Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра *Інформаційно-мережної інженерії*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження шляхів автоматизації перевірки цілісності хмарної інфраструктури
(тема)

Виконав: студент 2 курсу групи *ІМІзм-19-2*
Батов В. В.
(прізвище та ініціали)

Спеціальність *172 Телекомунікації та радіотехніка*
(код і повна назва спеціальності)

Тип програми *освітньо-наукова*
(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

Навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра *ІНФОРМАЦІЙНО-МЕРЕЖНОЇ ІНЖЕНЕРІЇ*
(повна назва)

Рівень вищої освіти *ДРУГИЙ (МАГІСТЕРСЬКИЙ)*

Спеціальність *172 Телекомунікації та радіотехніка*
(код і повна назва)

Тип програми *ОСВІТНЬО-НАУКОВА*

Освітня програма Інформаційно-мережна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)

«____» _____ 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Батову Вадиму Володимировичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження шляхів автоматизації перевірки цілісності хмарної інфраструктури*

затверджена наказом університету від «*25*» *березня* 2021 р. № 33*Стз*

2. Термін подання студентом роботи до екзаменаційної комісії *26 травня 2021 р.*

3. Вихідні дані до роботи:. *Розробити алгоритм перевірки цілісності хмарної інфраструктури. Проаналізувати можливості використання подібної методики для автоматизації перевірки завдань, навчання та підготовки спеціалістів в області хмарних технологій*

4. Перелік питань, що потрібно опрацювати в роботі

Вступ

1. *Аналітичний огляд літератури*
2. *Практична частина*

Висновки

5. Перелік графічного матеріалу із зазначенням креслень, схем, плакатів, комп'ютерних ілюстрацій *слайди презентації в форматі Power Point: 11 слайдів*

(назва, мета роботи, огляд хмарних технологій, поняття інфраструктури як код, структура алгоритму автоматизації навчання, висновки)

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ.*	29.03 – 25.05.21	
2	*Підбір літератури за темою роботи.*	30.03 – 20.04.21	
3	*Виконання розділу 1*	05.04 – 20.05.21	
4	*Виконання розділу 2*	21.04 – 28.04.21	
5	*Оформлення пояснювальної записки*	10.05 – 18.05.21	
6	*Оформлення презентаційного матеріалу,*	14.05 – 20.05.21	
7	*Підготовка до захисту у ЕК*	15.05 – 24.05.21	

Дата видачі завдання ___*29.03.2021* р.____

Студент _____ *Батов В. В.*
(підпис) (прізвище, ініціали)

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

РЕФЕРАТ

Пояснювальна записка: 54 с., 20 рис., 14 джерел, 4 додатки.

Об’єкт роботи – хмарні технології

Мета роботи – Розробити алгоритм перевірки цілісності хмарної інфраструктури.

Розроблено алгоритм для перевірки цілісності інфраструктури. На основі якого створено прототип системи для автоматизації навчання та підготовки спеціалістів в області хмарних технологій з використанням продуктів з відкритим програмним кодом Docker, Jenkins та Terraform.

ХМАРНІ ТЕХНОЛОГІЇ, ЦІЛІСТНІСТЬ ІНФРАСТРУКТУРИ, AWS, TERRAFORM, JENKINS.

ABSTRACT

Explanatory slip: 54 p., 20 fig., 14 sources, 4 app.

The object of research – cloud technologies.

The purpose of work – Develop an algorithm for cloud infrastructure integrity check.

An algorithm has been developed to check the integrity of the infrastructure. A prototype system was created based on the algorithm to automate the training and education of specialists in the field of cloud technologies using open-source products Docker, Jenkins and Terraform.

CLOUD TECHNOLOGIES, INFRASTRUCTURE INTEGRITY, AWS, TERRAFORM, JENKINS.

ЗМІСТ

ВСТУП

Сьогодні все більшу популярність набирають публічні хмарні сервіси. Вони дають змогу фокусуватись на створення функціоналу і перекласти підтримку та обслуговування інфраструктури на сторону хмарних провайдерів. Також можна зазначити розвиток гнучких методології розробки програмного забезпечення та DevOps методології для покращення якості та збільшення швидкості доставки програмного забезпечення. Це зумовило підвищення попиту на спеціалістів, які мають досвід роботи з хмарними технологіями. Актуальною проблемою є підготовка кваліфікованих спеціалістів. Для цього можна використовувати автоматизовані системи перевірки, що дозволить підготувати більшу кількість спеціалістів та задовольнити високий попит.

