Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

(повне найменування вищого навчального закладу)

Факультет *Інфокомунікацій*

(повне найменування інституту, факультету (відділення))

Кафедра *Інформаційно-мережної інженерії*

(повна назва кафедри (предметної, циклової комісії))

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Автоматизація перевірки інфраструктурного рішення в хмарі

(тема)

Виконав: студент 2 курсу групи *ІМІм-20-1*

Калиняк І.Д.

(прізвище та ініціали)

Спеціальність *172 Телекомунікації та радіотехніка*

(код і повна назва спеціальності)

Тип програми *освітньо-професійна*

(освітньо-професійна або освітньо-наукова)

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)

2021 р

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 та радіотехніка*

(код і повна назва)

Тип програми *освітньо-професійна*

(освітньо-професійна або освітньо-наукова)

Освітня програма *Інформаційно-мережна інженерія*

(повна назва)

ЗАТВЕРДЖУЮ

Зав. кафедри «ІМІ»

_____ проф. Безрук В.М.

(підпис)

«*08*» *листопада* 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Калиняку Ігорю Дмитровичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *Автоматизація перевірки інфраструктурного рішення в хмарі*
керівник роботи *к.т.н., доц. Костромицький Андрій Іванович*
затверджена наказом університету від «*08*» *листопада* 2021 р. № 1674Ст
2. Термін подання студентом роботи до екзаменаційної комісії *16 грудня* 2021 р.
3. Вихідні дані до роботи *Проаналізувати методи автоматизації перевірки інфраструктурного рішення для підготовки спеціалістів в області хмарних технологій. Розробити концепцію перевірки автоматичної завдань в публічній та приватній хмарі.*
4. Перелік питань, що потрібно опрацювати в роботі

Вступ

1. *Аналіз необхідності автоматизації перевірки інфраструктури*
2. *Проектування*
3. *Розробка і тестування автоматизованого рішення*

Висновки

5. Перелік графічного матеріалу із зазначенням креслень, схем, плакатів, комп'ютерних ілюстрацій *слайди презентації в форматі Power Point: 14*

слайдів (назва, мета кваліфікаційної роботи, результат опитування студентів, взаємодія студентів з автоматизованою системою, системи управління навчальним контентом, хмарні обчислення на прикладі Amazon Web Services, концепція «Infrastructure as a Code», методи безперервної доставки CI/CD, алгоритм перевірки автоматизованого рішення, інтеграція автоматизованої системи з LCMS, приклад інфраструктури практичного завдання, перевірка працездатності рішення, висновки)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз завдання та літературних джерел	08.11.2021 – 30.11.2021	
2	Створення наукової публікації	09.11.2021 – 11.11.2021	
3	Виконання розділу 1	11.11.2021 – 20.11.2021	
4	Виконання розділу 2	20.11.2021 – 25.11.2021	
5	Виконання розділу 3	25.11.2021 – 02.12.2021	
6	Оформлення пояснювальної записки	23.11.2021 – 04.12.2021	
7	Оформлення презентаційного матеріалу,	10.12.2021 – 12.12.2021	
8	Підготовка до захисту у ЕК	12.12.2021 – 15.12.2021	

Дата видачі завдання *08.11.2021* р.

Студент _____ *Калиняк І.Д.*
(підпис) (прізвище, ініціали)

Керівник роботи _____ *доц. Костромицький А. І.*
(підпис) (посада, прізвище , ініціали)

РЕФЕРАТ

Пояснювальна записка: 82 с., 28 рис., 11 джерел, 5 додатків.

Об’єкт роботи – хмарні технології

Мета роботи – Розробити автоматизовану систему перевірки інфраструктурного рішення в хмарі

Проаналізовано методи автоматизації перевірки інфраструктурного рішення для підготовки спеціалістів в області хмарних технологій. Розроблено прототип та концепцію автоматизованої перевірки завдань у хмарному провайдері AWS за допомогою інструментів Jenkins та Terraform.

ХМАРНІ ТЕХНОЛОГІЇ, IAC, AWS, TERRAFORM, JENKINS.

ABSTRACT

Explanatory slip: 82 p., 28 fig., 11 sources, 5 app.

The object of research – cloud technologies.

The purpose of work – Develop an automated system for testing infrastructure solutions in the cloud

Methods of automation of infrastructure solution verification for training of specialists in the field of cloud technologies are analyzed. A prototype and concept for automated task verification in the AWS cloud provider has been developed using Jenkins and Terraform tools.

CLOUD TECHNOLOGIES, INFRASTRUCTURE-AS-A-CODE, AWS, TERRAFORM, JENKINS.

ПЕРЕЛІК ТЕРМІНІВ І СКОРОЧЕНЬ

Cloud Mentor – назва автоматизованої системи перевірки завдань.

DevOps – development и operations.

IaaS (Infrastructure-as-a-Service) – інфраструктура як послуга.

IaC (Infrastructure-as-a-Code) – інфраструктура як код.

Terraform – інструмент для написання інфраструктурного коду.

Cloud Computing – хмарні обчислення.

AWS – Amazon Web Services, один з найбільших хмарних провайдерів.

CI (Continuous Integration) – безперервна інтеграція

ВСТУП

Революція хмарних обчислень за останнє десятиріччя принесла за собою масштабну трансформацію різних видів бізнесу та інфраструктури різних проектів. Все більше і більше компаній залежать від хмарних сервісів і центрів обробки даних для щоденного використання. Це зробило життя легшим і зручнішим для споживачів, навіть тих, хто не розуміє, який вплив справили хмарні технології на нього. До того як заявились хмарні обчислення, підприємствам часто доводилося стикатися з проблемами та витратами на запуск власної локальної IT-інфраструктури, а також на покриття витрат на внутрішній IT-персонал.

Актуальність комплексного навчання хмарним обчисленням на сьогодняшній день є дуже важлива для сучасного інженера та кваліфікованого спеціаліста, який спеціалізується на розробці та підтримці інфраструктури. Зумовлений спрос на досвідчених спеціалістів є причиною для створення систем для більш детального вивчення хмарних технологій на практичних прикладах.

В кваліфікаційній роботі був проведен аналіз необхідності використання подібних практик у навчальному процесі та створено концепцію повноцінної системи автоматизації перевірки інфраструктурного рішення в хмарних технологіях, яка может бути використана як для студентів ВНЗ, так я для спеціалістів, що прагнуть покращити свої навики володіння хмарними технологіями.

1 АНАЛІЗ НЕОБХІДНОСТІ АВТОМАТИЗАЦІЇ ПЕРЕВІРКИ ІНФРАСТРУКТУРИ В ХМАРІ

1.1 Інтенсифікація навчання з хмарним технологіями

Сучасний розвиток організації навчальних процесів дедалі більше спонукає переходу повноцінного навчання до онлайн форми. Результатом цих розвитку є постійне збільшення кількості студентів завдяки розповсюдженню більш зручної форми навчання серед різних форм навчання на лекціях, курсах і т.п. Всесвітня пандемія також внесла несподівані корективи і змусила всіх терміново опановувати цифрові інструменти й нові педагогічні підходи для створення підґрунтя переходу більшості практичних завдань на дистанційну основу. Відповідно до цього слід зазначити, що інтенсифікація навчального процесу через автоматизацію є вкрай важливим аспектом для перевірки розв’язання завдань дистанційного навчання.

Інтенсифікація процесу навчання являє собою совокупність внутрішніх та зовнішніх факторів, які впливають на успішність досягнення цілей навчання. Інтенсификация, насамперед, характеризується передачею великого обсягу інформації при незмінному продовженні навчання без зниження вимог до качеству знань та підвищенням продуктивності праці викладачів та студентів у кожній одиниці часу. Саме тому, побудова процессу навчання хмарним технологіям потребує більш децентралізованого підходу, отже також нових засобів перевірки та валідації завдань.

Існує величезна кількість платформ для створення онлайн-курсів, оболонок для онлайн-тестів та вікторин, сервісів для створення онлайн-презентацій, інтерактивних відео, документів для спільного користування тощо. Але опанування кожного ресурсу вимагає від викладача курсу часу та певного рівня комп’ютерної грамотності, щоб вивчити усі швидкоплинні методи роботи з хмарними технологіями. Використання таких ресурсів дає можливість осучаснити навчальний процес та винести його за межі офлайн

режиму. Включення в офлайн навчання елементів онлайн навчання називають змішаним навчанням. Саме змішане навчання зараз активно впроваджується в багатьох країнах, оскільки воно дає можливість інтенсифікувати навчальний процес та розвинути самостійність студентів.

