Харківський національний університет радіоелектроніки
(повне найменування вищого навчального закладу)

Факультет *інфокомунікацій*
(повна назва)

Кафедра *інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

на тему *Дослідження можливостей взаємодії із тимчасовими акаунтами системи автоматичної перевірки навичок роботи з AWS*

Виконав: студент *2* курсу, групи *ІМІм-21-2*
напряму підготовки
172 "Телекомунікації і радіотехніка"
(шифр і назва напряму підготовки)
Мишко М.М.
(прізвище та ініціали)
Керівник *Костромицький А.І*
(прізвище та ініціали)

Допускається до захисту

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

Харків - 2023 року

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)*

Напрям підготовки *172 «Телекомунікації і радіотехніка»*
(шифр і назва)

ЗАТВЕРДЖУЮ

Зав.кафедри _____
(Підпис)

"____" _____ 2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мишко Максиму Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження можливостей взаємодії із тимчасовими акаунтами системи автоматичної перевірки навичок роботи з AWS*

затверджені наказом ВНЗ від "*17*" *березня* 2023 року № *275 Ст.*

2. Строк подання студентом роботи *19 травня 2023 року*

3. Вихідні дані до роботи *Дослідити можливість інтеграції навчальної платформи, що автоматично перевіряє практичні навички роботи з AWS із стороннім сервісом, що надає можливість створювати тимчасові акаунти, та використовувати їх замість власних акаунтів студентів.*

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

Вступ
1. Сучасні провайдери хмарних сервісів
2. Існуючі рішення для розвитку навичок роботи з AWS
3. Огляд платформи для здобуття навичок роботи з AWS
4. Дослідження можливості інтеграції навчальної платформи із сервісом надання тимчасових AWS акаунтів
5. Програмна реалізація інтеграції навчальної платформи із сервісом надання тимчасових AWS аккаунтів
6. Перевірка працездатності рішення та огляд можливостей для подальшого вдосконалення
Висновок

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

Слайди у форматі PowerPoint (назва та мета роботи, актуальність, способи навчання з AWS, концепція навчальної платформи та її технічні відомості, проблема платформи та можливість вирішення, огляд стороннього сервісу, автоматизація з Python та Selenium, збірка проекту з Docker та Pipenv, інтеграція контейнера у Jenkins, демонстрація)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	*доц. Костромицький А.І*	21.03.23	

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Ознайомлення і уточнення завдання*	*21.03-22.03.2023*	
2	*Підбір і вивчення літератури*	*23.03-25.03.2023*	
3	*Розробка розділу 1*	*26.03-03.04.2023*	
4	*Розробка розділу 2*	*04.03-11.04.2023*	
5	*Розробка розділу 3*	*12.04-20.04.2023*	
6	*Розробка розділу 4*	*21.04-27.04.2023*	
7	*Розробка розділу 5*	*28.04-07.05.2023*	
8	*Розробка розділу 6*	*08.05-15.05.2023*	
9	*Оформлення презентаційного матеріалу,*	*16.05-19.05.2023*	
	підготовка до захисту у ЕК		

Дата видачі завдання *21 березня 2023 р.*

Студент _____ *Мишко М.М.*
(підпис) (прізвище та ініціали)

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

РЕФЕРАТ

Пояснювальна записка: 57 с., 21 рис., 7 джерел

Об’єкт роботи – навчальна платформа з автоматичної перевіркою навичок роботи з AWS.

Мета роботи – створити програмну інтеграцію для навчальної платформи зі стороннім сервісом надання тимчасових акаунтів AWS.

Розглянуті сучасні тенденції розвитку хмарних обчислень. Розглянута архітектура навчальної платформи з автоматичною перевіркою навичок роботи з AWS. Розроблена інтеграція платформи зі стороннім сервісом. Протестована працездатність рішення та розглянуті можливості для розвитку.

AWS, TERRAFORM, FASTAPI, PYTHON, SELENIUM, JENKINS, PIPENV, DOCKER

ABSTRACT

Explanatory note: 57 pages, 21 figures, 7 sources

Object of the study is an educational platform with automatic skill assessment for AWS.

The aim of the work is to create software integration for the educational platform with a third-party service for temporary AWS accounts provisioning. Modern trends in cloud computing are discussed. The architecture of the educational platform with automatic skill assessment for AWS is examined. The integration of the platform with a third-party service is developed. The functionality of the solution is tested, and possibilities for further development are discussed.

AWS, TERRAFORM, FASTAPI, PYTHON, SELENIUM, JENKINS, PIPENV, DOCKER

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AWS (Amazon Web Service) – провайдер хмарних сервісів компанії Amazon.

GCP (Google Cloud Platform) – провайдер хмарних сервісів компанії Google.

IaC (Infrastructure as Code) – підхід до керування інфраструктурою за допомогою коду.

CI/CD (Continuous Integration/Continuous Delivery) – підхід до автоматизації процесів у розробці.

IT (Information Technologies) – інформаційні технології.

API (Application Programming Interface) – інтерфейс взаємодії між програмами.

JSON (JavaScript Object Notation) – формат обміну даних.

JWT (JSON Web Token) – стандарт передачі даних між сутностями у мережі.

SQL (Structured Query Language) – мова структурованих запитів.

HCL (HashiCorp Configuration Language) – мова конфігурування компанії HashiCorp.

HTML (HyperText Markup Language) - мова розмітки гіпертексту

HTTP (Hypertext Transfer Protocol) – протокол передачі документів у веб просторі.

CLI (Command-line Interface) – інтерфейс командного рядка

ВСТУП

За останні роки популярність сервісів хмарних обчислювань стабільно зростає, що зумовлено значними перевагами, які вони принесли у світ IT-індустрії. Одним з провідних провайдерів хмарних послуг є Amazon Web Services (AWS), популярність якого продовжує стрімко зростати разом зі впливом на ринок хмарних обчислень. На рисунку В.1 показано частки прибутку різних компаній від загального прибутку усіх провайдерів хмарних сервісів за перший квартал 2023 року.

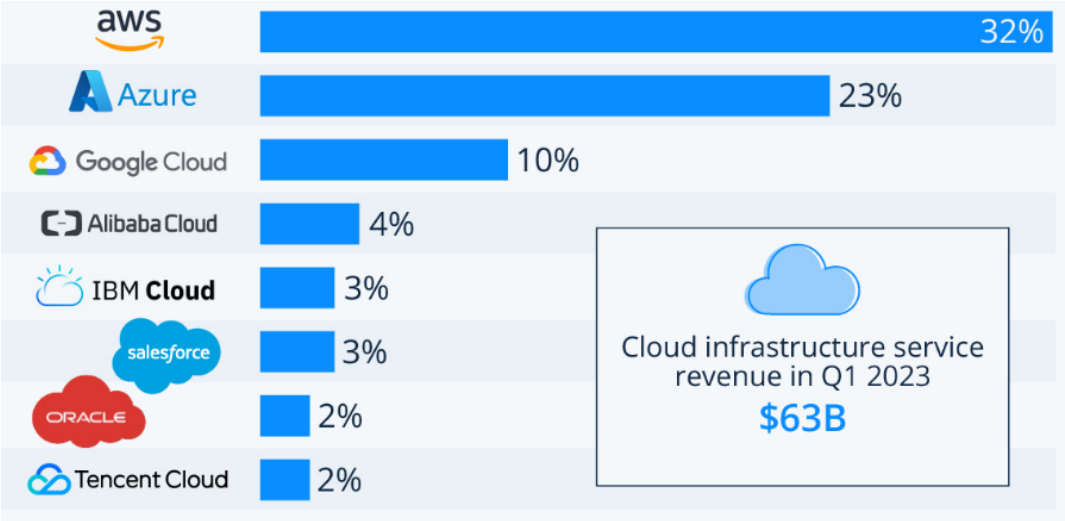


Рисунок В.1 – Прибутки провідних провайдерів хмарних послуг за перший квартал 2023 року

Зростаюча популярність хмарних обчислень, зокрема AWS, породжує попит на кваліфікованих спеціалістів, які повинні володіти глибокими знаннями та розумінням сервісів, що надає цей провайдер. Компанії, що використовують AWS, шукають професіоналів з експертизою у цій області, оскільки вони бажають максимізувати переваги хмарних обчислень та забезпечити ефективну роботу своїх інфраструктур та додатків.

На щастя, на сьогоднішній день існує ряд можливостей для підготовки та отримання кваліфікації у роботі з AWS. Навчальні платформи, такі як AWS

Training and Certification, надають широкий спектр курсів та сертифікаційних програм, які дозволяють студентам та професіоналам отримати необхідні знання та навички.

Крім того, самостійне вивчення також є важливою складовою підготовки. AWS надає доступ до документації, різноманітних посібників, блогів, вебінарів та відеоуроків, які допомагають користувачам самостійно освоїти платформу та розширити свої знання. Крім того, існують спільноти, форуми та соціальні мережі, де спеціалісти можуть обмінюватись досвідом та задавати питання.

