації та налаштування сервера після того, як він був створений у хмарній інфраструктурі за допомогою Terraform. Структура розташування конфігураційних файлів Ansible наведена на рис. 4.4.

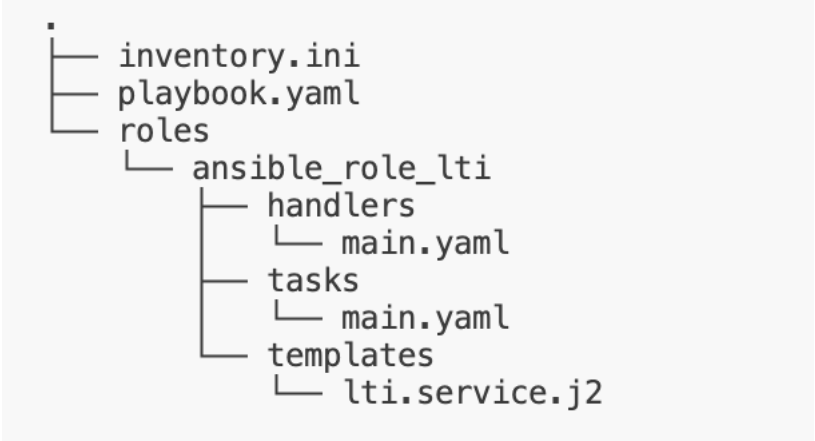

Рисунок 4.4 – Структура розташування файлів Ansible

Цей Ansible плейбук призначений для налаштування сервера з необхідними залежностями, встановлення Python 3.11, розгортання додатку LTI (Learning Tools Interoperability) та забезпечення його запуску як сервісу за допомогою systemd. Нижче наведено детальний опис кожного завдання в плейбуці.

Оновлюємо список доступних пакетів та їхніх версій, без їх встановлення або оновлення та встановлюємо список основних залежностей збірки, необхідних для компіляції та встановлення програмного забезпечення, в тому числі Python 3.11, рис. 4.5

```yaml
- name: Update apt package list
  apt:
    update_cache: yes
  become: yes

- name: Ensure build dependencies are installed
  apt:
    name: "{{ item }}"
    state: present
    update_cache: yes
  with_items:
    - software-properties-common
    - build-essential
    - libssl-dev
    - zlib1g-dev
    - libncurses5-dev
    - libnss3-dev
    - libgdbm-dev
    - libreadline-dev
    - libffi-dev
  become: yes

- name: Add deadsnakes PPA
  apt_repository:
    repo: ppa:deadsnakes/ppa
    state: present
  become: yes

- name: Install Python 3.11
  apt:
    name: python3.11
    state: present
  become: yes

- name: Ensure pip and venv for Python 3.11 are installed
  apt:
    name: "{{ item }}"
    state: present
  with_items:
    - python3.11-venv
    - python3.11-distutils
  become: yes
```

Рисунок 4.5 – Оновлення та встановлення залежностей

Далі проходить перевірка існування каталогу /opt/apps/lti з правильними правами та дозволами, копіювання файли програми LTI з локального каталогу-джерела до цільового каталогу на сервері, створення віртуального середовища Python 3.11 у заданому каталозі та встановлення у нього залежностей Python, перелічених у файлі requirements.txt, програмний код цих дій зображений на рис 4.6

```
- name: Ensure the target directory exists
  file:
    path: /opt/apps/lti
    state: directory
    owner: ubuntu
    group: ubuntu
    mode: '0755'
  become: yes

- name: Copy the entire lti directory recursively
  copy:
    src: ../../../server/lti/
    dest: /opt/apps/lti
    owner: ubuntu
    group: ubuntu
    mode: '0755'
  become: yes

- name: Create a virtual environment with Python 3.11
  command: /usr/bin/python3.11 -m venv /opt/apps/lti/env
  args:
    creates: /opt/apps/lti/env/bin/python
  become: yes

- name: Install dependencies in the virtual environment
  pip:
    requirements: /opt/apps/lti/app/requirements.txt
    virtualenv: /opt/apps/lti/env
```