В цій роботі розроблено алгоритм для перевірки цілісності хмарної інфраструктури, що може бути використаний як основа для автоматизації навчання та підготовки висококваліфікованих спеціалістів. Розроблений прототип системи складається з трьох компонентів, і дозволяє дуже легко додавати новий функціонал якщо це необхідно або провести його інтеграцію з існуючими системами.

ПЕРЕЛІК СКОРОЧЕНЬ

SaaS – Software as a Service - Програмне забезпечення як послуга;
PaaS – Platform as a Service – Платформа як послуга;
IaaS – Infrastructure as a Service – Інфраструктура як послуга;
API – Application Programing Interface – Програмний інтерфейс додатку
DSL – Domain-specific language – Предметно-орієнтована мова програмування
DevOps – Акронім від англ. development и operations
AWS – Amazon Web Services
GCP – Google Cloud Platform
CI – Continuous Integration – Безперервна інтеграція
CD – Continuous Delivery - Безперервна доставка
CLI – Command line interface – Інтерфейс командного рядка

1 АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ

1.1 Огляд хмарних технологій

1.1.1 Хмарні обчислення

Останні 15 років розвиток хмарних технологій почав стрімко зростати. Все більше компаній використовують переваги хмарних обчислень для більш ефективного вирішення бізнес задач, в певних випадках це дозволяє суттєво зменшити вартість обчислювальних послуг при цьому маючи змогу швидко масштабуватись для забезпечення потреб великої кількості [1, 2].

Хмарні обчислення представляють собою модель для забезпечення повсюдного, зручного доступу до мережі за запитом та загального доступу до пулу конфігурованих обчислювальних ресурсів (наприклад, мереж, серверів, сховищ, програм та послуг), які можуть бути швидко надані та випущені з мінімальними зусиллями. Хмарна модель складається з п’яти основних характеристик, трьох моделей обслуговування та чотирьох моделей розгортання [3].

Основні характеристики хмарних обчислень:

Самообслуговування. Споживач може в односторонньому порядку надавати обчислювальні можливості, такі як час сервера та мережеве сховище, за необхідності автоматично, не вимагаючи взаємодії людини з кожним постачальником послуг.

Широкий доступ до мережі. Можливості доступні через мережу та доступ до них здійснюється за допомогою стандартних механізми, що сприяють використанню різнорідних тонких або товстих клієнтських платформ (наприклад, мобільні телефони, планшети, ноутбуки та робочі станції)

Спільний пул ресурсів. Обчислювальні ресурси постачальника об’єднуються для обслуговування кількох споживачів за допомогою моделі з кількома орендарями, при цьому різні фізичні та віртуальні ресурси динамічно призначаються та перепризначуються відповідно до попиту споживачів. Існує

відчуття незалежності від локації в тому, що клієнт, як правило, не має контролю або знань щодо точного розташування наданих ресурсів, але може мати можливість вказати місце розташування на більш високому рівні.

Еластичність. можливість запросити або звільнити ресурси, в деяких випадках автоматично, для швидкого масштабування пропорційно попиту. З боку споживача можливості, доступні для забезпечення, часто здаються необмеженими і можуть бути надані в будь-якій кількості в будь-який час.

Вимірюваність обслуговування. Хмарні системи автоматично контролюють і оптимізують використання ресурсів, використовуючи можливості вимірювання на певному рівні абстракції, що відповідає типу послуги (наприклад, зберігання, обробка, пропускна здатність та активні облікові записи користувачів). Використання ресурсів можна контролювати та моніторити, забезпечуючи прозорість як для постачальника, так і для споживача послуги, що використовується.

Основні моделі обслуговування хмарних обчислень:

Програмне забезпечення як послуга (SaaS). Послуга, що надається споживачу, полягає у використанні програм постачальника, що працюють на хмарній інфраструктурі. До програм можна отримати доступ з різних клієнтських пристроїв через клієнтський інтерфейс, такий як веб-браузер (наприклад, веб-адреса електронної пошти), або програмний інтерфейс. Споживач не управляє та не контролює базову хмарну інфраструктуру, включаючи мережу, сервери, операційні системи, сховище чи навіть окремі можливості додатків, за винятком можливих обмежених параметрів конфігурації додатків.