Однак, практичний досвід є невід'ємною частиною підготовки. Можливості практичних вправ та лабораторних робіт з AWS надаються через онлайн-платформи, які дозволяють студентам експериментувати та відпрацьовувати свої навички. Це дозволяє засвоїти практичні навички роботи з різноманітними AWS сервісами, налаштуванням, моніторингом та управлінням інфраструктурою.

В даній дипломній роботі розглянуто як влаштована існуюча платформа з системою автоматичної перевірки навичок роботи з AWS, а також її поліпшення метою розробки інтеграції зі стороннім сервісом, що надає тимчасові облікові записи AWS. Замість створення власних AWS акаунтів студенти зможуть використовувати тимчасові акаунти, що забезпечує більшу гнучкість та швидкість доступу до AWS ресурсів, економію коштів студентів, та сприяє більш ефективному навчанню та отриманню практичного досвіду з AWS.

1 СУЧАСНІ ПРОВАЙДЕРИ ХМАРНИХ СЕРВІСІВ

1.1 Зростання популярності хмарних сервісів та перехід бізнесів на хмарну інфраструктуру.

Хмарні обчислення є прогресивною парадигмою обчислювальної інфраструктури, що надає доступ до обчислювальних ресурсів, зберігання даних та інших послуг через Інтернет. Замість того, щоб мати власну фізичну інфраструктуру, компанії можуть орендувати ці ресурси від провайдерів хмарних сервісів за потребою.

Історія хмарних обчислень сягає своїх корінь у 1960-х роках, коли виникла ідея "розподіленої обчислювальної мережі" (distributed computing network). Проте, справжній прорив в розвитку хмарних обчислень стався в 2006 році, коли компанія Amazon представила свою платформу Amazon Web Services (AWS). AWS створила революцію у сфері інформаційних технологій, надаючи гнучкі та масштабовані обчислювальні ресурси через Інтернет. [1]

На сучасному ринку спостерігається стрімке зростання популярності хмарних сервісів. Компанії все частіше виявляють потребу в масштабованих та гнучких обчислювальних ресурсах, які можуть забезпечити ефективну роботу їх інфраструктури та додатків. Підприємства розуміють, що перехід до хмарної інфраструктури дозволяє знизити витрати на обладнання, забезпечити високу доступність та надійність, а також більш швидке розгортання нових продуктів та послуг. На рисунку 1.1 можна побачити графік зростання популярності хмарних сервісів у країнах Європи з 2020 по 2021 роках.

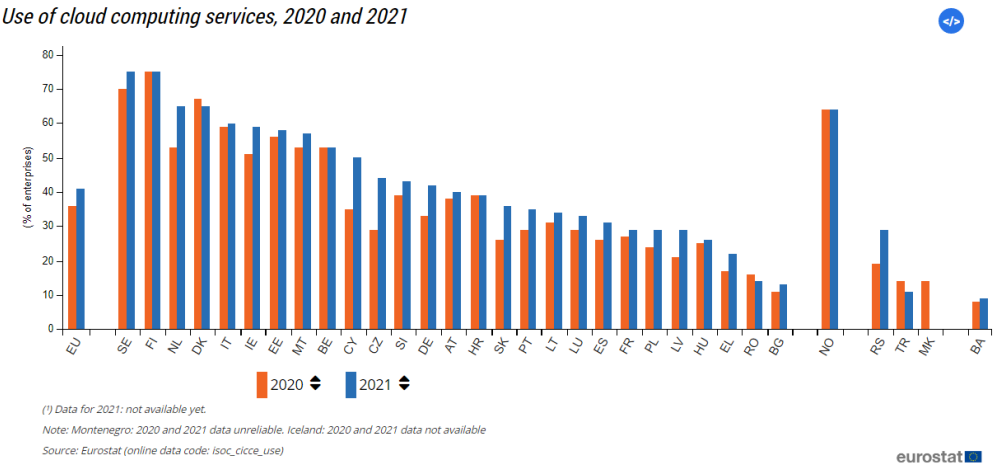


Рисунок 1.1 – Графік використання хмарних ресурсів у Європі

Ключовою причиною популярності хмарних сервісів є спільні характеристики майже всіх хмарних сервісів, що забезпечують багато переваг для бізнесів. Серед них можна виділити наступні:

1) Масштабованість і гнучкість. Провайдери хмарних сервісів забезпечують можливість масштабування ресурсів відповідно до потреб користувачів. Вони пропонують широкий спектр обчислювальних, мережевих послуг та послуг зберігання даних, які можуть бути легко адаптовані до зростаючих потреб бізнесу.

2) Доступність та надійність. Провайдери хмарних обчислень гарантують високу доступність своїх послуг, забезпечуючи безперебійну роботу інфраструктури та систем. Вони розташовують свої центри обробки даних у різних географічних регіонах, що забезпечує резервування та відновлення даних в разі виникнення проблем.

3) Безпека. Провайдери хмарних сервісів вкладають значні зусилля в забезпечення безпеки даних та інфраструктури. Вони використовують шифрування, механізми аутентифікації та контролю доступу, а також забезпечують захист від зламів та злочинних дій.

4) Гнучкість в ціноутворенні. Провайдери хмарних сервісів пропонують різні моделі ціноутворення, такі як оплата за використання, пакети послуг або

підписка. Це дозволяє бізнесам вибрати найбільш оптимальний варіант залежно від своїх потреб та бюджету.

5) Екосистема та інтеграція. Провайдери хмарних сервісів надають широкий спектр інструментів, сервісів та додатків, які допомагають підприємствам розгортати та управляти своїми хмарними ресурсами. Вони надають можливість інтеграції з різноманітними програмними рішеннями та сервісами, що дозволяє компаніям створювати комплексні та ефективні рішення для своїх бізнес-потреб. Такі інтегровані середовища дозволяють підприємствам здійснювати міграцію та управляти своїми даними та додатками у хмарному середовищі, спрощуючи розробку, впровадження та управління хмарними ресурсами. Це забезпечує зручність та ефективність роботи бізнесу у хмарному середовищі.

6) Технічна підтримка. Провайдери хмарних сервісів забезпечують технічну підтримку своїм користувачам. Це може включати як консультації окремих користувачів, так і виділення окремих спеціалістів у разі придбання сервісу підтримки рівня підприємства.

7) Інновації та оновлення. Провайдери хмарних сервісів постійно вдосконалюють свої послуги, впроваджуючи нові технології та функціонал. Вони пропонують оновлення, що забезпечують користувачам доступ до передових можливостей та нових інструментів.

8) Універсальність. Провайдери хмарних сервісів можуть пропонувати широкий спектр рішень, які підходять для малого бізнесу, стартапів, середніх підприємств або великих корпорацій. Вони забезпечують гнучкість вибору та можливість масштабування ресурсів відповідно до розміру та потреб бізнесу.

На сьогоднішній день вже багато відомих компаній успішно перейшли на хмарну інфраструктуру та розгорнули свої проекти. Одним з найвідоміших прикладів є Netflix, який перетворився з компанії, що надає послуги по оренді DVD, на провідний стрімінговий сервіс. Завдяки хмарній інфраструктурі, Netflix забезпечує високу доступність свого відео контенту та швидке розгортання нових функцій для своїх користувачів. [2]

Іншим вражаючим прикладом є Airbnb, платформа для бронювання житла. Завдяки хмарним сервісам, Airbnb змогла масштабувати свою інфраструктуру та обробку даних, щоб відповідати потребам мільйонів користувачів по всьому світу. [2]

Також варто згадати компанію Spotify, яка перетворила спосіб, яким ми слухаємо музику. Завдяки хмарним обчисленням, Spotify може забезпечувати мільйонам користувачів доступ до величезної кількості музичних творів. [2]

Ці приклади підтверджують, що хмарні сервіси стають вирішальним фактором у сфері інформаційних технологій та сприяють інноваціям та змінам у різних галузях. Вони дозволяють компаніям швидко реагувати на зміни в ринкових умовах, підвищувати ефективність та забезпечувати високу якість послуг для своїх користувачів.

1.2 Концепція інфраструктури як коду як метод керування хмарною інфраструктурою

Інфраструктура як код (Infrastructure as Code, IaC) - це практика управління та налаштування IT-інфраструктур за допомогою коду замість традиційної інтерактивної конфігурації. Ця концепція виникла з потреби більшої автоматизації, швидкості та ефективності в розгортанні та управлінні IT-інфраструктурою, особливо з розвитком та популяризацією хмарних технологій.

Інфраструктура як код вирішує ряд проблем, пов'язаних з традиційним управлінням IT-інфраструктурою. Вона забезпечує швидкість та ефективність, знижує можливість людських помилок, забезпечує послідовність та відтворюваність інфраструктури, а також дозволяє легко масштабувати інфраструктуру відповідно до потреб організації.

Однак, IaC не обходиться без недоліків. Наприклад, вона може вимагати значних витрат на початкову конфігурацію та обслуговування, а також потребує спеціалістів високого рівня технічної кваліфікації та розуміння інформаційних системи.