Рисунок 4.5 – Створення директорії серверу та залежностей Python

Далі проходить створення службового файлу systemd для програми LTI з шаблону Jinja2, перезавантаження конфігурації менеджера systemd для розпізнавання нового службового файлу та перевірка, що службу LTI

увімкнено для запуску під час завантаження і що вона запущена, програмний код зображений на рис. 4.6.

```yaml
- name: Create systemd service file
  template:
    src: lti.service.j2
    dest: /etc/systemd/system/lti.service
    owner: ubuntu
    group: ubuntu
    mode: '0644'
  notify:
    - Restart LTI service
  become: yes

- name: Reload systemd daemon
  systemd:
    daemon_reload: yes
  become: yes

- name: Enable and start the LTI service
  systemd:
    name: lti
    enabled: yes
    state: started
  become: yes
```

Рисунок 4.6 – Створення службового файлу та запуск служби

5 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

5.1 Ефективність впровадження Terraform та Ansible

Terraform – це інструмент, який застосовує IaC для забезпечення інфраструктури. Він спрощує створення хмарної інфраструктури, оскільки дозволяючи користувачам створювати, змінювати та контролювати версії за допомогою конфігураційних файлів. Стан інфраструктури визначається в конфігураційних файлах Terraform конфігураційних файлах і забезпечується виконанням скриптів через бінарний файл Terraform. Однією з головних переваг використання Terraform є простота використання та гнучкість у додаванні або модифікації інфраструктури для різних хмарних провайдерів.

За допомогою Terraform було створено кодову базу, яка дозволяє консистентно та швидко розгортати інфраструктуру. На рис. 5.1 зображений графк зменшення часу розгортання інфраструктури після впровадження Terraform. По осі X представлені місяці, а по осі Y — час розгортання в хвилинах. Як можна побачити, інтеграція Terraform значно знизила час, необхідний для розгортання інфраструктури, що демонструє ефективність автоматизації процесів.