Платформа як послуга (PaaS). Можливість, що надається споживачеві, полягає у розгортанні на хмарній інфраструктурі створених або придбаних споживачем додатків, створених з використанням мов програмування, бібліотек, сервісів та інструментів, що підтримуються провайдером. Споживач не управляє та не контролює базову хмарну інфраструктуру, включаючи мережу серверів, операційних систем або сховища, але контролює розгорнуті програми та, можливо, параметри конфігурації для середовища розміщення додатків.

Інфраструктура як послуга (IaaS). Можливість, що надається споживачеві, полягає в забезпеченні обробки, зберігання, мереж та інших

основних обчислювальних ресурсів, де споживач може розгортати та запускати довільне програмне забезпечення, яке може включати операційні системи та додатки. Споживач не керує та не контролює базову хмарну інфраструктуру, але контролює операційні системи, сховище та розгорнуті програми; і, можливо, обмежений контроль вибраних мережевих компонентів (наприклад, брандмауери хостів).

Типи розгортання хмар:

Приватна хмара. Хмарна інфраструктура призначена для виключного використання однією організацією, що складається з декількох споживачів (наприклад, бізнес-підрозділів). Це може бути власністю, управлінням та управлінням організації, третьої сторони або якоїсь їх комбінації, і воно може існувати як в приміщенні, так і поза ним (рис 1.1).

Рисунок 1.1 - Схема приватної хмари

Хмари спільного використання. Хмарна інфраструктура призначена для ексклюзивного використання певною спільнотами або організаціями, що мають спільні цілі (наприклад, місія, вимоги безпеки, політика та міркування щодо дотримання). Вона може належати, управлятися та експлуатуватися однією або кількома організаціями громади, третьою стороною або деякою їх комбінацією, і може існувати як у закритому центрі обробки даних так і у центрі загального користування.

Публічна хмара. Хмарна інфраструктура надана для відкритого використання широкому загалу (рис 1.2). Це може бути власністю, управлінням та управлінням бізнесу, академічної чи державної організації, або їх поєднання. Цей тип хмар існує в центрах обробки даних хмарного провайдера.

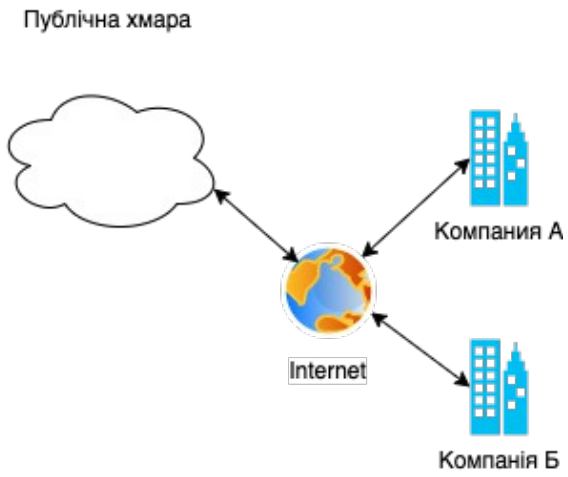

Рисунок 1.2 - Схема публічної хмари

Гібридна хмара. Хмарна інфраструктура - це сукупність двох або більше різних хмарних інфраструктур (приватної, спільного використання або публічної), які залишаються унікальними сутностями, але пов’язані між собою стандартизованою або власною технологією, що забезпечує перенесення даних та додатків (наприклад, розподіл хмари для балансування навантаження хмари).

1.1.2 Хмарні провайдери

Провідна світова дослідницька і консалтингова компанія у сфері інформаційних технологій Gartner проводить щорічний огляд хмарних провайдерів. До основних лідерів на ринку входять:

- AWS (Amazon Web Services);
- GCP(Google Cloud Platform)
- Microsoft Azure
- Alibaba cloud
- Oracle cloud

- IBM cloud
- Tencent Cloud

Згідно з останнім дослідження Gartner AWS (Amazon Web Services) являється лідером (рис. 1.3) на ринку і втримує цю позицію останні 10 років [4]. Тому в рамках цієї роботи використовувався провайдер AWS.