Є декілька ключових інструментів, які використовуються для реалізації IaC, включаючи Terraform, Ansible, Chef, Puppet та інші. Кожен з цих інструментів має свої власні характеристики та переваги, і вибір між ними залежить від конкретних потреб організації.

В умовах хмарних технологій, IaC відіграє важливу роль, дозволяючи організаціям більш ефективно та гнучко управляти своєю інфраструктурою. Це особливо важливо для організацій, які переходять від статичних, власних центрів обробки даних до гнучких хмарних середовищ.

1.3 Принципи та використання CI/CD в хмарних середовищах

Continuous Integration / Continuous Deployment (CI/CD) - це практика розробки програмного забезпечення, яка включає постійну інтеграцію змін до коду проекту і їх подальше автоматичне розгортання.

Continuous Integration (CI) - це процес, при якому зміни вихідного коду регулярно (часто декілька разів на день) інтегруються в основну кодову базу. Це дає змогу виявити і вирішити проблеми та помилки на ранніх стадіях, забезпечуючи стабільність та якість коду.

Continuous Deployment (CD), з іншого боку, автоматизує розгортання змін в коді до виробничого середовища. Це забезпечує швидке внесення змін до продукту без необхідності вручну розгортати кожну зміну.

У контексті хмарних технологій, CI/CD створює потужну комбінацію, яка дозволяє швидко та ефективно вносити зміни до програмного забезпечення. Це особливо корисно для хмарних додатків, які потребують постійного оновлення та удосконалення.

Є різні інструменти для реалізації CI/CD, такі як Jenkins, GitLab CI, Travis CI, CircleCI, та інші. Вони забезпечують автоматичне тестування коду, його збірку, інтеграцію та розгортання, забезпечуючи безперервність і якість розробки.

CI/CD в хмарних середовищах також має свої виклики. Зокрема, потрібно забезпечити безпеку процесу, управління конфіденційністю та доступом, а також масштабування CI/CD-процесів для великих та розподілених середовищ.

Використання CI/CD у хмарних технологіях має багато переваг. Воно забезпечує швидке внесення змін, покращує продуктивність команди, знижує час виходу на ринок та забезпечує високу якість продукту. Також, це дає змогу розробникам фокусуватися на коді, замість процесів розгортання та управління інфраструктурою.

2 ІСНУЮЧІ РІШЕННЯ ДЛЯ РОЗВИТКУ НАВИЧОК РОБОТИ З AWS

2.1 Сучасні сервіси для розвитку навичок роботи з AWS

На сьогоднішній день існує багато рішень, які допомагають студентам розвивати навички роботи з AWS. Деякі з них є безкоштовними, а інші - платними, і кожен з них має свої переваги та недоліки.

Одним з безкоштовних рішень є AWS Free Tier ,що надає безкоштовну підписку на AWS на перший рік після реєстрації. Це дає студентам можливість вивчати AWS та виконувати практичні завдання безкоштовно. Проте, не всі ресурси входять до AWS Free Tier, що може стати проблемою для студентів з обмеженим бюджетом.

Іншим рішенням є AWS Educate. Це програма, яка надає безкоштовний доступ до AWS для студентів, викладачів та інших осіб, пов'язаних з освітою. AWS Educate надає студентам можливість отримати навички використання AWS через практичні завдання та курси. Однак, для отримання доступу до AWS Educate, студентам потрібно отримати запрошення від викладача або реєструватися самостійно і проходити верифікацію, що може зайняти деякий час.

Також існує багато комерційних платформ, які надають навчальні курси та лабораторні роботи з AWS. Наприклад, Linux Academy, A Cloud Guru та Cloud Academy. Вони надають широкий спектр курсів, які допомагають студентам вивчити AWS та отримати практичний досвід використання різних сервісів. Однак, вартість таких платформ може бути досить високою, що може стати проблемою для студентів з обмеженим бюджетом.

2.2. Фінансові та безпекові виклики студентів

Фінансові виклики студентів можуть бути важливим фактором, що обмежує їхню можливість вивчати та розвивати навички роботи з AWS. Більшість сервісів AWS є платними, і ціни на них можуть бути досить високими. Крім того,

користування AWS може включати додаткові витрати, такі як оплата за трафік, що може збільшувати загальні витрати студентів. AWS Free Tier частково вирішує цю проблему, проте, як було зазначено раніше, не всі сервіси компенсуються за цією програмою. Крім того, студент повинен буде уважно перевіряти чи компенсуються витрати за той чи інший сервіс і не забувати видаляти ресурси, щоб не вийти за межі Free Tier. Це може зменшити мотивацію студентів до вивчення та розвитку навичок роботи з AWS, тому що вони можуть вважати це занадто дорогим або недоступним.

2.3. Обмеження існуючих підходів

Існують різні підходи до навчання та розвитку навичок роботи з AWS. Однак, деякі з них можуть мати обмеження, які ускладнюють процес навчання та розвитку навичок.

Один з підходів базується на проходженні комплексних практичних завдань, які охоплюють одразу декілька сервісів. Такі завдання, крім здобуття початкових практичних навичок, спонукають студентів ознайомитися з рекомендованими практиками від професійних фахівців та застосувати їх для виконання завдання. На додаток, складність таких завдань дозволяє студентам зіткнутися із проблемами, які трапляються також на продуктових проектах і отримати досвід з вирішення таких проблем. Цей підхід є досить ефективним, оскільки студенти мають можливість навчитися на реальних завданнях та розбиратися з реальними проблемами, що виникають при роботі з AWS. Однак, цей підхід може бути дорогим та потребувати значних витрат часу та зусиль.

Інший підхід - це використання онлайн-курсів та матеріалів для самостійного навчання. Цей підхід може бути зручним та економічним, оскільки він не потребує значних витрат часу та зусиль. Інколи, до навчальних матеріалів прикріплюють завдання з покроковою інструкцією по створенню та видаленню певних ресурсів, для того, щоб студент міг попрактикуватися. Однак, цей підхід може бути менш ефективним, оскільки студенти можуть не мати можливості

практично застосовувати свої знання та навички у реальних проектах для вирішення складних завдань.

3 ОГЛЯД ПЛАТФОРМИ ДЛЯ ЗДОБУТТЯ НАВИЧОК РОБОТИ З AWS

3.1 Інфраструктура платформи

Навчальна платформа складається з наступних компонентів:

1) Jenkins. Jenkins являє собою безкоштовне програмне забезпечення та служить CI/CD інструментом для автоматичної збірки програмних проектів, їх тестування та розгортання в обчислювальних середовищах. У контексті навчальної платформи Jenkins є веб застосунком з обліковими записами студентів, які можуть авторизуватися та запускати доступні практичні завдання. В свою чергу Jenkins створює необхідні ресурси у власних AWS облікових записах студентів та відображає сторінку із завданням. Коли студент завершує виконання завдання, він нажимає на кнопку перевірки, Jenkins перевіряє чи завдання було виконано, виставляє студенту оцінку та видаляє усі ресурси, що були створені для виконання завдання.

2) DynamoDB. Amazon DynamoDB - це повністю керована NoSQL база даних, яка доступна на платформі Amazon Web Services (AWS). Вона дозволяє зберігати та отримувати дані в реальному часі з мінімальною затримкою та без необхідності відчутно масштабувати систему. Навчальна платформа використовує DynamoDB для зберігання даних студентів, включаючи ідентифікаційний номер власного AWS акаунту студента, історію виконання завдань студентом з його оцінками та інші службові дані.

3) API. Навчальна платформа має свій власний програмній інтерфейс реалізований за допомогою бібліотеки мови програмування Python – FastAPI. API сервіс платформи приймає HTTP запити від Jenkins та виконує читання та запис даних у DynamoDB.

4) Terraform. Terraform - це інструмент для управління інфраструктурою, який дозволяє декларативно описувати та керувати різними ресурсами інфраструктури, такими як сервери, мережі, бази даних, контейнери та інші, використовуючи код. Підхід IaC згадувався раніше у цій роботі. Terraform

використовує конфігураційні файли у форматі HCL (HashiCorp Configuration Language), в яких декларативно описані різні ресурси інфраструктури та їх залежності. Terraform може працювати з різними провайдерами інфраструктури, такими як Amazon Web Services, Google Cloud Platform, Microsoft Azure, DigitalOcean, та інші. Навчальна платформа використовує Terraform для створення та видалення конкретних ресурсів залежно від обраного студентом завдання. [3]

3.2 Принцип автоматичної перевірки завдань платформою

Послідовність кроків від запуску завдання студентом до його завершення, перевірки та видалення ресурсів виглядає таким чином:

1) Авторизувавшись у платформу, студент запускає пайплайн, що являє собою програмовану послідовність операцій, яку виконує Jenkins, під назвою task_pipeline. Студент має можливість обрати одне з доступних завдань одразу при запуску пайплайну. Якщо він цього не зробить, пайплайн першим чином попросить студента обрати завдання зі списку та підтвердити вибір.