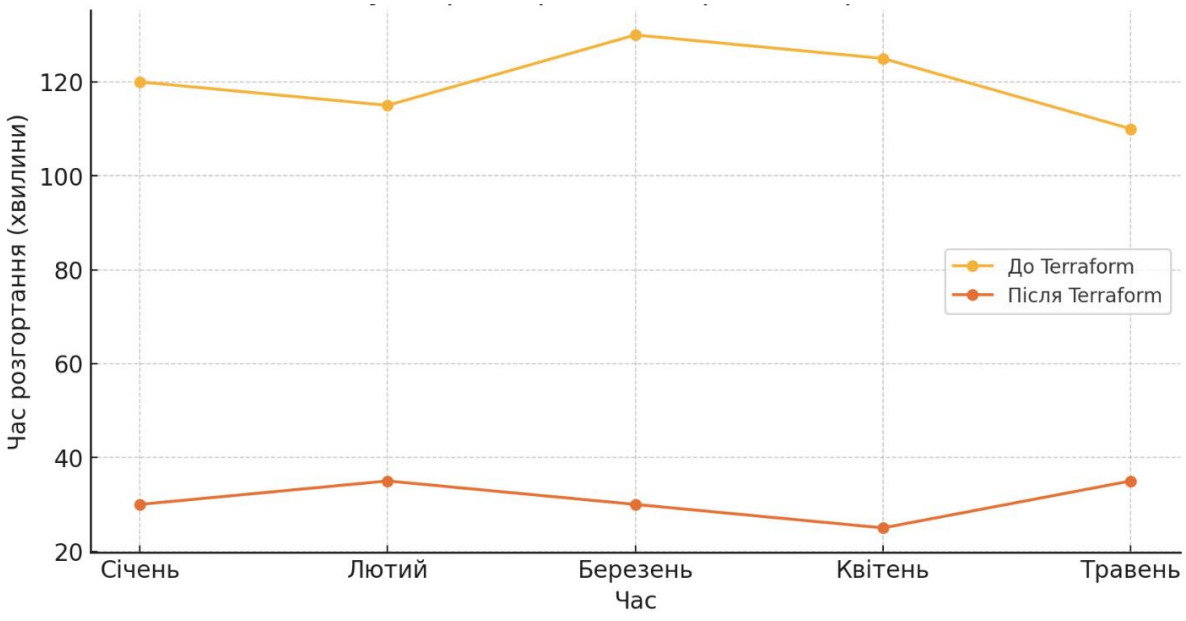

Рисунок 5.1 – Графік порівняння часу розгортання

Файли Terraform визначаються за допомогою мови конфігурації HCL (HashiCorp Configuration Language). Це включає функції виразів, які пропонують потужні можливості для динамічного створення ресурсів і значень, маніпулювання структурами даних і виконання простих або складних обчислень у коді. Функції виразів допомагають зробити код Terraform більш динамічним і гнучким. Це зменшує кількість повторюваної або ручної роботи.

Ansible - це інструмент автоматизації IT, який дозволяє використовувати інфраструктуру як код (IaC). Проект Ansible започаткував Майкл ДеХаан. Перший реліз Ansible був оприлюднений 20 лютого 2012 року. Майкл черпав натхнення з декількох інструментів, які він написав, а також з деякого практичного досвіду управління конфігурацією на той час. Деякі з унікальних атрибутів Ansible, такі як модульна архітектура та безагентний підхід, швидко привернули увагу у світі відкритого програмного забезпечення.

Перехід на Ansible для автоматичного конфігурування сервера LTI замість ручного налаштування також зменшує можливість помилок і підвищує ефективність, автоматизація таких процесів зменшує час конфігурації з кількох годин до лічених хвилин.

5.2 Поліпшення безперервної доставки коду

Підтримка хмарних технологій для розгортання додатків шляхом реалізації концепцій безперервної інтеграції та безперервного розгортання (CI/CD). CI/CD - це концепція, яка функціонує для автоматичного виконання процесу розгортання, що допомагає перевіряти якість і функціонування кожної програми, яка створюється, і допомагає виявляти помилки раніше, роблячи її більш ефективною, результативною і заощаджуючи більше часу в процесі створення програми. Це може допомогти індустрії програмного забезпечення досягти високої якості та продуктивності.

Інтеграція етапу тестування перед розгортанням (CI/CD pipeline) дозволяє забезпечити більшу надійність коду. Дослідження показують, що включення автоматичних тестів може значно знижувати кількість багів у продукті, покращувати якість коду і забезпечувати кращу документацію процесів.

Вдосконалення Github Actions включало перехід від одного кроку до подвійного етапу розгортання та тестування, що могло спочатку здатися, як збільшення часу на розгортання. Однак, додавання автоматизованих тестів фактично може скоротити загальний час впровадження, оскільки зменшується кількість випадків невдач при розгортанні і потреба в подальших виправленнях.

На рис. 5.2 зображено графік кількісті виявлених помилок перед та після впровадження автоматизованих тестів у процес розробки через GitHub Actions. Точки показують кількість помилок за кожен місяць. З графіка видно, що після інтеграції тестування кількість помилок значно знизилася, що демонструє високу ефективність тестувальних процедур у виявленні та виправленні багів на ранніх етапах розробки.