Рисунок 1.3 - Магічний квадрадрант в області хмарних обчислень

1.2 Методології розробки програмного забезпечення

1.2.1 Agile методологія

Сьогоднішня ситуація на ринку програмного забезпечення можна охарактеризувати як дуже динамічну, ринок змінюється під впливом зовнішніх факторів. Це зумовило потребу у створенні більш гнучких моделей розробки програмного забезпечення, здатних швидко адаптуватися під нові потреби ринку. Зараз дуже поширеним є комплекс “гнучких” методологій (з англ. Agile methods) [5]. Суть даних методологій в ітераційному підході, ми розділяємо наш продукт на маленькі частини. В кожну частину можна розглядати як маленький проект з наступними стадіями: планування роботи, виконання, тестування, фіксація результату, аналіз результатів. Таким чином після виконя кожної частини ми можемо видозмінювати вимоги для наступних. В сучасному світі, де технології розвиваються особливо стрімко можливість швидко адаптуватись може бути вирішальною для багатьох компаній.

1.2.2 Безперервна доставка

Поняття безперервної доставки з’явилось достатньо давно [6, 7], проте активний розвиток та масштабне використання в Україні почалось близько 5 років тому. Безперервна доставка (з англ. Continuous Delivery) являється однією зі складових DevOps методології [8]. Основна ідея якої полягає в збільшенню частоти випусків програмного забезпечення, його розробка у більш короткі терміни, для можливості мати більш надійний випуск нової версії в будь-який час. Це допомагає зменшити кількість змін від випуску до випуску, зменшити ймовірність критичних багів, та пришвидшити швидкість розробки.

Для реалізації безперервної доставки, зазвичай використовують автоматизацію процесів збірки, тестування та розгортання програмного забезпечення часто їх об’єднують у конвеєри (з англ. pipelines) (рис. 1.4).

Рисунок 1.4 - Приклад конвеєру безперервної доставки

Зазвичай для створення конвеєрів використовують спеціальне програмне забезпечення. Для створення простих конвеєрів можна використовувати вбудований функціонал в системи контролю версій:

- Github Actions
- Gitlab CI
- Travis CI

Для більш складних конвеєрів, де потрібна гнучкість використовують окреме програмне забезпечення для створення та зручної підтримки конвеєрів. Серед великих гравців найбільш популярними є Jenkins з відкритим програмним кодом, а також платні Atlasian Bamboo та TeamCity. Jenkins якляється найбільш поширеним через те, що він дуже гнучкий та безкоштовний. До недоліків Jenkins можна віднести те що він є досить складний у конфігуруванні та потребує окрему обчислювальну інфраструктуру, яку потрібно обслуговувати.

В даній роботі використовувався Jenkins, тому що він дозволяє створювати досить як прості так і складні конвеєри і підтримує велику кількість інтеграцій зі сторонніми сервісами. Існує можливість розгорнути Jenkins у

контейнері локально, що значно спрощує його конфігурацію.

1.2.3 Інфраструктура як код

Кожен хмарний провайдер пропонує набір послуг для користувача. Зазвичай провайдери використовують однакові рівні абстракції, для основних сервісів наприклад:

- Сервіс віртуальних машин, основне джерело обчислювальної потужності.
- Сервіс для зберігання великих об’ємів даних, так званих “data lakes” озер даних.
- Сервіс для віртуалізації мереж
- Сервіс для безсерверних обчислень, де провайдер бере на себе відповідальність за розгортку обчислювальної потужності з високою доступністю та автоматичним масштабуванням відповідно до трафіку.
- Сервіс контроля доступів до хмарних сервісів

Контроль цих послуг здійснюється через прикладний програмний інтерфейс (англ. Application Programming Interface, API). Для більш комфортної роботи з хмарними сервісами провайдери створюють графічні інтерфейси у вигляді веб консолей, де консолідовано інформацію по створеним ресурсам.

З часом у розробників з’явилась потреба в автоматичному створенні так контролю великої кількості ресурсів, при чому дуже важливо мати змогу створювати інфраструктуру яка має повторюваність і відтворюваність. Це дає змогу суттєво зменшити кількість помилок пов’язану з людським фактором і значно зменшити час для розгортання нового середовища. Такі вимоги підштовхнули розвиток поняття інфраструктура як код (англ. Infrastructure as Code). Основна ідея цього поняття полягає в декларативному описі конфігурації інфраструктури використовуючи підходи мов програмування. Зазвичай цей підхід реалізований через предметно-орієнтовану мову програмування (англ. Domain-specific language, DSL) та комп’ютерну програму, яка дозволяє створювати хмарні ресурси інтерпретуючи шаблони описані на предметно-орієнтованій мові (рис. 1.5).