Рисунок 1.1 – Схема навчального процесу

У процесі інтенсифікації процесу навчання хмарним технологіям на основі використання різних форм навчання формуються наступні важливі задачі:

- формування у студентів готовності до самостійної роботи, розвиток їх інформаційної компетентності;
- забезпечення ефективного зворотнього зв'язку на основі поєднання тестового комп'ютерного контролю знань з рішенням і представленням за допомогою інформаційних технологій узагальнених задач;
- більш успішна реалізація принципу професійної спрямованості студента;

1.2 Аналіз необхідності практичних занять з хмарних технологій

Метою дослідження однієї з тем кваліфікаційної роботи було визначення ефективності практичного симуляційного підходу (віртуальні лабораторії) проти теоретичного підходу до викладання та вивчення навичок роботи з інфраструктурними рішеннями у вищих навчальних закладах та методів перевірки завдань у хмарних середовищах. Зокрема, дослідження мало відповісти на такі дослідницькі питання:

- Яке позиція студентів та викладачів щодо викладання та вивчення хмарних технологій з використанням лабораторного та теоретичного підходів вивчення матеріалів?
- Якою мірою теоретичні та практичні лабораторні підходи

використовуються при викладанні та навчанні хмарним технологіям?

- Які переваги використання практичних лабораторних занять існують перед теоретичним викладанням матеріалу?

З аналізу даних, отриманих від викладачів, було виявлено, що теоретичний підхід використовується більшою мірою в порівнянні з практичними лабораторними роботами (Virtual Hands-On labs) у вищих навчальних закладах. Крім того, студенти вважали корисними як моделювання ситуації у режимі реального часу, так і практичні підходи методи навчання для покращення викладання та вивчення навичок роботи з хмарними технологіями. Незважаючи на те, що теоретичний та демонстраційний підхід дозволяє студентам набути базове підґрунтя для роботи з хмарними провайдерами, за результатами цього дослідження, практичний лабораторний підхід є більш ефективним для передачі практичних навичок і навичок вирішення проблем, знань і компетентності студентів порівняно з імітаційним підходом. Студенти віддають перевагу використанню практичного лабораторного підходу, а не демонстраційного теоретичного підходу з різних причин, зокрема: практичний підхід дає змогу студентам набути практичних або конкретних навичок та навичок розв’язування проблем, послідовне вирішення та розв’язання завдання.

Рисунок 1.2 – Результат опитування студентів ISSA Virtual Labs

Як свідчать результати дослідження ISSA (рис 1.2), хмарні практичні лабораторні роботи забезпечують ефективний практичний досвід навчання. Опитування студентів показує, що ISSA Virtual Hands-on Labs надали їм реалістичний досвід, який ефективно підготував їх до завдань, з якими стикаються спеціалісти по роботі з хмарними технологіям та змогу отримати достатньо практичних навичок та досвід по відношенню до того, що викладається в аудиторії на лекціях. Студенти відмитили, що практичні

завдання прості у використанні та мають покрокову інструкцію, як успішно завершити лабораторні завдання, а також є можливість отримати доступ до цих лабораторних завдань у навчальному закладі або вдома. Зважаючи на результат опитування, більш ніж 79 відсотків опитаних згодні з твердженням, що пропонований метод найкращим чином сприяє підготовці студента до робочих завдань на проектах та систематизує усі знання для того, щоб продовжувати успішно виконувати завдання у галузі хмарних обчислень.

Практичний лабораторний підхід можна розглядати як найбільш ефективний підхід для надання студентам практичних навичок та навичок вирішення проблем, знань та компетенції порівняно з теоретичним підходом.

1.3 Системи управління навчальним контентом

Система управління навчальним контентом (або Learning content management system) характеризує собою платформу, що виконує інтеграцію доставки, публікації та аналіз контенту в середовище для користувачів. Це платформа, яка дозволяє створювати, керувати, розміщувати та відстежувати цифровий навчальний контент. Основною функцією LCMS є адміністрування, відстеження та зберігання навчального матеріалу, що виключно зберігається всередині системи управління даними. Система забезпечує механізми для навчання фахівців, студентів та експертів з предметних галузей, за її допомогою можливе створення навчальних заходів (практичних, лабораторних робіт, семінарів), а також з'являється можливість для співпраці та ефективнішого використання навчальних матеріалів різними фахівцями з різних географічних місць.

Існує деякий перехід між системами управління навчальним контентом (LCMS) і системами управління навчанням (LMS). Обидві дозволяють розміщувати та надавати цифрове навчання, і обидва сумісні з SCORM (набір стандартів та специфікацій, розроблений для систем дистанційного навчання) Але між ними є важливі відмінності, такі як, тип навчання, процес створення

контенту, функції адміністрування. Системи керування навчальним контентом (LCMS) спеціалізуються на цифровому навчальному контенті, тоді як LMS – це платформи, які дозволяють керувати більш широким спектром навчальних досвідом. Одним з них є електронне навчання, але LMS також керує більш традиційними формами навчання та такими, як планування особистих семінарів, сприяння розмовам студентів через соціальні форуми та обмін ресурсами. Серед існуючих LMS є такі як LearnUpon , Docebo або Cornerstone, які дозволяють об'єднати ці різні види навчання в змішаний шлях навчання, який сприяє найбільш ефективної інтеграції студентів в онлайн навчанні. Інші LMS йдуть ширше, додаючи до них управління персоналом – хоча це не є універсальною особливістю LMS.

Рисунок 1.3 – Етапи роботи системи управління навчанням (LMS)

Однією з найважливіших особливостей LCMS є те, що все можна зробити лише з одного централізованого порталу. Це допомагає спростити процеси навчальної практики для навчального закладу. Викладачі та студенти зможуть отримати доступ до всього, що їм потрібно, тільки за допомогою єдиного програмного забезпечення, а оскільки воно засноване на хмарі, вони зможуть це робити з навчального закладу, вдома, у будь-якій точці світу, де є інтернет.

Системи управління навчанням включають у себе різний набір інструментів для інтеграції контента у різних формах навчання, як практичні задвання, тести, відеоматеріали. Усе це сприяє утворенню комплексному навчальному процесу, який може стати повноцінною альтернативою традиційному навчанню, враховуючи попит на розвиток віддаленного навчання. На прикладі порталу Udemy розглянемо одну з найбільш успішних систем управління навчанням та постачальник відкритих онлайн-курсів та Hands-On Lab. У Udemy понад 35 мільйонів користувачів, які навчаються новим навичкам у 57 000 експертів-інструкторів, які створили понад 130 000

онлайн-курсів. Навчальні посібники охоплюють студентів з більш ніж 180 країн більш ніж 65 мовами, а тисячі корпоративних клієнтів використовують Udemy для навчання співробітників. Компанія заснувана Ереном Балі, Октаєм Кагларом і Гаганом Біяні в 2010 році. Штаб-квартира компанії знаходиться в Сан-Франциско, офіси в Денвері, Бразилії, Індії, Ірландії та Туреччині.

Рисунок 1.4 – Порівняння платфоми Udemy з іншими LMS

Udemy — це платформа, яка дозволяє викладачам створювати онлайн-курси за різними темами. Використовуючи інструменти розробки курсів Udemy, вони можуть завантажувати відео, презентації PowerPoint, PDF-файли, аудіо, ZIP-файли та живі заняття для створення курсів. Викладачі також можуть спілкуватися та взаємодіяти з користувачами через онлайн-дошки обговорень.

Рисунок 1.5 – Приклад інтерфейсу видеоматеріалу на Udemy

Курси на Udemy пропонуються в різних категоріях, включаючи бізнес і підприємництво, науковців, мистецтво, здоров’я та фітнес, мову, музику та технології. Більшість занять проводяться з практичних предметів, таких як програмне забезпечення Excel або хмарні технології. Udemy також пропонує Udemy for Business, що надає підприємствам доступ до цільового набору з понад 7000 навчальних курсів на теми від тактики цифрового маркетингу до продуктивності офісу, дизайну, управління, програмування тощо. За допомогою Udemy for Business організації також можуть створювати власні навчальні портали для корпоративного навчання. Курси Udemy можуть бути платними або безкоштовними, залежно від викладача.

Рисунок 1.6 – Приклад сторінки переходу до Hands-On заняття

На рисунку 1.6 відображена сторінка доступу студента до практичного заняття курсу хмарних технологій. Система навчального перенаправляє студента до порталу з практичними заняттям и отримає результат після проходження завдання. У випадку успішного виконання студентом практики, результат буде відображен на головній сторінці та наступний модуль буде розблокован для виконання. Портал практичного завадння інтегрован з навчальним аккаунтом у хмарному провайдері (залежно від курсу) та створюється окремо для кожного студента платформи. Практичне рішення включає в себе покрокове створення або модифікацію хмарних ресурсів.