2) Jenkins, за допомогою API-виклику отримує необхідні дані студента з DynamoDB та запускає Terraform скрипт що відповідає обраному завданню. Terraform створює усі необхідні ресурси у стані, якби завдання вже було виконано, та зберігає стан інфраструктури у файлі «terraform.tfstate».

3) Jenkins виконує скрипт відповідно до обраного завдання, який видаляє деякі зі створених ресурсів, тим самим руйнуючи працюючу інфраструктуру. Фактично, виконавши завдання, студент полагодить зламану інфраструктуру та приведе її назад до стану, який описаний у файлі «terraform.tfstate».

4) Jenkins публікує html сторінку із описом завдання та ставить пайплайн на паузу, доки студент не натисне на кнопку перевірки завдання.

5) Отримавши команду перевірки завдання, Jenkins виконує Terraform команду, яка перевіряє стан існуючої інфраструктури зі станом у terraform.tfstate.

6) Якщо стани збігаються, завдання вважається виконаним, студент отримує оцінку та інфраструктура прибирається. Якщо стани не збігаються, студент отримує повідомлення на сторінці пайплайну, що завдання не було виконано.

7) Студент може виправити свої помилки та перевірити виконання ще декілька разів, або завершити завдання до виконання при необхідності. В такому випадку інфраструктура буде видалена і в DynamoDB буде записано, що студент не пройшов завдання.

4 ДОСЛІДЖЕННЯ МОЖЛИВОСТІ ІНТЕГРАЦІЇ НАВЧАЛЬНОЇ ПЛАТФОРМИ ІЗ СЕРВІСОМ НАДАННЯ ТИМЧАСОВИХ AWS АКАУНТІВ

4.1. Переваги інтеграції

Інтеграція навчальної платформи з платформою створення тимчасових AWS-акаунтів може бути ефективним рішенням для навчання та розвитку навичок роботи з AWS. Це дозволить студентам отримати доступ до реальних AWS-ресурсів та сервісів без необхідності користуватися власними аккаунтами AWS та без витрат на користування сервісами AWS.

Навчальна платформа може бути інтегрована з сторонньою платформою, яка дозволяє створювати тимчасові AWS-акаунти за допомогою програмної взаємодії двох платформ.

Серед очікуваних вимог можна виділити наступні:

1) Виключення додаткових витрат на навчання та розвиток навичок роботи з AWS.

2) Створення можливості для студентів отримати практичний досвід роботи з AWS-ресурсами та сервісами на завданнях з реальних проектів без обмежень.

3) Забезпечення безпеки та конфіденційності особистих даних студентів.

4.2. Аналіз можливостей використання Python та Selenium для інтеграції

Python - це потужна та гнучка мова програмування, яка широко використовується для автоматизації та інтеграції з різними платформами. Selenium - це набір інструментів для автоматизації веб-браузера, який також може бути використаний з Python. Розглянемо можливості інтеграції нашої платформи з третьою стороною за допомогою цих інструментів:

1) Автоматизація процесу авторизації. За допомогою Python та Selenium можна автоматизувати процес входу на платформу третьої сторони, використовуючи облікові дані студента.

2) Отримання токена. Після успішної авторизації на платформі, Python та Selenium можуть бути використані для вилучення токена, який надалі використовується для програмного створення та видалення тимчасових облікових записів AWS.

3) Створення та видалення тимчасових облікових записів AWS. З отриманим токеном можна використовувати Python для звернення до API платформи, що використовується для створення та видалення тимчасових облікових записів AWS для студентів. Це дозволяє уникнути використання особистих AWS-аккаунтів студентів та знижує фінансові та безпекові ризики.

4) Інтеграція з навчальною платформою. Отримані дані про облікові записи тимчасових AWS-аккаунтів можуть бути використані нашою навчальною платформою, що забезпечує зручний та безпечний доступ до AWS-ресурсів для виконання навчальних завдань та практики.

Використання Python та Selenium для отримання токена авторизації шляхом емуляції аутентифікації користувача через веб-браузер може мати деякі проблеми та обмеження, які варто розглянути:

1) Залежність від веб-браузера. Selenium емулює дії користувача в веб-браузері, що може призвести до проблем із сумісністю або оновленнями браузера. При зміні версії браузера або платформи третьої сторони можуть виникнути проблеми з роботою скрипта.

2) Нестабільність та ненадійність. Процес емуляції аутентифікації користувача може бути менш стабільним і ненадійним, ніж використання офіційного API. Веб-інтерфейс платформи може змінюватися, що призведе до збою скрипта. Проте, через відсутність офіційного відкритого API ризики є очікуваними.

3) Безпека. Емуляція аутентифікації користувача може створювати ризики безпеки. Доступ до токенів авторизації має контролюватися.

4) Обмежена швидкість та масштабованість. Використання Python та Selenium для емуляції аутентифікації користувача може бути повільнішим, ніж використання API. Це може обмежити швидкість створення та видалення

тимчасових AWS-аккаунтів та ускладнити масштабування рішення для великої кількості користувачів.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕГРАЦІЇ НАВЧАЛЬНОЇ ПЛАТФОРМИ ІЗ СЕРВІСОМ НАДАННЯ ТИМЧАСОВИХ АКАУНТІВ

Для взаємодії з тимчасовими аккаунтами сервіс використовує точку доступу GraphQL, з якою взаємодіє клієнт (браузер), щоб отримувати інформацію про стан тимчасового акаунту, та змінювати цей стан, тобто створювати та видаляти тимчасові акаунти. GraphQL - це запитова мова та середовище виконання запитів для взаємодії з сервером, яке було розроблене командою Facebook у 2012 році та публічно опубліковано у 2015 році. GraphQL використовує типи даних та схему для опису даних, які можуть бути запитані та відповідей на запити. Це дозволяє клієнтам бути впевненими в тому, що дані, які вони отримують, будуть у форматі, який вони очікують. GraphQL використовується для створення API для веб-додатків, мобільних додатків та інших сервісів, які використовують різноманітні джерела даних. Він став популярним серед розробників з його здатністю допомагати у вирішенні проблем з масштабуванням та складністю запитів на сервер. [4]

Точка доступу GraphQL є однією для всіх користувачів сервісу, вона приймає схему запиту та авторизаційний токен у форматі JWT. JWT токен (JSON Web Token) - це стандартний формат токена, який може бути використаний для передачі структурованої інформації між сторонами. JWT токен складається з трьох частин: заголовку (header), тіла (payload) та підпису (signature). Таким чином, JWT токен містить усю необхідну інформацію про особистість запитувача, тому нема необхідності у використанні логіну на паролю при запитах. [5]

JWT токен можливо отримати лише після авторизації до веб додатку. Для автоматизації авторизації було використано розширення до Selenium під назвою Seleniumwire, що на додаток до звичайного Selenium дозволяє гнучко працювати з HTTP заголовками, в яких міститься необхідний токен.

5.1 Автоматизація отримання авторизаційного токену за допомогою Seleniumwire

Для взаємодії із браузерами під капотом використовується Selenium WebDriver API. Selenium WebDriver API - це набір програмних інтерфейсів додатків, які дозволяють взаємодіяти з браузерами через Selenium WebDriver. Цей API надає можливість програмно керувати веб-браузером, виконувати дії користувача та отримувати результати. Для того, щоб почати використовувати Selenium WebDriver треба його імпортувати у проект, я показано на рисунку 5.1. [6]

```
2
3   from seleniumwire import webdriver
4
```

Рисунок 5.1– імпорт Selenium WebDriver

Наступним кроком необхідно ініціалізувати вебдрайвер разом із опціями. Таким чином, буде відкрито сесію браузера із зазначеними опціями, з якої буде проведена автоматична авторизація у сторонній сервіс. На рисунку 5.2 представлена ініціалізація вебдрайверу з опціями. Першим чином створюємо об'єкт «op», який є екземпляром класу «ChromeOptions()». Цей об'єкт використовується для налаштування параметрів браузера Chrome.

```
op = webdriver.ChromeOptions()
op.add_argument('headless')
op.add_argument('no-sandbox')
op.add_argument('user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64)')
driver = webdriver.Chrome(options=op)
```

Рисунок 5.2 – ініціалізація вебдрайверу з опціями

Далі додаємо два параметри до об'єкту «op». Перший параметр headless налаштовує браузер на «безголовий» режим (headless mode), що означає, що він буде працювати без графічного інтерфейсу користувача. Другий параметр no-

sandbox додається, щоб уникнути помилок безпеки, які можуть вплинути на постійність скрипта.

Наступним чином додаємо третій параметр до об'єкту «op». Цей параметр, user-agent, налаштовує рядок агента користувача, який відправляється до сервера при кожному запиті. В даному випадку агент користувача встановлюється як "Mozilla/5.0 (Windows NT 10.0; Win64; x64)", що означає, що запити будуть виглядати так, ніби вони були відправлені з браузера Firefox встановленого на операційній системі Windows 10.