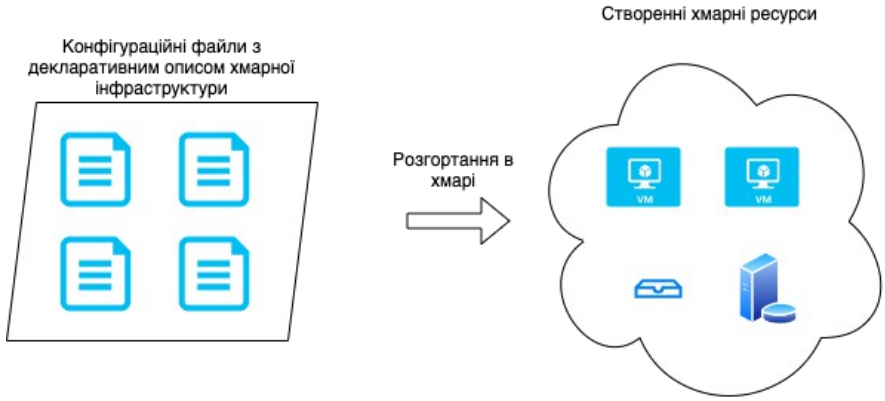

Рисунок 1.5 – Схема роботи підходу інфраструктура як код

1.3 Огляд інструментів для розгортання інфраструктури в хмарах

Хмарні провайдери часто пропонують використовувати свої сервіси, які імплементують поняття інфраструктура як код, проте вони мають суттєвий недолік, зазвичай вони специфічні для кожного провайдера. Натомість існують інструменти з відкритим програмним кодом, які не зав'язані на певного хмарного провайдера:

- Serverless framework
- Terraform
- Pulumi

Ці інструменти дозволяють створювати та оновлювати інфраструктуру спираючись на її декларативний опис. Як правило при створенні ресурсів вони формують файли які зберігають опис створених ресурсів та їх параметри, це дозволяє інкрементально оновлювати інфраструктуру. Дає можливість моніторити, що в кожен момент часу, конфігурація залишається правильною і можливість точно видалити інфраструктуру яка була створена без ризику пошкодити інші ресурси.

Pulumi найбільш молодий інструмент який з'явився у 2017 [9], Terraform та Serverless framework у 2014 та 2015 відповідно. Це дозволило йому ввібрати найліпше та вирішити певні проблеми попередників. Варто відмітити, що Serverless framework має фокус на створення інфраструктури за безсерверною моделлю, тому порівнювати його з Terraform та Pulumi, не зовсім доречно.

Якщо ж порівнювати Terraform та Pulumi між собою то основним параметром буде предметно-орієнтована мова програмування, яка використовується для опису інфраструктури. Terraform має свою предметно-орієнтовану мову програмування — HCL (рис. 1.6), Pulumi створений щоб використовувати об’єктно-орієнтовані мови програмування для опису інфраструктури, такі як TypeScript, Go, Python, C# (рис 1.7). Нижче приведено приклади створення віртуальної машини.

Рисунок 1.6 - Приклад Terraform конфігурації

Рисунок 1.7 - Приклад Pulumi конфігурації з використання Python 3

Базуючись на своєму досвіді праці з Terraform, в більшості випадків його можливостей достатньо и я не мав потреби в додатковій гнучкості, яку пропонує Palumi. У Terraform набагато більш активна спільнота, яка піклується про оновлення та виправлення багів. Також важливим плюсом Terraform є його популярність, він активно використовується для рішення реальних бізнес задач багатьма компаніями. Його розповсюдженість суттєво полегшує роботу з ним, більшість проблем вже було вирішено. Статистика запитів популярного ресурсу StackOverflow (рис 1.8) показує, що Terraform ще набирає популярність [10].