1.4 Методи автоматизації перевірки інфраструктури в хмарі

Метою проведених досліджень протягом виконання кваліфікаційної роботи є винаходження методу побудови повноцінної автоматизації перевірки завдань інфраструктурного рішення в хмарі. Метод перевірки та валідації завдання повинен відповідати наступним критеріям:

- опис інфраструктурної автоматизації відпоіває концепції IaC, є можливість його оновлення та версіювання.
- підтримка ідемпотентної конфігурації інфраструктури, яка гарантує що наступне застосування призводить до 0 змін відносно базової
- можливість виконати запуск плану перед застосуванням коду на реальну інфраструктуру. Це дає змогу побачити які зміни будуть застосовані відносно вже існуючих ресурсів
- можливість валідації покрокового розв’язання завдання студентом для відстеження прогресу у завданні
- метод повинен мати основу для подальшої інтеграції алгоритму с LMS або LMCS

Рисунок 1.7 – Принцип концепції Infrastructure as a Code

Пропонована на основі порівняння базисного та модифікованого стану

хмарної інфраструктури після виконання студентом завдання по створенню певних ресурсів у хмарному середовищі. Даний метод являє собою підрахунок різниці стану середовища між попередньо розгорнутими хмарними ресурсами для завдання та створеними студентами власноруч. Безпосередньо, метод перевірки складається з таких кроків, як запуск автоматизованого розгортання базисної інфраструктури завдання, розв’язання студентом завдання у хмарному середовищі та перевірки стана інфраструктури або шляху вирішення завдання студентом за ключовими етапами.

Рисунок 1.8 – Схема взаємодії з автоматизованою системою

Серед переваг вищезгаданого методу автоматизації треба відмітити ріст швидкості перевірки завдань, універсальність підходу для створення методології автоматизації та зменшення ролі людського фактору під час перевірки завдання.

1.5 Інструменти реалізації автоматизованого рішення

1.5.1 Хмарний провайдер Amazon Web Services

Згідно опитування компанії Synergy Research, за останні 2 роки у галузі провайдерів публічних хмар спостерігається значний імпульс, оскільки організації, які використовують хмарні сервіси, збільшують витрати на хмару через пандемію COVID-19 та карантинні обмеження. Хмарні обчислення

кардинально змінили світ за останнє десятиріччя, оскільки локальна архітектура почала мігрувати в хмару. У той же час для вирішення бізнес-потреб з'являються й інші постачальники хмарних послуг. Але, безперечно, основні три клауд провайдера – AWS, Azure та GCP, сприяють зросту попиту на ринку, утримуючи свої позиції незмінними протягом останнніх 6 років.

Рисунок 1.9 - Порівняння провайдерів в області хмарних обчислень

Хмарні обчислення – це доступ за запитом через Інтернет до обчислювальних ресурсів – програм, серверів (фізичних серверів і віртуальних серверів), сховищ даних, інструментів розробки, мережевих можливостей тощо – розміщених у віддаленому центрі обробки даних, яким керують хмарні сервіси. Термін «хмарні обчислення» характеризує забезпечення можливості віддаленого доступу до обчислювальних ресурсів, що передбачає надання різних видів послуг.

Рисунок 1.10 – Методи використання хмарних обчислень

Порівняно з традиційними локальними обчисленнями та залежно від вибраних хмарних служб, переваги хмарних обчислень наступні:

- Нижчі витрати на IT: Хмара дозволяє звільнити частину або більшість витрат і зусиль, пов'язаних із придбанням, встановленням, налаштуванням та керуванням власною локальною інфраструктурою.
- Підвищення гнучкості та економія часу: Завдяки хмарі проектна організація може почати використовувати корпоративні програми значно швидше, замість того, щоб чекати тижнями чи місяцями, поки IT відповість на запит, придбає та налаштує допоміжне обладнання та встановить програмне забезпечення. Хмара також дає змогу певним користувачам, зокрема розробникам і науковцям з даних, допомагати собі програмним забезпеченням та інфраструктурою

підтримки.

- Рентабельне масштабування інфрастуктури: Cloud забезпечує еластичність — замість того, щоб купувати зайву ємність, яка залишається невикористаною протягом повільних періодів, хмарні провайдери дають можливість збільшувати та зменшувати кількість обчислювальних ресурсів у відповідь на стрибки та спади трафіку. Також є можливість скористатися перевагами глобальної мережі постачальника хмарних послуг у різних регіонах, щоб поширити свої програми ближче до користувачів по всьому світу.

Термін «хмарні обчислення» також відноситься до технології, яка забезпечує роботу хмари. Це включає в себе певну форму віртуалізованої IT-інфраструктури — сервери, програмне забезпечення операційної системи, мережу та іншу інфраструктуру, яка абстрагується за допомогою спеціального програмного забезпечення, щоб її можна було об’єднати та розділити незалежно від фізичних кордонів обладнання. Наприклад, один апаратний сервер можна розділити на кілька віртуальних серверів. Віртуалізація дозволяє хмарним провайдерам максимально використовувати ресурси своїх центрів обробки даних. Згідно загальносвітової тенденції, більшість світових корпорацій прийняли модель хмарної доставки для своєї локальної інфраструктури, щоб вони могли реалізувати максимальне використання та заощадити витрати в порівнянні з традиційною IT-інфраструктурою і запропонувати кінцевим користувачам таке ж самообслуговування та гнучкість.

Хмара може бути приватною або загальнодоступною. Публічна хмара модже оказувати послуги будь-кому кінцевому користувачу в мережі, натомість, приватна хмара — це власна мережа або центр обробки даних, який надає розміщені послуги обмеженій кількості користувачів з певними налаштуваннями доступу та дозволів.

AWS (Amazon Web Services) — це комплексна платформа хмарних обчислень, що розвивається, надана корпорацією Amazon. AWS було

запущено в 2006 році з внутрішньої інфраструктури компанії, створеної Amazon.com для роботи з роздрібними онлайн-операціями. AWS була однією з перших компаній, яка представила модель хмарних обчислень з оплатою по мірі використання, яка масштабується, щоб забезпечити користувачам обчислення, сховище данних або пропускну здатність за потреби. AWS пропонує багато різних інструментів і рішень для підприємств і розробників програмного забезпечення, які можна використовувати в центрах обробки даних у 190 країнах. Служби AWS можуть використовувати такі групи, як державні установи, навчальні заклади, некомерційні та приватні організації.

Однією з ключових концепцій Amazon Web Services є побудова резервування та максимальне підвищення надійності для усієї інфраструктури, від регіонів до мережевих посилань для балансування навантажень до маршрутизаторів і баз данних. Для надання хмарних додатків і послуг клієнти надають і підключають своїх кінцевих користувачів і організаційні середовища до таких компонентів глобальної інфраструктури AWS:

- Регіони (AWS Regions)
- Зони доступності (AWS Availability Zones)
- Локальні зони (AWS Local Zones)
- Зони довжини хвилі (AWS Wavelength Zones)

Amazon Web Services надає послуги з десятків центрів обробки даних, розподілених у зонах доступності (AZ) у регіонах по всьому світу. AZ — це місце, яке містить кілька фізичних центрів обробки даних, група з одного або кількох дискретних центрів обробки даних, які надають програми та послуги в регіоні AWS. Кожна AZ містить резервні можливості підключення, живлення та мережі, а окремі AZ фізично відокремлені (ізольовані) одна від одної значною відстанню. Усі AZ в регіоні AWS підключаються через мережеві канали з низькою затримкою та високою пропускною здатністю. Завдяки можливості підключення та надлишковості AZ надають клієнтські

додатки та операційні середовища баз даних, які є більш масштабованими та відмовостійкими. Оскільки регіональні AZ фізично ізольовані одна від одної, програми можна розділити між кількома AZ для високої доступності.

Рисунок 1.11 - Зв’язок між регіонами та зонами доступності

Регіон – це сукупність АЗ у географічній близькості, з’єднаних мережевими зв’язками з низькою затримкою, де Amazon об’єднує центри обробки даних для надання додатків і послуг у зонах доступності AWS. Регіони також надають розширення для інших варіантів доставки, таких як локальні зони AWS. Кожен регіон AWS може запропонувати різну якість обслуговування з точки зору затримки, портфелю рішень та вартості залежно від його географічного розташування та відстані від сайтів клієнтів. Локальні зони AWS забезпечені для запуску високошвидкісних програм, таких як медіа, розваги, ігри в реальному часі, потокове відео в прямому ефірі та машинне навчання, які потребують однозначної мілісекундної затримки для обслуговування користувачів у певних географічних місцях.