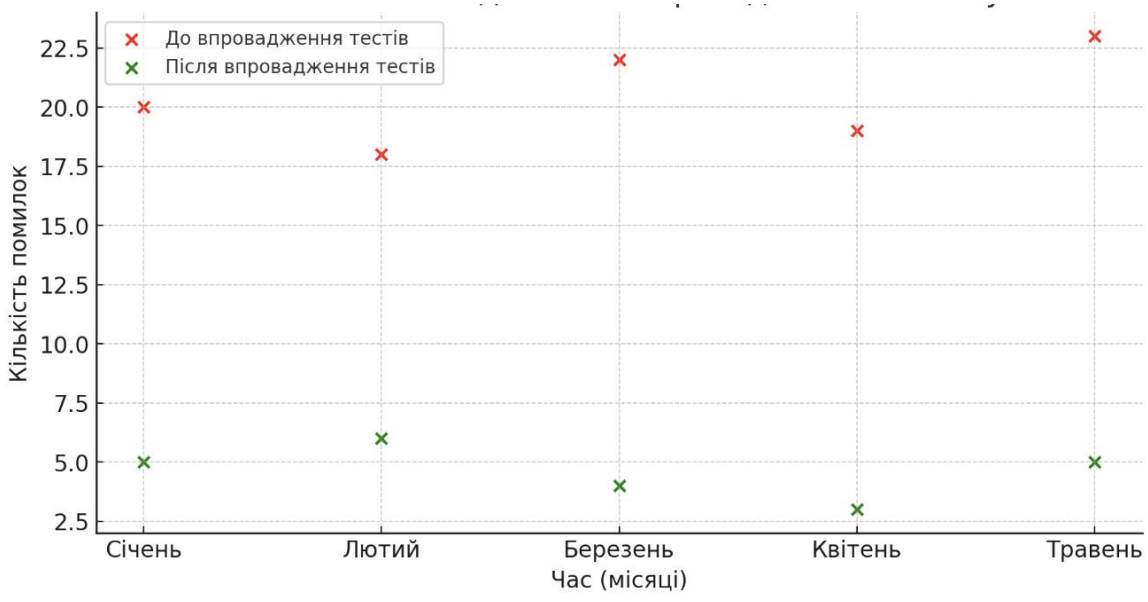

Рисунок 5.2 – Графік залежності виявлених помилок у часі

Впровадження автоматичних тестів в процес розгортання має зменшити кількість невдалих розгортань. На рис 5.3 зображено графік змін в тривалості розгортання проекту до та після впровадження автоматичних тестів. Кожна точка представляє тривалість розгортання для різних версій релізів. Як видно з графіка, після додавання автоматичного тестування час, необхідний для розгортання, знизився, що свідчить про зменшення кількості невдалих спроб розгортання та підвищення ефективності розробки.

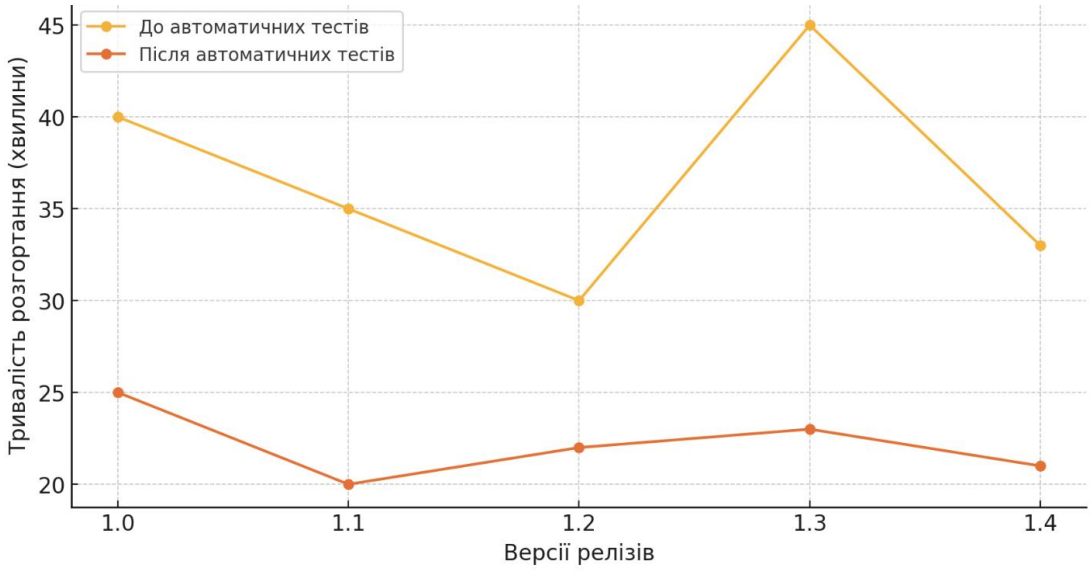

Рисунок 5.3 – Графік тривалості розгортання серверної частини

5.3 Дослідження відповідності архітектури стандартам індустрії

Оформимо таблицю 5.1, яка відображатиме важливі аспекти, яким повинен відповідати клієнт-серверний додаток.

Таблиця 5.1 - Порівняння відповідності використаних рішень стандартам з подальшими поліпшеннями.