Нарешті, створюємо об'єкт "driver", який є екземпляром класу "webdriver.Chrome()". В конструкторі цього класу передається параметр "options=op", щоб налаштувати параметри браузера.

На рисунку 5.3 представлено процес авторизації, який складається з декількох кроків:

1) У першому рядку коду ми запускаємо браузер та відкриваємо сторінку авторизації сервісу. Оскільки процес не моментальний, перед виконанням наступних дій робимо паузу на 5 секунд для того, щоб усі елементи сторінки завантажилися.

2) У третьому та четвертому рядках передаємо у форму авторизації email адресу у якості імені користувача та пароль. На п’ятому рядку емулюємо натискання на кнопку Login та знов очікуємо 5 секунд для отримання відповіді з сервера.

3) На сьомому рядку звертаємось до адреси, з якої виконуються операції з тимчасовими аккаунтами. Це необхідно для того, щоб клієнт звернувся до точки доступу GraphQL з авторизаційним токеном, щоб далі ми змогли знайти його серед HTTP заголовків. Нарешті, знову даємо скрипту п’ять секунд на завантаження веб сторінки.

```
driver.get(AUTH_URL)
sleep(5)
driver.find_element("id","1-email").send_keys(username)
driver.find_element("id","1-password").send_keys(password)
driver.find_element("id","1-submit").click()
sleep(5)
driver.get(AWS_SANDBOX_URL)
sleep(5)
```

Рисунок 5.3 – Авторизація у сторонній сервіс

На рисунку 5.4 представлено частину коду, в якій здійснюється пошук авторизаційного токену. Усі HTTP запити фільтруються по адресі GraphQL, адже лише там використовується необхідний токен. У цих запитах знаходимо заголовок з назвою "Authorization", який повинен мати значення типу "Bearer Token_Value", оскільки нам потрібне лише значення токену (Token_Value), розбиваємо значення заголовку на 2 частини та записуємо другу частину у змінну під назвою "AUTH_TOKEN". Виклик функції отримання токену повертає саме це значення. Якщо токен не був знайдений, повертаємо помилку.

```
for request in driver.requests:
    if request.url == GRAPHQL_ENDPOINT:
        AUTH_HEAD = request.headers['Authorization'].split(" ", 1)
        if len(AUTH_HEAD) > 1:
            AUTH_TOKEN = AUTH_HEAD[1]
if AUTH_TOKEN:
    driver.close()
    return AUTH_TOKEN
else:
    driver.close()
    raise Exception(f"Failed to retrieve authorization token")
```

Рисунок 5.4 – Процес пошуку токена для авторизації

Надалі токен буде використовуватися для створення та видалення тимчасових облікових записів AWS через HTTP запити до адреси GraphQL. Для простих HTTP запитів була використана популярна бібліотека "requests".

5.2 Програмування API інтерфейсу мовою Python

Для можливості уніфікованої взаємодії з базою даних DynamoDB було необхідно створити API інтерфейс. Це надасть нам можливість за допомогою HTTP запитів, які можуть бути відправлені Python кодом з бібліотеки requests, або з самого Jenkins пайплайну мовою Groovy, отримувати та змінювати дані стосовно облікових записів сторонньої платформи.

На рисунку 5.5 зображена модель таблиці Accounts, що буде зберігати ім’я користувача та пароль користувача сторонньої платформи. Початково, ми можемо отримувати дані про окремих користувача за допомогою унікального ідентифікаційного номеру, що присвоюється до користувача при додаванні його у базу, але це не дуже зручно. Оскільки нам буде необхідно отримувати облікові дані лише вільних акаунтів, які не використовуються у момент запуска пайплайну, було додано додатковий індекс AvailabilityIndex за яким можна буде отримувати JSON об’єкт, що містить усі вільні акаунти, а саме акаунти, в яких атрибут isAvailable дорівнює true.

```python
from .base import BaseTable
from pynamodb.indexes import GlobalSecondaryIndex, AllProjection
from pynamodb.attributes import UnicodeAttribute

class AccountsAvailabilityIndex(GlobalSecondaryIndex):
    class Meta:
        index_name = "isAvailableIndex"
        projection = AllProjection()
        read_capacity_units = 1
        write_capacity_units = 1

    isAvailable = UnicodeAttribute(hash_key=True)

class Accounts(BaseTable):
    class Meta:
        table_name = "AccountPool"
        host = BaseTable.Meta.host
        region = BaseTable.Meta.region
        read_capacity_units = 1
        write_capacity_units = 1

    id = UnicodeAttribute(hash_key=True)
    username = UnicodeAttribute(null=True)
    password = UnicodeAttribute(null=True)
    isAvailable = UnicodeAttribute(null=True)

    AvailabilityIndex = AccountsAvailabilityIndex()
```

Рисунок 5.5 – Модель таблиці DynamoDB

На рисунку 5.6 зображено як описана поведінка FastAPI при отриманні запитів. Таким чином, маємо наступні запрограмовані дії для нашого API:

1) При відправленні HTTP запита методу Get на API з параметром id, який відповідає ідентифікаційному коду існуючого у базі акаунта, буде повернуто відповідь з усіма даними цього акакунта; якщо замість параметра id вказати isAvailable=true, то буде повернуто усі вільні акаунти.

2) Якщо відправити запит методу Post, то буде створено новий запис у таблиці з даними акаунта. Запит повинен мати усі дані відповідно до моделі, інакше API поверне помилку.

3) Якщо відправити запит методу Put вказавши при цьому параметр id, то можна змінити запис користувача у базі даних. Змінити можна ім’я користувача, пароль та атрибут isAvailable з false на true та навпаки. Це необхідно, щоб помічати акаунти як вільні та зайняті.

```python
router = APIRouter()

######### DESCRIBE #########
@router.get("", response_model=List[AccountOut])
async def get_accounts(querry: AccountInGet = Depends()) -> List[AccountOut]:
    try:
        if querry.id:
            return [AccountController.get_by_id(querry.id)]
        if querry.isAvailable:
            return AccountController.get_available(querry.isAvailable)
        return AccountController.get_all()
    except DoesNotExist as e:
        raise HTTPException(status_code=404)

######### CREATE #########
@router.post("", response_model=AccountOut, status_code=200)
async def create_account(item: AccountInCreate) -> AccountOut:
    return AccountController.create(item)


######### UPDATE #########
@router.put("", response_model=AccountOut, status_code=200)
async def update_account(item: AccountInUpdate) -> AccountOut:
    try:
        return AccountController.update(item)
    except DoesNotExist as e:
        raise HTTPException(status_code=404)
```

Рисунок 5.6 – Маршрути API

5.3 Збірка залежностей для Python проекту

Для встановлення заложностей для проекта було вирішено використовувати Pipenv, що являє собою інструмент для управління Python залежностями, який поєднує в собі можливості інсталятора pip та менеджера віртуальних середовищ Virtualenv. Pipenv дозволяє створювати ізольоване середовище для проекта та управляти залежностями, що використовуються в цьому проекті.

Одна з головних проблем, які вирішує Pipenv, - це уникнення конфліктів між різними проектами, які використовують одну й ту ж версію пакету. Якщо

встановити пакет глобально, то будь-який проект, який використовує цей пакет, може випадково оновити його до новішої версії, що може порушити роботу як інших проектів, що також використовують цей пакет, так і системи загалом, тому що велика частина операційних систем Linux залежить від Python модулів, що встановлені глобально. На рисунку 5.7 представлений зміст файлу Pipfile, з Pipenv отримує пакети та їхні версії, які необхідно встановити у віртуальне середовище Python.

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
selenium-wire = "*"
boto3 = "*"
requests = "*"

[requires]
python_version = "3.9"
```

Рисунок 5.7 – зміст файлу Pipfile

5.4 Контейнеризація Python проекту за допомогою Docker

Docker - це платформа для контейнеризації програмного забезпечення, яка дозволяє ізолювати програмне забезпечення від інших процесів в системі. Контейнеризація дозволяє розгорнути додаток на будь-якому середовищі, що підтримує Docker, без потреби налаштовувати його для кожного окремого середовища. [7]

Загалом, Docker разом з Pipenv допомагають розгортати та керувати Python-проект на будь-якому середовищі без потреби встановлювати та налаштовувати залежності та середовище вручну. Це робить процес розгортання та розробки програмного забезпечення більш простим та ефективним. Крім того,

використання Docker та Pipenv забезпечує консистентність середовищ виконання та залежностей між різними розробниками та серверами, що полегшує розробку та тестування програмного забезпечення.

На рисунку 5.8 зображений зміст файлу Dockerfile в якому встановлюється необхідне програмне забезпечення, таке як Chromedriver, що являє собою рушій браузера, утиліта Pipenv та інші допоміжні утиліти. Після цього, за допомогою Pipenv створюється віртуальне Python середовище зі збіркою усіх необхідних залежностей. Останніми двома рядками визначено які команди будуть виконані у контейнері при його запуску.