AWS Wavelength дозволяє розробникам створювати програми, які забезпечують наднизькі затримки мобільним пристроям і кінцевим користувачам. Wavelength розгортає стандартні послуги обчислень і зберігання даних AWS на межі мереж 5G телекомунікаційних операторів. Розробники можуть розширити віртуальну приватну хмару (VPC) на одну або кілька зон довжини хвилі, а потім використовувати ресурси AWS, як-от екземпляри Amazon EC2, для запуску програм, які потребують надзвичайно низької затримки та підключення до служб AWS у регіоні. Зона довжини хвилі – це ізольована зона в місці оператора, де розгорнута інфраструктура довжини хвилі. Зони довжини хвилі прив’язані до регіону. Зона довжини хвилі є логічним розширенням регіону, і керується площиною управління в регіоні.

Рисунок 1.12 – Схема розташування регіонів AWS

AWS поділяється на різні сервіси; кожен може бути налаштований різними способами залежно від потреб користувача. Користувачі повинні мати можливість бачити параметри конфігурації та окремі карти серверів для служби AWS. Сервіси AWS можуть запропонувати такі інструменти організації, як обчислювальна потужність, сховище бази даних і послуги доставки вмісту. Понад 100 сервісів включає в себе портфоліо Amazon Web Services, зокрема для обчислень, баз даних, управління інфраструктурою, розробки додатків та безпеки. Ці послуги за категоріями включають:

- Compute services – сервіси, що надають користувачу надають -обчислювальні потужності (віртуальні машини або функції). Найбільш поширені такі сервіси, як EC2 (віртуальні машини у хмарі), EKS (Elastic Container Service for Kubernetes), AWS Lambda – інстумент для використання безсерверних обчислень (serverless)

- Storage services – сервіси, що представлені для зберігання інформації та баз данних. Найпоширеніші з них S3 (Simple Storage Service) – сервіс для зберігання будь-яких об'єктів навіть великого обсягу даних, EBS (Elastic Block Store) - сервіс для зберігання даних у блоковому вигляді для використання у EC2 інстансах, EFS (Elastic File Sytem) – еластична файлова система для монтування між екземплярами EC2 на основі Linux, Amazon DynamoDB – швидка та надійна база даних формата NoSQL, RDS (Relation Databse Service) – реляційна база даних

- Networking services – сервиси, що відповідають за створення та використання мережевого з’єданння між віртуальними інстансами EC2 або мережеву коммунікацію з фізичними локальними мережами, такі як VPC (Virtual Private Cloud) – сервіс, що дає повний контроль над віртуальним мережевим середовищем, включаючи розподіл ресурсів,

підключення та безпеку, AWS WAF (Web Application Firewall) - брандмауер для інтернет-застосунків, який дозволяє захистити їх (або різні API) від поширених мережевих експлойтів.

- Security services – сервіси, що забезпечують безпеку користування при роботі с іншими хмарними сервісами Amazon, як авторизація, відокремлення ролей, політики доступу. Найпоширеніші з них є IAM (Identity and Access Management), що допомагає керувати користувачами,призначати політики, формувати групи для керування кількома користувачами і обмежувати доступ до ресурсів та KMS (Key Management Service) – служба безпеки, що відповідає за створення та керування ключами шифрування, які дозволяють забезпечити повне шифрування данних.

- Monitoring service – сервіси спостереження та моніторінгу створених обчислювальних ресурсів, веб-порталів, додатків, що реагування на загальносистемні зміни продуктивності, оптимізації використання ресурсів та отримання єдиного уявлення про працездатність. Найбільш популярні з них є AWS CloudWatch (сервіс збирає моніторингові та оперативні дані у вигляді журналів, показників і подій, надаючи вам єдине уявлення про ресурси, програми та служби AWS, які працюють на AWS і локальних серверах) та AWS CloudTrail (сервіс, який використовується для відстеження подій та автоматично записує журнали подій і активності інших сервісів і зберігає дані в AWS S3)

Рисунок 1.13 – Найбільш популярні веб сервіси Amazon за 2021 рік

Враховуючи усі вищезгадані фактори, хмарний провайдер AWS був обран як основне хмарне рішення в рамках кваліфікаційної роботи.

1.5.2 Інструменти валідації коду хмарної інфраструктури

Відповідно до вибраного методу автоматизації перевірки студентських завдань у вигляді інфраструктурного рішення в хмарі одним з важливих пунктом є розробка та підтримка інфраструктурного коду для порівняння базової конфігурації з тією, що створить студент в процессі виконання свого завдання. Саме тому за допомогою інструментів валідації інфраструктурного коду для цього методу потрібно розробити алгоритм перевірки різниці між створенними та існуючими хмарними ресурсами.

Для створення прототипу системи автоматизиваної первірки навчального процесу у розділі хмарних технологій важливо дотримуватися концепції IaC (Infrastructure as a Code). Підхід «Інфраструктура-як-код» описує ідею використання мови програмування високого рівня для керування IT-системами або будь-якими ресурсами інфраструктури та системного софту. У розробці програмного забезпечення такі інструменти, як автоматизовані тести, репозиторії коду та сервери збірки, підвищують якість програмної інженерії. Якщо хмарну інфраструктуру розглянути як вихідний код (source code), то є можливість застосувати ті самі методи до коду інфраструктури, що й до коду програмного забезпечення. Використання IaC для реалізації методу перевірки завдань може бути задіян як управління інфраструктурою (мережами, віртуальними машинами, балансирам навантаження та топологією підключення) в описовій моделі з використанням того ж керування версіями. Подібно до принципу, що той самий вихідний код генерує той самий двійковий файл, модель IaC генерує те саме середовище кожного разу, коли вона застосовується.

IaC є ключовою практикою DevOps і використовується в поєднанні з безперервною доставкою.

Ідемпотентність є принципом інфраструктури як коду. Ідемпотентність – це властивість, що команда розгортання завжди встановлює цільове середовище в однакову конфігурацію, незалежно від початкового стану середовища. Ідемпотентність досягається автоматичною конфігурацією існуючої цілі або відкиданням існуючої мети та відтворенням нового середовища тому якщо потрібно внести зміни, вона редагує джерело, а не ціль. Відповідно до цього, за допомогою IaC є можливість вносити зміни до опису середовища та версії моделі конфігурації, яка зазвичай міститься у добре документованих форматах коду, таких як JSON. Інструменти IaC використуваються в основному для автоматизації серії кроків, які необхідно повторювати багаторазово на декількох ресурсах, для досягання однакового результату.

Процес розробки IaC коду подібен до практики розробки програмного забезпечення, в якій розробники контролюють версії коду, тестують ітераціями, поки програмне забезпечення не буде готове для використання. Для реалізації принципу «Інфраструктура-як-код» використовують високорівневі мови опису для кодування більш різноманітних процесів розгортання, наприклад такі як YAML, JSON, HCL.

Серед найбільш популярних інструментів реалізації підходу «Infrastructure as a Code» для автоматизації розгортання представлені Ansible, CloudFormation і Terraform.

Рисунок 1.14 – Порівняльна таблиця інструментів IaC

AWS CloudFormation був випущений у 2011 році як безкоштовний сервіс для використання в хмарі Amazon. Команди можуть автоматизувати розгортання складних середовищ, створюючи декларативні шаблони у форматі JSON і передаючи їх у службу через виклик API або інтерактивну веб-консоль. CloudFormation використовує модель декларативного шаблону,

тому розробникам інфраструктури не потрібно виконувати низькорівневе програмування; вони просто вказують необхідні їм ресурси в шаблоні JSON. Автори можуть моделювати складні середовища в цих шаблонах, які включають мережеву інфраструктуру, екземпляри Elastic Compute Cloud (EC2) і балансувальники навантаження . CloudFormation підтримує більшість ресурсів AWS, але розробники шаблонів також можуть використовувати власні ресурси, які використовують складнішу логіку програмування.

Рисунок 1.15 – Схема взаємодії AWS CloudFormation

Кінцевий результат шаблонів CloudFormation називають стеками (stack), які представляють собою набори пов’язаних ресурсів, з яких складається програма. Наприклад, шаблон може запустити набір веб-серверів і серверів баз даних. Разом ці сервери складають повний стек веб-додатків. AWS рекомендує організовувати стеки на основі життєвого циклу та права власності. Шаблони AWS CloudFormation записуються як файли JSON із звичайним текстом, тому їх легко зрозуміти, прочитавши код. Ці шаблони можна перевірити в загальнодоступних або приватних сховищах контролю версій. Рекомендується контроль версій AWS, оскільки він дозволяє командам співпрацювати під час процесу розробки та легко розуміти, які зміни відбувалися від однієї зміни до іншої. Для досвідчених команд зміни шаблону, перевірені в системі контролю версій, можуть автоматично запускати процес складання в рамках безперервної доставки . Конвеєр може включати етапи тестування, що гарантують, що шаблон використовує дійсний JSON, а також завдання після збірки, щоб переконатися, що інфраструктура належним чином забезпечена та функціонує належним чином. CloudFormation дозволяє користувачам передавати значення під час запуску через параметри шаблону. Це надає можливості налаштування та усуває необхідність жорсткого кодування інформації в шаблоні. Вхідні параметри часто використовуються для визначення назви пари ключів, що

використовуються в SSH для екземплярів EC2 або для діапазонів Central Identities Data Repository, які використовуються підмережами Amazon Virtual Private Cloud (VPC). Це типові приклади, але розробники шаблонів часто створюють і використовують параметри для різних випадків використання.