Пункт	Як було	Як стало	Наступні поліпшення
Швидкість розгортання	Розгортання серверів виконувалось вручну, що займало багато часу.	Розгортання через Terraform, що дозволяє швидко і автоматизовано налаштовувати інфраструктуру.	Інтеграція з cloud-специфічними сервісами для подальшого зниження часу розгортання і збільшення еластичності.
Конфігурація сервера	Конфігурація виконувалась вручну, з високим ризиком помилок.	Автоматичне конфігурування за допомогою Ansible, що забезпечує консистентність і знижує ризики.	Застосування Immutable Infrastructure підходу для уникнення конфігураційного дрейфу.
Тестування коду	Відсутність систематичного тестування до розгортання.	Додавання автоматичних тестів (Bandit, Flake8) в GitHub Actions перед розгортанням.	Впровадження інтеграційних та навантажувальних тестів для перевірки взаємодії компонентів та шкальованості.
Управління версіями	Розгортання на основі останніх змін у гілці master.	Розгортання за версіями, що відслідковується через теги в Git, забезпечуючи кращий контроль версій.	Автоматизація управління залежностями та сумісності між версіями для забезпечення безперебійної роботи додатку.
Мінімізація помилок при розгортанні	Високий ризик помилок через ручні процеси.	Зниження помилок через автоматизацію та валідацію кроків розгортання.	Впровадження розгортання за допомогою Blue/Green або Canary стратегій для мінімізації впливу помилок.

Час на виправлення помилок	Виправлення помилок займало багато часу після розгортання.	Швидке виправлення помилок завдяки ранньому виявленню на етапі тестування.	Впровадження системи моніторингу та алертів для оперативного виявлення та виправлення помилок у реальному часі

Ця розширена таблиця порівняння не тільки підкреслює ефективність впровадженії вдосконалень проекту, але й надає чіткий шлях для майбутніх удосконалень. Кожен з запропонованих напрямків може забезпечити значні переваги для стабільності, безпеки та ефективності клієнт серверної архітектури додатків.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було зроблено дослідження ефективності поточної архітектури системи, що розроблена на базі системи керування навчанням Moodle та інтеграції її з додатком розробленим для комунікації між системами LMS та LTI. В ході дослідження було виявлено декілька напрямків вдосконалення архітектури, які були впроваджені у проект, що позитивно сприяли на якість, стабільність та час доставки програмного коду.

До таких поліпшень відносяться: опис та розгортання хмарної інфраструктури у коді за допомогою Terraform; конфігурація серверу описана у Ansible коді; впровадження версіонування серверного додатку та розподілення системи доставки коду на декілька етапів з додавання тестування програмного коду серверної частини проєкту.

Вдосконалення не є кінцевими і у роботі наведені подальші напрямки з якими можна працювати у майбутньому.

Крім того за результатами досліджень опубліковано тези доповіді на науковій конференції [6]. Таким чином всі пункти технічного завдання на работу виконано в повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація до LMS Moodle [Електронний ресурс] Режим доступу до ресурсу: https://docs.moodle.org/403/en/Main_page

2. Специфікація LTI [Електронний ресурс] Режим доступу до ресурсу: https://www.imsglobal.org/spec/lti/v1p3/impl/

3. Communication protocol [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Communication_protocol

4. Serverless and IaC // International Journal For Science Technology And Engineering Режим доступу до ресурсу: https://www.ijraset.com/best-journal/serverless-and-iac Serverless and IaC. International Journal For Science Technology And Engineering, (2023).;11(5):3271-3279. doi: 10.22214/ijraset.2023.51356

5. Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures [Електронний ресурс] Режим доступу до ресурсу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith Alde, Alanda., HA, Mooduto., Rizka, Hadelina. (2022). Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures. JITCE (Journal of Information Technology and Computer Engineering), 6(02), 50-56. Available from: 10.25077/jitce.6.02.50-56.2022

6. Федорченко О. М. Дослідження ефективності системи автоматизації перевірки виконання практичних завдань в локальному віртуальному оточенні користувача / О. М. Федорченко, А. І. Костромицький. // 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті » Харків, 16 – 18 квітня 2024.