```
FROM python:3.9 as base

# Setup env
ENV LANG C.UTF-8
ENV LC_ALL C.UTF-8
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONFAULTHANDLER 1
WORKDIR /root

# Install required software
RUN apt update && apt install -y gcc chromium unzip

RUN wget -O /tmp/chromedriver.zip http://chromedriver.storage.googleapis.com/110.0.5481.77/chromedriver_linux64.zip && \
unzip /tmp/chromedriver.zip chromedriver -d /usr/local/bin/

RUN pip install pipenv

# Install application into container
COPY . .
RUN PIPENV_VENV_IN_PROJECT=1 pipenv install --deploy
ENV PATH=".venv/bin:$PATH"

# Run the application
ENTRYPOINT ["python"]
CMD ["src/main.py"]
```

Рисунок 5.8 – Dockerfile проекта

5.5 Оновлення Jenkins пайплайну для використання інтеграції

Для того, щоб дозволити студентам виконувати завдання у тимчасових акаунтах замість персональних, останнім кроком потрібно було внести зміни до пайплайну. З самого початку, пайплайн мав три великі етапи та обов’язковий прикінцевий етап:

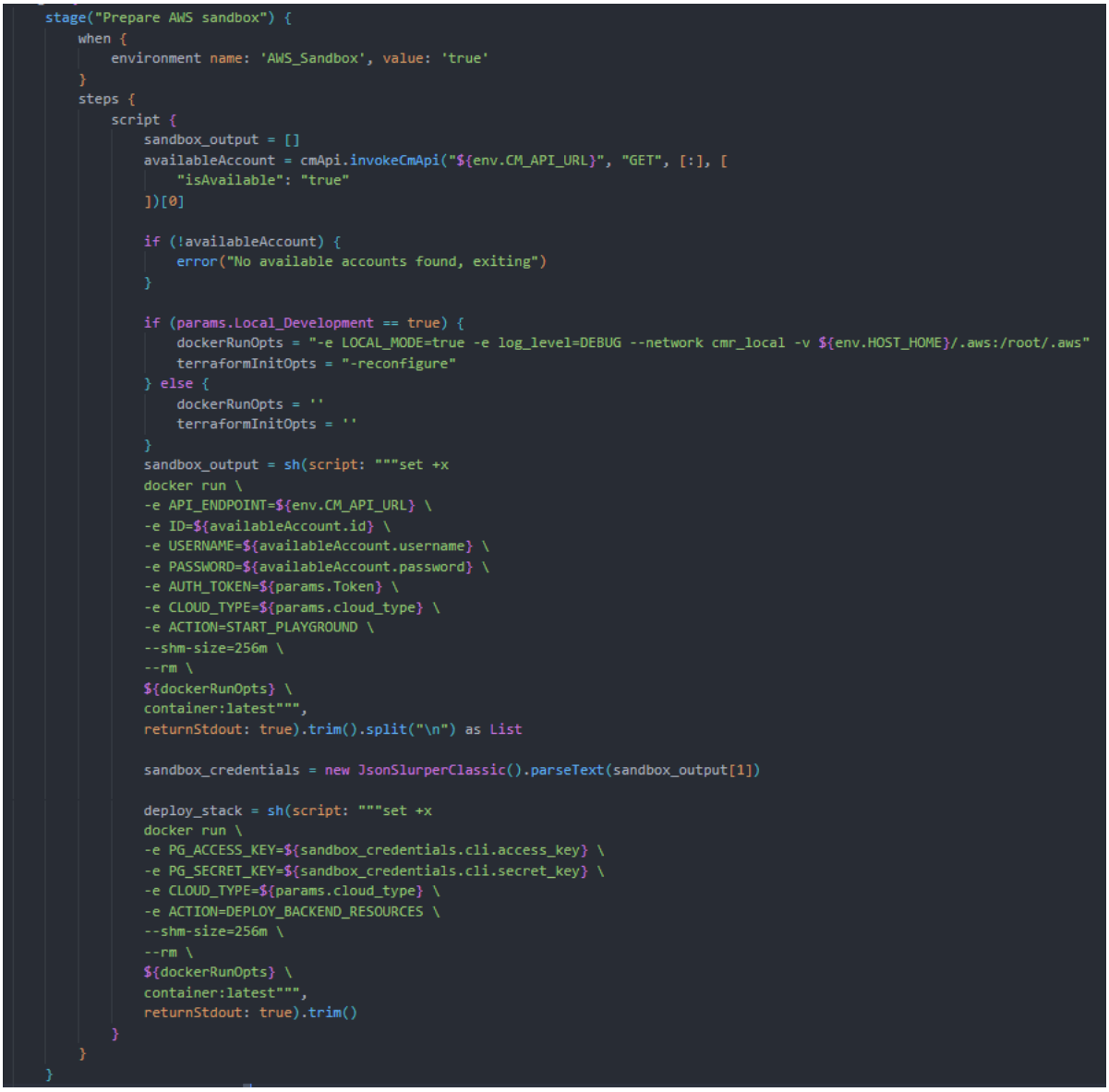
1) Initialization. На цьому етапі, пайплайн пропонує студенту обрати завдання для виконання, отримує необхідні дані з DynamoDB через FastAPI та ініціалізує багато інших допоміжних параметрів, зокрема для Terraform, для подальшого розгортання ресурсів.

2) Deploy Task Resources. На цьому етапі за допомогою Terraform розгортується інфраструктура, виконується скрипт, який видаляє певні ресурси та публікується HTML сторінка з описом завдання.

3) Validate Task. Це передостанній етап, який перевіряє виконання завдання та визначає оцінку, яку отримав студент.

4) Post actions. Так звані пост-дії не відносяться до послідовності попередніх етапів, оскільки код написаний у цій секції виконується завжди, як після успішного виконання останнього етапу, так і у разі непередбачуваних помилок які трапилися в одному з етапів. Саме у цій секції виконується видалення ресурсів. Це необхідно для того, щоб у разі помилки пайплайн не завершився не видаливши створені ресурси.

Щоб інтеграція запрацювала, було створено етап "Prepare AWS sandbox", зміст якого зображено на рисунку 5.9, перед етапом ініціалізації, метою якого було виконання нашого Docker контейнера з Python кодом та отримання даних нового тимчасовго акаунту.



```
stage("Prepare AWS sandbox") {
    when {
        environment name: 'AWS_Sandbox', value: 'true'
    }
    steps {
        script {
            sandbox_output = []
            availableAccount = cmApi.invokeCmApi("${env.CM_API_URL}", "GET", [:], [
                "isAvailable": "true"
            ])[0]

            if (!availableAccount) {
                error("No available accounts found, exiting")
            }

            if (params.Local_Development == true) {
                dockerRunOpts = "-e LOCAL_MODE=true -e log_level=DEBUG --network cmr_local -v ${env.HOST_HOME}/.aws:/root/.aws"
                terraformInitOpts = "-reconfigure"
            } else {
                dockerRunOpts = ''
                terraformInitOpts = ''
            }
            sandbox_output = sh(script: """set +x
            docker run \
            -e API_ENDPOINT=${env.CM_API_URL} \
            -e ID=${availableAccount.id} \
            -e USERNAME=${availableAccount.username} \
            -e PASSWORD=${availableAccount.password} \
            -e AUTH_TOKEN=${params.Token} \
            -e CLOUD_TYPE=${params.cloud_type} \
            -e ACTION=START_PLAYGROUND \
            --shm-size=256m \
            --rm \
            ${dockerRunOpts} \
            container:latest""",
            returnStdout: true).trim().split("\n") as List

            sandbox_credentials = new JsonSlurperClassic().parseText(sandbox_output[1])

            deploy_stack = sh(script: """set +x
            docker run \
            -e PG_ACCESS_KEY=${sandbox_credentials.cli.access_key} \
            -e PG_SECRET_KEY=${sandbox_credentials.cli.secret_key} \
            -e CLOUD_TYPE=${params.cloud_type} \
            -e ACTION=DEPLOY_BACKEND_RESOURCES \
            --shm-size=256m \
            --rm \
            ${dockerRunOpts} \
            container:latest""",
            returnStdout: true).trim()
        }
    }
}
```

Рисунок 5.9 – Етап створення сендбоксу у Jenkins

Ці дані використовуються на етапі ініціалізації (рис. 5.10) для того, щоб в подальшому Terraform створював ресурси у тимчасовому акаунті замість власного акаунту студента.

```
studentData = cmApi.invokeCmApi("${env.CM_API_URL}/students", "GET", [:], [
    "id": student_cm_id
])[0]
println("Printing sandbox_credentials during init stage")
println(sandbox_credentials)
env.student_aws_account_id = sandbox_credentials.account_id
echo "Using AWS region ${env.AWS_REGION}."
```

Рисунок 5.10 – Частину етапу ініціалізації

Також, дані з нового першого етапу використовуються на етапі створення ресурсів – публікується HTML сторінка (рис. 5.11) з описом завдання та обліковими даними тимчасового акаунту, щоб студент міг авторизуватися та виконати завдання.