HashiCorp Terraform – це програмний інструмент з відкритим вихідним кодом у вигляді коду (IaC), який дозволяє програмно створювати фізичні ресурси, необхідні для роботи програми. Користувачі Terraform визначають і забезпечують конфігурацію інфраструктури за допомогою мови конфігурації, схожої на JSON, яка називається HCL (HashiCorp Configuration Language). Простий синтаксис HCL дозволяє легко надавати та повторно надавати інфраструктуру в кількох хмарних і локальних центрах обробки даних.

Також HashiCorp пропонує комерційну версію Terraform під назвою Terraform Enterprise. Згідно з веб-сайтом HashiCorp, комерційна версія включає корпоративні функції поверх Terraform з відкритим кодом і включає фреймворк під назвою Sentinel, який може реалізовувати політику як код. Інструмент є відкритим для контриб'юторів на GitHub, написаний на мові програмування Golang.

Рисунок 1.16 – Схема взаємодії Terraform з API хмари

Terraform дозволяє користувачам визначати всю свою інфраструктуру просто за допомогою файлів конфігурації та контролю версій . Коли надана команда для розгортання та запуску сервера, бази даних або балансувальника навантаження, Terraform аналізує код і переводить його у виклик інтерфейсу прикладного програмування (API) до постачальника ресурсів. Оскільки Terraform є відкритим вихідним кодом, розробники завжди можуть розширити корисність інструменту, написавши нові плагіни або зібравши різні версії існуючих плагінів.

Terraform контролює зчитування та інтерполяцію виконання планів

ресурсів, графіків ресурсів, функцій керування станом та файлів конфігурації. Ядро складається з скомпільованих двійкових файлів, написаних мовою програмування Go. Кожен скомпільований двійковий файл діє як інтерфейс командного рядка (CLI) для зв'язку з плагінами через віддалені виклики процедур (RPC). Додатки Terraform відповідають за визначення ресурсів для конкретних служб. Це включає автентифікацію постачальників інфраструктури та ініціалізацію бібліотек, які використовуються для викликів API. Плагіни Provisioner також написані на Go використовуються для виконання команд для визначеного ресурсу.

Файли конфігурації, які описані у форматі HCL або JSON, збергають опис необхідних специфічними ресурсів для кожного провайдера, необхідні для запуску однієї сервісу або всієї інфраструктури. Terraform генерує план виконання с записом у файл «terraform.state» та описує, що він буде робити для досягнення бажаного стану, а потім виконує його для побудови описаної інфраструктури.

Рисунок 1.17 – Організація роботи Terraform зі станом

На рисунку 1.17 окремо зображений один с найважливіших об'єктів Terraform – це terraform.state файл. Terraform має зберігати стан керованої інфраструктури та конфігурації. Цей стан використовується для порівняння різниці між реальними, вже створеними, ресурсами та ресурсами описаними у конфігурації. Файли з розширенням «*.tf» містять опис майбутніх ресурсів та після виконання команди «terraform plan» стан інфраструктури у форматі JSON зберігається за замовчуванням у локальному файлі terraform.state або у конфігурованому бекенд сховищі. Віддалене місце для файла дуже корисне у випадку, коли над інфрастуктурою працюють команда і хтось з інженерів працює с неактуальною версієї інфраструктури. У такому випадку Terraform не буде бачити відповідності ресурсів в файлі до реальних ресурсів та виникне коллізія та інструмент може використовувати віддалене блокування,

щоб уникнути одночасного запуску сценарію Terraform двома або більше різними користувачами. Як висновок, Terraform заблокує стан для всіх операцій, які можуть записувати його в хмарному середовищі.

Переваги використання Terraform включають можливість:

- підтримка кількох хмарних платформ;
- можливість вносити поступові зміни в ресурси;
- забезпечення підтримки програмно-визначених мереж;
- можливість імпортувати наявні ресурси в стан Terraform та блокувати модулі перед застосуванням змін стану, щоб гарантувати, що лише один користувач не може вносити зміни одночасно.

Однак використання Terraform має деякі недоліки.

- Нові випуски та оновлення можуть мати помилки.
- Terraform state повинен постійно синхронізуватися з актуальною інфраструктурою.
- Потрібно додаткове розуміння HCL.
- Інструмент немає обробки помилок.

Порівнюючи IaC інструменти та враховуючі вищезгадані переваги було прийнято рішення обрати Terraform для описання та поріняння станів у контексті перевірки вирішеного завдання у хмарній інфраструктурі.

1.5.3 Безперевна інтеграція автоматизованого рішення

Для реалізації перевірки хмарних завдань вкрай важливо, щоб процесс створення необхідної базової інфраструктури був автоматизований та був звучним у користуванні. Саме для цього у подібних рішеннях потрібно

використовують автоматизацію процессу розгортання Terraform коду та готовность його до модифікацій студентом. Для створення фундаменту інтеграції Terraform коду потрібно організовати процесс безперервної інтеграції та доставки.

Рисунок 1.18 – Організація конвеєру для CI/CD

CI/CD – це скорочення від Continuous Integration/Continuous Delivery/Continuous Deployment. Це дозволяє командам швидше створювати, тестувати та випускати програмне забезпечення. CI/CD по можливості виключає ручне втручання людини – автоматизує все, окрім остаточного розгортання коду вручну у виробничому середовищі. У якості основного інструмента для реалізації автматичного розгортання хмарних ресурсів був обран Jenkins – давно створену і добре відому платформу CI/CD. Інструмент автматизації є безкоштовним і відкритим, хоча корпоративні розробники можуть платити за підтримувані версії, такі як CloudBees CI. Як сервер автоматизації CI, Дженкінс автоматично запускає кроки в конвеєрі CI/CD, а сама платформа взаємодіє з іншими інструментами в конвеєрі, такими як сховища артефактів і вихідного коду, програмне забезпечення для керування конфігурацією та засоби автоматизації.

Однією з основних переваг Jenkins є ряд плагінів, доступних для платформи. Плагіни Jenkins можуть бути розроблені для локального та хмарного використання. Компанії, які використовують основних постачальників хмарних послуг, можуть інтегрувати свої розгортання з Jenkins за допомогою плагінів. Наприклад, користувачі AWS можуть запускати процеси CI від Amazon Cloud до Jenkins і навпаки.

Рисунок 1.19 – Приклад інформаційної панелі Jenkins збірки для Terraform

Найбільш корисними та актуальними Jenkins плагінами протягом виконання кваліфікаційної роботи були наступні:

- Перегляд інформаційної панелі: для перегляду та контролю стану всіх завдань CI з інформаційної панелі Jenkins.
- Тестові плагіни аналізу: група плагінів для тестування коду на етапі CI, перш ніж команда програмного забезпечення розгорне код у середовищі розробки, стадії або виробництва.
- Плагін Build Pipeline: для перегляду завдань у конвеєрі та визначення ручних завдань і тригерів для завдань, які можуть потребувати втручання перед будь-яким типом виконання.

Jenkins може бути встановлен або розгортнут в Microsoft Windows, будь-якому дистрибутиві Linux та Apple MacOS. Оскільки Jenkins може працювати на будь-якій ОС, IT-команда може встановити сервер CI незалежно від систем компанії. Безперечно є можливість запуску Jenkins у хмарі, завантажуючи та розгортаючи його на віртуальній машині. Крім того, Jenkins може також працювати на контейнері або масштабувати та організувати інструмент CI за допомогою Kubernetes за допомогою безкоштовного образу Jenkins на Docker Hub.

2 ПРОЕКТУВАННЯ

2.1 Формування вимог до автоматизованого рішення

На підставі аналізу даних серед усіх можливих методів було прийнято рішення спроектувати повноцінну систему автоматизації перевірки інфраструктури в хмарному середовищі з використанням розгортання базової

інфраструктури у хмарі та відстеження змін стану за допомогою методології «Інфраструктрура-як-код» та безперервної інтеграції системи при кожному запуску завдання. Життєвий цикл системи автоматизації розпочинається із задуму про її створення, потім вона проходить етапи проектування, впровадження, випробовувань і використання. Закінчується життєвий цикл виводом системи автоматизації з використання із причин морального або фізичного старіння. Формування вимог до автоматизованої системи включає наступні етапи робіт:

- обстеження системи і обґрунтування необхідності створення системи;
- формування вимог користувачів до системи;
- оформлення звіту про обстеження системи

Для створення автоматизаційного рішення, описаного у дипломній роботі, були висунуті наступні вимоги:

- Система повинна автоматично перевіряти завдання студента та відображати результат виконання на користувацький інтерфейс.
- Алгоритм перевірки завдання мусить виявляти відмінності між конфігурацією інфраструктури вирішеного та невирішеного завдання.
- Система повинна бути гнучкою до модернізації та зручної для кінцевого користувача
- Розробка системи має відбуватись з використанням продуктів з відкритим програмним кодом (Jenkins, Terraform, Docker).
- Система повинна мати можливість інтеграції з LCMS/LMS у перспективі.

2.2 Проектування концепції практичних завдань навчального курсу

Як було відмічено вище, основною задачею системи автоматизації є виявлення різниці між базисним та модифікованим станом хмарної

інфраструктури після виконання студентом завдання по створенню певних ресурсів у хмарному середовищі. Завдання являє собою набір послідовних дій по створеню, редагуванню або модифікації ресурсів інфраструктури (наприклад, EC2 екземплярів або балансувальників навантаження ELB), які студент поступово виконує у особистому AWS аккаунті з метою отримання результату після перевірки правильності виконання системою автоматизації. Складність завдань може бути різною – від простих задач у 2-3 крока до повноцінної побудови елементу проектної інфраструктури, яка може бути використана у рельному проекті. Слід зазначити, що концепція завдань також може передбачати плавний прогресивний перехід від простих завдань до більш складних та комплексних, наприклад від створення одного EC2 інстанса у публичній підмережі до створення повної мережі з налаштуванням мережевих маршрутів до інших мереж EC2 та прокладання фаєрволу між ними для відокремлення вхідного трафіку.

Залежачи від цього, на базовому етапі проектування концепції пропонується перевірка працездатності системи на невеликій групі студентів та опрацьовуючи декілька розділів навчального курсу предмета хмарних технологій. Отримані результати часткового запуску системи треба детально проаналізувати та опрацювати для визначення актуальності роботи системи у майбутньому та необхідності модернізації.

Рисунок 2.1 – Схема взаємодії користувачів навчальної системи

На рисунку 2.1 відображена концепція взаємодії усіх користувачів системи, включаючи викладачів, студентів та IT- спеціалістів. Концепція передбачає урегулювання окремого доступу кожного користувача до системи та паралельний запуск конвеєра для створення базової інфраструктури для виконання завдання.

2.3 Прототипування автоматизації для валідації завдань

Як було відмічено у попередніх розділах кваліфікаційної роботи, автоматизація первірки повинна бути імплементована за допомогою інструменту IaC Terraform, який може зберігати стан розгорнутої хмарної інфрастурктури у state файлі та відстеження змін до неї за допомоги інструменту безперервної інтеграції Jenkins.

Але відкритим залишилось питання про розміщення окремого хмарного середовища для студента, у якому планується виконання студентом завдань та інтеграція його з системою. Враховуючі усі висунуті вимоги до системи, пропоновано створити концепцію взаємодії між студентським AWS аккаунтом, який студент мусить зареєструвати самостійно до початку курсу та AWS аккаунту, на якому розгорнута система через використання AWS IAM ролей та довірених політик для доступу на сторонні хмарні аккаунти. Адміністратор створює роль для доступу між обліковими записами AWS та встановлює довіру між обліковим записом, якому належить ця роль, і ресурсами (довіряючий обліковий запис) і обліковим записом, який містить користувачів (довірений обліковий запис) через механізм AWS role policy assume. В обох аккаунтах повинна бути щонайменше 2 ролі – cloud-mentor-management у аккаунті AWS, де розгорнут функціонал системи, та cloud-mentor-connector для надання користувачеві дозволу на запуск базової інфраструктури завдання в особистому аккаунті.

Рисунок 2.2 – Схема архітектури автоматизованого рішення

На рисунку 2.2 представлена архітектурна схема інфраструктури яка забезпечує повноцінну роботу користувачів з автоматизованою системою Cloud Mentor. Вихідний код завдань має зберігатись у системи контролю версій, щоб розробники мали змогу відстежувати зміни у репозиторії та паралельно вести розробку нововведень або виправляти помилки. У процесі налаштування конвеєру безперевної інтеграції Jenkins один зі степов буде

містить загрузку усіх даних завдань з репозиторія GitLab або GitHub, щоб завдання студента завжди було актуальним при кожному запуску. Після цього Terraform ініціалізує вихідні файли завдання, надсилає запити на API хмарного провайдера (AWS) та розгортає базовий стан інфраструктури завдання зі збереженням його у файл terraform.tfstate. Пайплайн робить копію інфраструктури вирішенного завдання у окремий файл completed_task.tfstate та видаляє частину ресурсів, модифікація або створення яких є метою та критерієм виконання завдання студентом. Паралельно цьому скрипт у конвеєрі створює окрему копію невирішеного завдання uncompleted_task.tfstate. Студенту дається час на виконання завдання, прописаний у логіці пайплайну у вигляді timeout, який може бути різним в залежності від типу та рівні складності завдання. Принцип перевірки завдання полягає у заміні файлу terraform.tfstate на кінцевий completed_task.tfstate та перевірці Terraform різниці у ресурсах між заздалегідь виконаним таском та станом ресурсів у інфраструктурі студента.

```
# Опис ресурсу мережевої інфраструктури AWS VPC
resource "aws_route_table" "public_rt" {
  count  = var.deploy_task_resources ? 1 : 0
  vpc_id = aws_vpc.cloud_mentor_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gw.id
  }
}
```

Видалення ресурсів на початку запуску конвеєра реалізовано за допомогою підрахунку створених ресурсів методом count у мові високого рівня HCL, що використовує Terraform для опису маніфестів. Мета аргумент «count» приймає ціле число і створює багато екземплярів ресурсу або модуля. Кожен екземпляр має окремий об’єкт інфраструктури, пов’язаний з ним, і кожен окремо створюється, оновлюється або знищується під час застосування конфігурації. За допомогою Terraform змінних і тернарних умовних виразів ми отримуємо створенний ресурс aws_route_table, якщо змінна «deploy_task_resources» дорівнюватиме «true», і у випадку значення

«false», ресурс створюватись не буде. Після виконання студентом вправи або по закінченню таймауту конвеєру усі ресурси будуть видалені з аккаунту студента.

3 РОЗРОБКА І ТЕСТУВАННЯ АВТОМАТИЗОВАНОГО РІШЕННЯ

3.1 Підготовка локального оточення

На початку активної фази розробки автоматизовного рішення перш за все необхідно підготувати локальне оточення для реалізації і створення інфраструктурного коду та налаштування усіх елементів загальної хмарного оточення. Першим кроком для цього є створення облікового запису AWS аккаунту на конкретний email та реєстрація платіжніх реквізитів (кредитної чи дебетової картки).Треба відмітити, якщо у користувача немає раніше створеного облікового запису, він маєте право на безкоштовний 12-місячний пробний період, протягом якого є можливість безкоштовно використання більшість продуктів AWS майже без суттєвих обмежень. Після завершення реєстрації та підтвердження облікового запису створити нового користувача для повсякденних завдань та видалити кореневий обліковий запис (root user) для створення менш потужних облікових записів. Остання дія зумовлена правилами безпеки з метою запобігти взлому або потраплянню до root юзера зловмисників, які будуть мати доступ до платіжної інформації.

Рисунок 3.1 – Інтерфейс консолі керування AWS

Для отримання доступу у AWS з локального терміналу потрібно встановити та сконфігурувати утиліту awscli. AWS CLI підтримує використання іменованих профілів (aws config profile), які зберігаються у файлах config та credentials. Також є можливість налаштувати додаткові профілі, використовуючи команду «aws configure –profile» параметром, або шляхом додавання записи до config і credentials файлів. Наступним кроком

конфігурації аккаунту буде створення ключів доступу до облікового запису користувача AWS – AWS Access Key ID та AWS Secret Access Key. Для генерації ключа необхідно на головній сторінці AWS вибрати пункт «My Security Credentials»; у розділі «Access keys» створити пару ключей та збергіти їх локально.