```
if (params.cloud_type == "aws") {
    env.TF_VAR_cloud = params.cloud_type
    env.TF_VAR_login_url = sandbox_credentials.console.URI
    env.TF_VAR_username = sandbox_credentials.console.Username
    env.TF_VAR_password = sandbox_credentials.console.Password
    env.TF_VAR_accesskey = sandbox_credentials.cli.access_key
    env.TF_VAR_secretkey = sandbox_credentials.cli.secret_key
}

try {
    sh "terraform apply --auto-approve"
    // definition of the task in the form of a report
    publishHTML (target : [allowMissing: true,
        alwaysLinkToLastBuild: true,
        keepAll: true,
        reportDir: "../${env.taskDocs}",
        reportFiles: '*.html',
        reportName: 'Task definition',
        reportTitles: 'Task description'])
}
catch  (Exception){
    infra_deploy_failed = true
}
```

Рисунок 5.11 – Етап розгортання ресурсів та публікації HTML сторінки

Останні зміни було внесено до секції пост-дій. Тепер при завершені пайплайну, або при неочікуваній помилці у будь-якому з етапів ще раз запускається наш Docker контейнер, тільки з параметрами на видалення тимчасового акаунту.

```
if (availableAccount && sandbox_output.size() == 2) {

    echo "Going to destroy sandbox and release locked account"