Після генерації данних для входу необхідно створити локальний профіль для запуску терраформу для локального дебагу:

Рисунок 3.2 – Вивід команди конфігурації aws cli

Після підготування AWS даних, треба встановити Terraform для можливості локального запуску створенних завдань у вигляді модулів. Для інсталяції інструменту Terraform була використана утиліта «tfenv», фнукціононал якої дозволяє вибирати основную версію продукта та дає можливість оперувати декількома версіями коду з різними версіями Terraform. Для первірки коректності локлального оточення для розробки потрібно ініціалізувати Terraform код через команду «terraform init» та переконатися, що локальнний бекенд синхронізовний з AWS S3 сховищем. Це дозволяє розробляти проект декільком девелоперам і не буде виникати ситуацій, коли один розробник застосував нові зміни до інфраструктури, перешкодивши роботу іншим.

```
terraform {

  backend "s3" {
    bucket = "test_bucket"
    key    = "cloud-mentor/terraform.state"
    region = "us-east-1"
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 3.0"
    }
}
```

На прикладі відображен Terraform код опису віддаленого бекенду та виклик провайдеру. Terraform покладається на плагіни, які називаються «провайдерами», для взаємодії з віддаленими системами. Конфігурації Terraform мають оголошувати, які постачальники програмного API їм потрібні, щоб Terraform міг їх встановити та використовувати. Крім того, деяким постачальникам потрібна конфігурація (наприклад, URL-адреси кінцевих точок або хмарні регіони). Також може виникнути проблема, якщо до одного віддаленого стейт файлу одночасно буде відбуватися доступ декількох розробників, тобто в один момент часу будуть внесені зміни до інфраструктури. В цьому випадку може виникнути коллізія terraform.state файлу, який може буде не коректно оброблений. Для уникання цієї ситуаціїї використовується Terraform lock, який може блокувати на момент будь-яких змін у файлі. Тобто, Terraform заблокує стан (terraform.state) для всіх операцій, які можуть записувати стан і це не дасть іншим отримати інший лок і потенційно пошкодити стан інфраструктури.

3.2 Розробка Terraform модулів для завдань

Огляд та концепцію розробки окремих Terraform модулів під кожне завдання буде розглянуто на прикладі модулю одного завдання на тему побудування мережевої інфраструктури з використання сервісів AWS VPC.

У мові програмування функція організовує код і робить його придатним для повторного використання. У Terraform від HashiCorp це важливе завдання виконується модулями. Модуль Terraform — це папка, яка містить набір файлів конфігурації. Це дає змогу адміністраторам створювати шаблони для компонентів інфраструктури як коду, які можуть приймати параметри для створення, збільшення та зменшення з незначними змінами.

Рисунок 3.3 – Структури модулів Terraform для створення AWS ресурсів

Повний план виконання Terraform перед застосуванням (terraform plan) включає кореневий модуль, папки для дочірніх модулів, а також вхідні та вихідні змінні. Кореневий модуль. Terraform визначає кореневий модуль як робочий каталог або каталог верхнього рівня, який містить файли конфігурації Terraform. Тут ви можете запустити команду terraform apply , яка забезпечує точку входу до модулів, які використовуються для побудови інфраструктури.

Повторно використовувані модулі визначаються за допомогою всіх тих самих концепцій мови конфігурації, які можна використовувати в кореневих модулях. Найчастіше за усе Terraform модулі використовують для створення повторно використовуваних модулів, які можуть включати інші конфігурації за допомогою блоків. Як параметри модулі можуть оперувати наступними елементами Terraform:

- Вхідні змінні (input variables) для прийняття значень від модуля, що викликає.
- Вивід значення (output variables) для повернення результатів до модуля, що викликає, який потім він може використовувати для заповнення аргументів в іншому місці.
- Ресурси (resource definition) для визначення одного або кількох об’єктів хмарної інфраструктури.

Для повторного використання значення ресурсів модулю завдання створені змінні для адресного пространства підмереж AWS VPC

```
variable "vpc_cidr" {
  description = "Initial VPC for network task"
  default     = "10.0.0.0/16"
}

variable "public_subnet_cidr" {
  description = "CIDR for the public subnet"
  default     = "10.0.1.0/24"
}
```

У прикладі модулю для завдання по мережевій інфраструктурі у файлі

ec2.tf використуається опис ресурів EC2 інстансів, які будуть розгорнуті у різних підмережах (приватній та публічній).

```
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.cloud_mentor_vpc.id
  cidr_block        = var.public_subnet_cidr
  availability_zone = "eu-central-1a"

  tags = {
    Name  = "Cloud Mentor Public Subnet"
    Owner = "E-mentor"
  }
}
```

Також був створений приватний SSH для авторизації користувача на інстанс у приватній підмережі через бастіон (інстанс у публічній підмережі).

```
resource "aws_key_pair" "default" {
  key_name   = "cloud-mentor-keypair"
  public_key = tls_private_key.ssh.public_key_openssh
}

resource "tls_private_key" "ssh" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
```

Мережева інфраструктура завдання зберігається у файлі vpc_resources.tf. Перед початком виконання вправи студенту надається AWS VPC мережа с вказаним адресним пространством, розподіленні на логічну приватну та публічну підмережа (aws_subnet).

```
# VPC мережа
resource "aws_vpc" "cloud_mentor_vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true

  tags = {
    Name  = "Cloud Mentor VPC"
    Owner = "E-mentor"
  }
}

# Публічна підмережа
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.cloud_mentor_vpc.id
```

```
  cidr_block        = var.public_subnet_cidr
  availability_zone = "eu-central-1a"

  tags = {
    Name  = "Cloud Mentor Public Subnet"
    Owner = "E-mentor"
  }
}

# Приватна підмережа
resource "aws_subnet" "private_subnet" {
  vpc_id            = aws_vpc.cloud_mentor_vpc.id
  cidr_block        = var.private_subnet_cidr
  availability_zone = "eu-central-1b"

  tags = {
    Name  = "Cloud Mentor Private Subnet"
    Owner = "E-mentor"
  }
}
```

Окрім цього, до базової мережевої інфраструктури завдання треба додати ресурс мережевого шлюзу AWS Internet Gateway (IGW) для трансляцію мережевих загальнодоступні адреси IPv4 у глобальний інтернет та AWS Security Groups для для контролю та обмеження вхідного та вихідного трафіку на інстанси EC2.

```
# Мережевий шлюз
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.cloud_mentor_vpc.id

  tags = {
    Name  = "Cloud Mentor IGW"
    Owner = "E-mentor"
  }
}

# Налашовані брандмауери для EC2
resource "aws_security_group" "server_sg" {
  name        = "server_sg"
  description = "Allow traffic from public subnet"
  vpc_id      = aws_vpc.cloud_mentor_vpc.id

  ingress = var.deploy_task_resources ? local.server_sg_ingress : null
  egress  = var.deploy_task_resources ? local.server_sg_egress : null

  tags = {
    Name  = "Private instance SG"
```

```
    Owner = "E-mentor"
  }
}

resource "aws_security_group" "server_sg" {
  name        = "server_sg"
  description = "Allow traffic from public subnet"
  vpc_id      = aws_vpc.cloud_mentor_vpc.id

  ingress = var.deploy_task_resources ? local.server_sg_ingress : null
  egress  = var.deploy_task_resources ? local.server_sg_egress : null
}
```

Для подальшого використання вихідних даних у блоку верифікації студентського завдання були прописані вихідні змінні для IP адресу приватного та публічного інстансу та змінна, що містить значення приватного ключа SSH, щоб студент мав змогу власноруч авторизуватись на інстанс.

```
output "private_instance_ip" {
  value = aws_instance.private.private_ip
}

output "public_instance_ip" {
  value = aws_instance.bastion.public_ip
}

output "ssh_pem_key" {
  sensitive = true
  value = tls_private_key.ssh.private_key_pem
}
```

3.3 Розробка концепції запуску завдань для перевірки

Розглянемо принцип розробки запуску завдання на прикладі одній з вправ, пропонованої до внесення у загальний курс автоматизації для дисципліни «хмарні технології». У попередньому розділі є детальний опис методу розробки модуля Terraform для використання його у якості базової інфраструктури як шаблон, що містить заздалегідь виконане завдання. Даний

розділ описує створення валідації завдання для мережевої інфраструктури, варіанти тест-кейсу та подальший запуск його за допомогою безперервної інтеграції.

Зміст завдання полягає у тому, що на початку вправи у власному AWS аккаунті студента розгортається окрема VPC мережа з адресним пространством 10.0.0.0/16 та дві підмережі - публічна 10.0.1.0 та приватна 10.0.2.02 з маскою /24. У вищезгаданих підмережах розгортаються дві віртуаільні інстанси EC2 – один, що у public subnet (Bastion host) має приватний та публічний IP адрес, інший (Server host) тільки приватний IP.

Роутінг мережевих пакетів інтернет трафіку ззовні проходить через ресурс мережевого шлюзу AWS Internet Gateway та внутрішні таблиці маршрутизації (route tables) под кожної підмережі. Також для кожного інстансу додається браундмауер (security group), який контролює вхідний/вихідний трафік.

Рисунок 3.4 – Схема інфрастурктури завдання AWS ресурсів

Суть завдання полягає у тому, щоб студент зміг власноруч виправити помилки в інфраструктурі та мав можливість під’єднатись до інстансу у приватній мережі через бастіон хост використовуючи протокол SSH та приватний ключ. Для вирішення завдання студенту потрібно додати корректні адреси у route tables на шлюз і приватну підмережу та додати правила у групи безпеки брандмауера, щоб дозволити SSH трафік на мережевий інтерфейс сервер інстансу у приватній підмережі. Критерієм оцінювання правильності виконання таска буде розгорнутий у кінці файл верифікації test.tf, який запустить скрипт для перевірки SSH з'єднання. Скрипт включає в себе генерацію SSH конфігу з вхідними параметрами, отриманими від вихідних змінних у ресурсах, які передають у команду оболонки Bash та запуск SSH з'єднання з указанням файлу приватного ключа бастіона.

```
resource "null_resource" "generate_ssh_config" {
  triggers = {
    always_run = timestamp()
  }

  provisioner "local-exec" {
    command = "cat >> ~/.ssh/config <<EOL\n$
{data.template_file.ssh_config.rendered}\nEOL"
  }

  depends_on = [
    null_resource.provision_ssh_key
  ]
}

module "check_ssh_connection" {
  source = "matti/resource/shell"

  depends = [
    null_resource.generate_ssh_config
  ]

  environment = {
    PRIVATE_INSTANCE_IP = data.terraform_remote_state.infra.outputs.private_inst
ance_ip
  }

  command = "eval `ssh-agent`; ssh -o ConnectTimeout=10 -i ~/.ssh/cloud-
mentor.pem -o StrictHostKeyChecking=no ec2-user@$PRIVATE_INSTANCE_IP;"
  trigger = timestamp()
}

output "result" {
  value = local.tests_result
}
```

Після отримання статусу від команди перевірки SSH з'єднання до server host він записується у змінну local.tests_result та передається як вхідний параметр до Jenkins конвеєру.

Рисунок 3.5 - Схема конвеєру для перевірки завдання

Як було відмічено раніше, Jenkins пайплайн під назвою «cloud-mentor-pipeline» (рис.3.5) відповідає за підготовку інфраструктури в хмарному аккаунті студента, виконує необмежену кількість перевірок при кожному

запуску завдання та у випадку отримання позитивного результата повідомляє студента про успішне виконання вправи та перехід до наступної. Треба зазначити, що на кожне завдання окремо встановлен таймер завдання (від 1 до 3 годин), за який студент має виконати вправу, щоб вона була зарахована в системі. У випадку, якщо ліміт відведеного часу на розв'язання завдання вичерпан або студент самостійно призупинив роботу над завданням, у конвеєрі присутня окрема стадія видалення усієї інфраструктури студента, яка створюється завдання у особистому аккаунті. Слід зазначити, що відповідальним за запуск Jenkins пайплайну може бути як викладач курсу, так і студент власноруч. Більш детальна блок-схема алгоритму перевірки завдання пайплайном відображена у додатку А.

Для кожного етапу завдання у конвеєрі є крок додаткого підтвердження від студента, наприклад при створенні інфрастурктури або готовності до перевірки.

```
05:54:44  Plan: 3 to add, 0 to change, 0 to destroy.
05:54:44
05:54:44  Changes to Outputs:
05:54:44    + Assume_role   = (known after apply)
05:54:44    + ReadOnly_role = (known after apply)
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] stash
05:54:59  Stashed 57 file(s)
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (approve task resources deploy)
[Pipeline] script
[Pipeline] {
[Pipeline] input
05:54:59  Approve deploy task infrastructure?
05:54:59  Approve or Abort
```

У випадку, коли студент не повностю виконав завдання або не виконав жодного кроку взагалі, конвеєер перевірки надає підказку з підрахуванням степів, які мусить зробити студент, щоб завдання було зроблено. Логіка

підрахунку ресурсів досягається шляхом перебору ресурсів зі змінною deploy_task_resources у стані «false» та виводі результату у консоль конвеєру Jenkins.

```
05:58:57  + terraform show -json tfplan
[Pipeline] readJSON
[Pipeline] }
[Pipeline] // dir
[Pipeline] echo
05:59:02  Steps left to finish the task -> '2'.
[Pipeline] error
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
05:59:03  ERROR: Task check failed -> The task has not been solved!
05:59:03  Retrying
[Pipeline] {
[Pipeline] stage
[Pipeline] { (check task input)
[Pipeline] script
[Pipeline] {
[Pipeline] input
05:59:03  Is the task completed and ready to check?
05:59:03  Check or Abort
```

3.4 Подальша інтеграція автоматизованої перевірки з LCMS

На даному етапі розвитку проекту Cloud Mentor функціонал взаємодії з користувачем представлений у Jenkins пайплані. Взаємодія у пайплані є у вигляді полів для введення тексту, форм для підтвердження формату Approve або Abort.

Рисунок 3.6 - Приклад взаємодії з користувачем у Jenkins пайплані

Система управління навчальним контентом дозволяє викладачам створювати власний приватний веб-портал, який може бути наповнений динамічним курсом, які розширюють рівень навчання без залежності до місця навчання. Інтерфейс LMCS, представлений на рис. 3.7 розроблений для

швидкої доступності користувачів до системи та може забезпечити просту навігацію як на десктопних, так і на мобільних пристроях. Для відображення отриманого резльутату від інструменту безперервної інтеграції слід використувати персоналізовану інформаційну панель для відображення актуальної інформації поточного курсу, а також статусу завдань на виконанні. Слід зазначити, що LMCS може бути стороння або розгорнута у хмарному середовищі проекта (Cloud Mentor AWS аккаунт), що забезпечить зручність у встановленні та підтримки у майбутньому.

Рисунок 3.7 – Блок-схема взаємодії користувача з LCMS

У наступній фазі розвитку проекту передбачена повноцінна інтеграція функціоналу користувацької взаємодії з платформою або системою управління навчальним контентом (LMCS). Запуск конвеєра студентом чи викладачем планується замінити на веб-інтерфейс відображення статусу завдань, результату перевірки та створення вікон для відображення помилок. Для зручного відображення актуальних статусів завдання у LMCS для студента чи викладача пропонується викоростати веб-додаток системи у вигляді інтернет платформи. Права доступу до навчального курсу та політики безпеки повинні бути розподілені відносно ролі користувача (адміністратор, викладач, студент).

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи була створена повноцінна система автоматизації перевірки інфраструктурного рішення в хмарних технологіях. У дипломній роботі проаналізовано можливості інтенсифікації навчання хмарним технологіям, проведено аналіз необхідності практичних занять у хмарних технологіях та винайдено метод автоматизації.

Розглянуто різновиди систем управління навчальним контентом, проведено огляд інстументів для використання в автоматизованому рішенні.

Вимоги до автоматизованого рішення були сформовані відносно результатів досліджень. Розроблено прототип та концепцію перевірки завдань у хмарному провайдері AWS за допомогою таких інструментів як Jenkins та Terraform

До недоліків автоматизованої системи можна віднести нереалізоване автоматичне резервне копіювання системи і відсутність відображення результату перевірки завдань у веб-інтерфейсі. Перелік питань поставлених для виконання роботи було опрацьовано у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Амато Віто. Основи організації мереж Cisco, том 1. - М.: Видавничий дім "Вільямс", 2004. - 512 с.
2. Селянінов С.В. Інтелектуальні мережі та комп'ютерна телефонія - СПб: Видавництво «Пітер», 2001. - 150 с.
3. Lukarić and Korin Lustig, A., Learning CAD by LMS and cloud computing implementation in Proc. 34th Int. Conv. MIPRO, May 2011, pp. 1123-1126
4. Кулаков Ю.А., Омелянський С.В. Комп'ютерні мережі. Вибір, встановлення, використання та адміністрування. - Київ: Видавництво. ЮНІОР, 1999, - 544с.
5. Оліфер В.Г., Оліфер Н.А. Комп'ютерні мережі. Принципи, технології, протоколи. - СПб.: Пітер, 2006 - 958 с.
6. Проблеми інформатизації: зб. наук. тез, 18-19 листопада 2021 р., Черкаси – Харків – Баку – Бельсько-Бяла, 2021р. - с. 7
7. Cloud Computing and its Impact on Online Education [Електронний ресурс] – Режим доступу до ресурсу, (Дата звернення: 27.11.21). https://iopscience.iop.org/article/10.1088/1757-899X/1094/1/012024
8. Johnson, L., Levine, A., & Smith, R. (2009). The 2009 Horizon Report. One Year or Less: Cloud Computing. Austin, Texas
9. AWS: [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/ (Дата звернення: 25.11.21).
10. Terraform [Електроний ресурс] – Режим доступу до ресурсу: https://www.terraform.io/ (Дата звернення: 23.11.21).
11. IaC [Електроний ресурс] – Режим доступу до ресурсу: https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC (Дата звернення: 21.11.21).