    sandbox_destroy = sh(script: """set +x
        docker run \
        -e API_ENDPOINT=${env.CM_API_URL} \
        -e ID=${availableAccount.id} \
        -e USERNAME=${availableAccount.username} \
        -e PASSWORD=${availableAccount.password} \
        -e AUTH_TOKEN=${sandbox_output[0]} \
        -e ACTION=DESTROY_PLAYGROUND \
        --shm-size=256m \
        --rm \
        ${dockerRunOpts} \
        container:latest""",
        returnStdout: true).trim()
}
else if (availableAccount && !sandbox_output) {
    echo "Credentials were not generated, releasing account"
    account_unlock = cmApi.invokeCmApi("${env.CM_API_URL}", "PUT", [
        "id": availableAccount.id,
        "isAvailable": "true"
    ])
}
```

Рисунок 5.12 – Етап завершальних дій пайплайну

6 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ РІШЕННЯ ТА ОГЛЯД МОЖЛИВОСТЕЙ ДЛЯ ПОДАЛЬШОГО ВДОСКОНАЛЕННЯ.

6.1 Валідація та тестування рішення

Для перевірки працездатності інтеграції усі складові навчальної платформи були розвернуті локально за допомогою інструмента docker-compose. Docker Compose є інструментом для оркестрування контейнерів Docker, який дозволяє описувати та запускати складні додатки, що складаються з багатьох контейнерів. Замість того, щоб кожен контейнер запускати окремо, можна використовувати docker-compose.yml файл, щоб описати всі контейнери та їх налаштування, а потім запустити їх всі однією командою. Окрім того, Docker Compose дозволяє встановлювати залежності між контейнерами, налаштовувати мережі для комунікації контейнерів між собою та зберігати дані за допомогою томів. Це дозволяє легко розгортати багатоконтейнерні додатки на локальній машині для тестування або на серверах.

Після розгортання додатку був створений користувач Maksym Myshko, від імені якого був запущений пайплайн.

На рисунку 6.1 показаний знімок екрану з Jenkins та успішне проходження етапу підготовки тимчасового AWS-акаунту для подальшого використання у завданні замість власного акаунта студента.

Також на рисунку 6.1 показано, що виконання пайпалйну призупинилось на етапі ініціалізації "(paused for 19s)". Це означає, що пайплайн очікує вхідних дані від користувача.

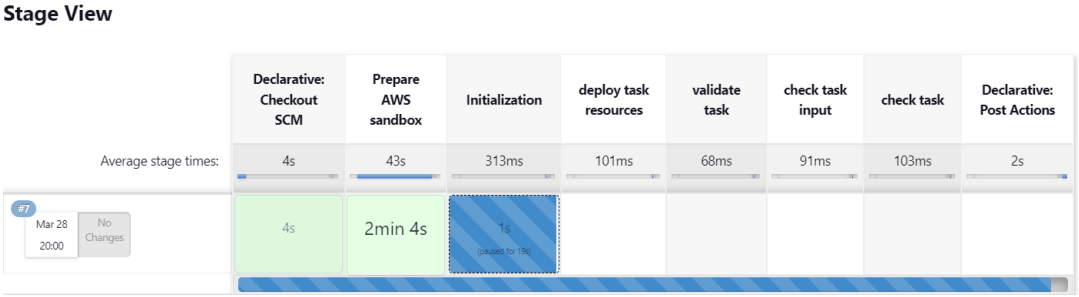


Рисунок 6.1 – зовнішній вид пайплайну у Jenkins

На рисунку 6.2 зображено як проходить вибір завдання студентом. Студент має можливість обрати будь-яке завдання зі списку та натиснути на кнопку Approve. Після цього пайплайн продовжує своє виконання та розгортає необхідні для проходження завдання ресурси у тимчасовому акаунті.

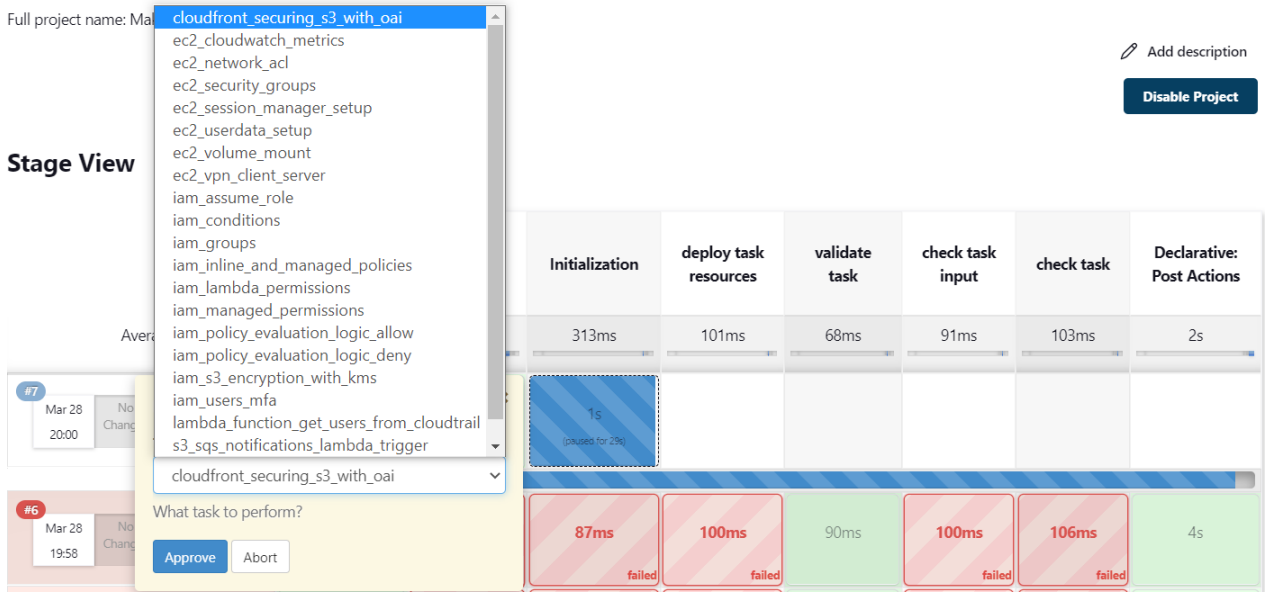


Рисунок 6.2 – Вибір завдання для проходження

Після того як усі необхідні ресурси були створені, публікується html сторінка із описом завдання та обліковими даними для входу в тимчасовий акаунт.

На рисунку 6.3 зображені облікові дані, які студенту необхідно використати для входу у тимчасовий акаунт. У додаток до імені користувача та паролю студент може використати пару ключів (access key та secret key) для виконання завдання за допомогою AWS CLI. AWS CLI – це інтерфейс командного рядка для взаємодії з Amazon Web Services (AWS). Він дозволяє користувачам керувати своїми послугами AWS через командний рядок з будь-якого комп'ютера з встановленим програмним забезпеченням AWS CLI і доступом до Інтернету.

Рисунок 6.3 – Облікові дані тимчасового AWS-акаунта

Після авторизації у правому верхньому куті можна побачити свого користувача та ідентифікаційний номер облікового запису AWS, як показано на рисунку 6.4.

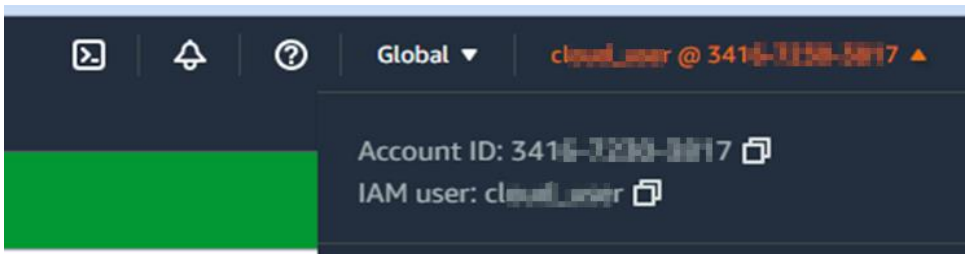


Рис 6.4 – Панель навігації AWS з ім’ям користувача та номером акаунта.

Наступним кроком необхідно виконати обране завдання протягом 3 годин. Якщо не впоратися вчасно, пайплайн завершиться з помилкою про тайм-аут без зберігання прогресу.

На рисунку 6.5 показано як студент повинен сповістити пайплайн про виконання завдання для автоматичної перевірки.

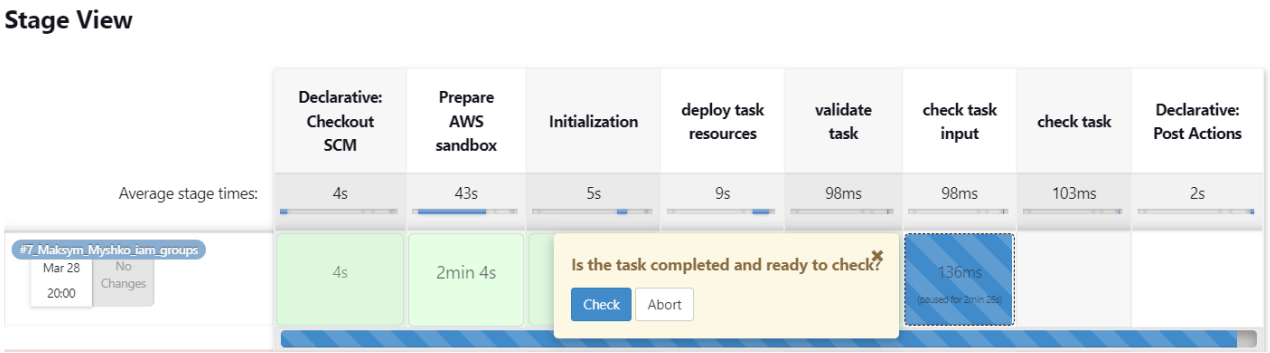


Рис. 6.5 – Запит автоматичної перевірки завдання.

Після успішної перевірки завдання пайплайн запитає студента заповнити коротку форму зворотного зв’язку (рис. 6.6). Це необхідно для того, щоб автори

завдань могли отримати оцінку завдання студентами. Отримані від користувачів відгуки опрацьовуються для подальшого вдосконалення поточного та майбутніх завдань.

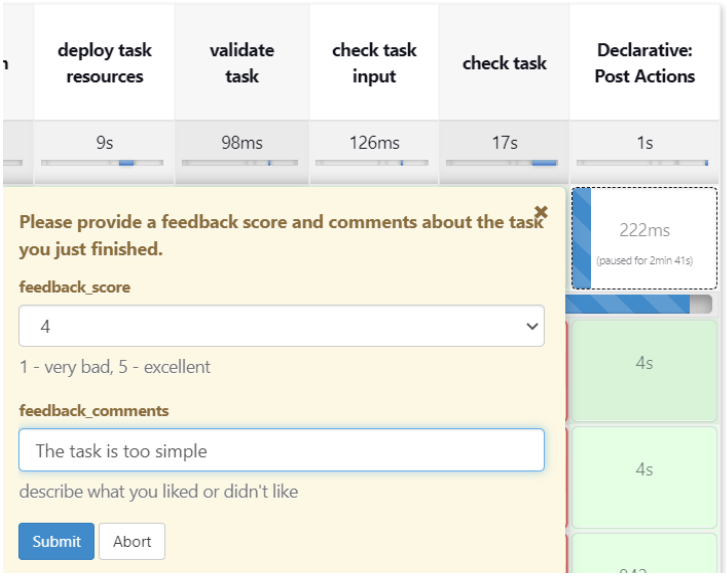


Рис. 6.6 – Запит зворотного зв’язку.

6.2 Перспективи для подальшого вдосконалення рішення

Тенденції на ринку провайдерів хмарних сервісів змінюються з часом. Хоча на сьогоднішній день AWS являється лідером, зростає популярність й інших відомих провайдерів: Microsoft Azure та GCP (рис. 6.7). Через це, навчальна платформа може бути розширена для підтримки інших провайдерів.

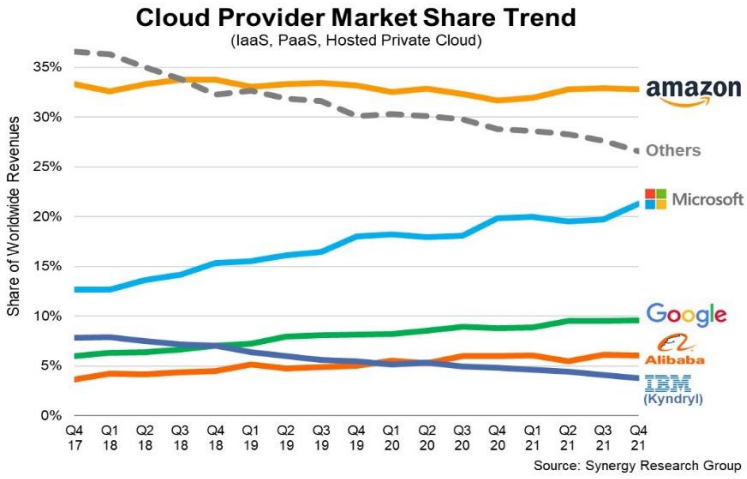


Рисунок 6.7 – Графік популярності хмарних провайдерів станом на кінець 2021 року

На сьогоднішній день вже існують сервіси, які надають тимчасовий доступ до Azure та GCP для ознайомлення з сервісами, що надають ці хмарні провайдери. Паралельно з розробкою завдань для студентів під нових провайдерів можна розширити проект інтеграції для підтримки цих провайдерів.

ВИСНОВОК

Під час наукової роботи було досліджено можливості розвитку навичок роботи з Amazon Web Services (AWS) для студентів. Було проведено огляд літератури з питань хмарних сервісів та порівняння провайдерів хмарних послуг. AWS було визнано одним з найпопулярніших та надійних провайдерів хмарних сервісів на ринку.

Також було проаналізовано існуючі рішення для розвитку навичок роботи з AWS та виявлено фінансові та безпекові виклики, з якими зіткнуться студенти, якщо вони використовують свої власні AWS-акаунти для навчання. Були розглянуті обмеження існуючих підходів.

Для розв'язання цієї проблеми була запропонована інтеграція навчальної платформи з платформою створення тимчасових аккаунтів AWS. Була розроблена програмна автоматизація аутентифікації та отримання токенів з використанням Python та Selenium у headless режимі. Отриманий токен надалі був використаний у програмі для створення та видалення тимчасових аккаунтів. Були внесені зміни до Jenkins пайплайну, що тепер запускає завдання на тимчасових акаунтах та до FastAPI для комунікації з базою DynamoDB.

Тестування та валідація рішення показали, що інтеграція між навчальною платформою та платформою створення тимчасових аккаунтів AWS забезпечує ефективне та безпечне навчання студентів. Впровадження такого рішення може покращити привабливість навчальної платформи та забезпечити економію коштів для студентів.

У майбутньому можна розширити функціональні можливості системи, додавши підтримку інших хмарних провайдерів.

Отже, можна зробити висновок, що розроблене рішення може забезпечити ефективне та безпечне навчання студентів з AWS, а також допомогти розробникам навчальних платформ покращити привабливість своїх продуктів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrews G. Foundations of multithreaded, parallel, and distributed programming / Gregory R Andrews., 2000.

2. Nikhil S. The Biggest AWS Users [Електронний ресурс] / Suryawanshi Nikhil – Режим доступу до ресурсу: https://www.linkedin.com/pulse/biggest-aws-users-nikhil-suryawanshi.

3. Gillis A. What is Terraform? [Електронний ресурс] / Alexander Gillis – Режим доступу до ресурсу: https://www.techtarget.com/searchitoperations/definition/Terraform.

4. Stemmler K. What is GraphQL? GraphQL introduction [Електронний ресурс] / Khalil Stemmler. – 2021. – Режим доступу до ресурсу: https://www.apollographql.com/blog/graphql/basics/what-is-graphql-introduction/.

5. JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://auth0.com/docs/secure/tokens/json-web-tokens.

6. Baskirt O. Selenium API and Architecture – All Details with Diagrams and MindMaps [Електронний ресурс] / Onur Baskirt. – 2021. – Режим доступу до ресурсу: https://www.swtestacademy.com/selenium-api-2/.

7. del Alba L. What Is Docker? [Електронний ресурс] / Lucero del Alba. – 2022. – Режим доступу до ресурсу: https://www.sitepoint.com/what-is-docker/.