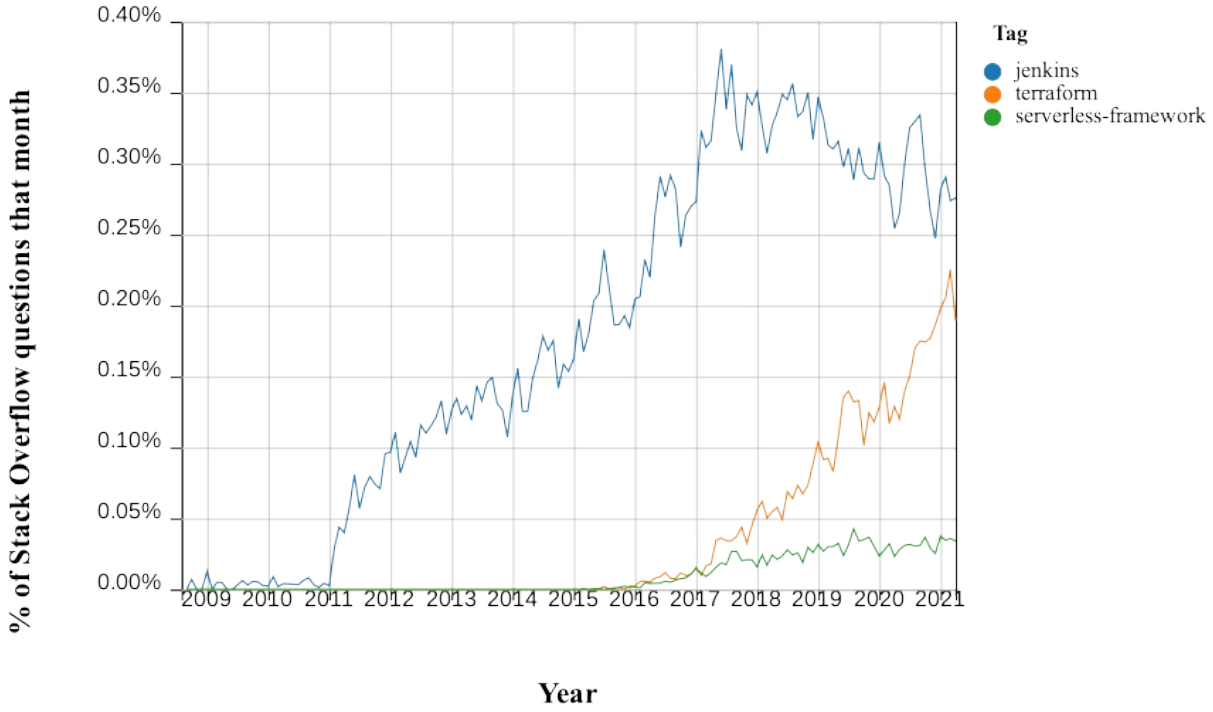

Рисунок 1.8 - Графік популярності технології на сайті StackOverflow

Це зумовило вибір Terraform як інструмента для роботи з хмарною інфраструктурою.

1.3.1 Робочий процес з Terraform

Робота з Terraform зазвичай включає в себе 3 основні етапи (рис. 1.9), які можна використовувати в системах безперервного розгортання.

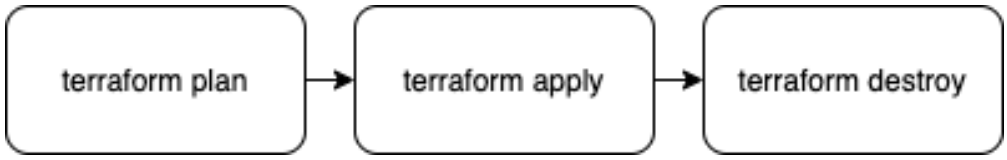

Рисунок 1.9 - Робочий процес з Terraform

Перший - це виконання команди ‘terraform plan’ вказавши все необхідні параметри для створення інфраструктури. Після виконання команди Terraform в текстовому виді згенерує зміни які будуть впроваджені, при першому розгортанні будуть виведені всі ресурси які описані в конфігураційних файлах та їх атрибути (рис. 1.10).

Рисунок 1.10 - Приклад виводу команди ‘terraform plan’

Другий етап – розгортання змін, які вивела команда ‘terraform plan’, для цього використовується команда ‘terraform apply’, при виконанні ресурси будуть створенні а їх конфігурація та ідентифікатори будуть зберігатись у спеціальному файлі ‘terraform.tfstate’ (рис 1.11). Terraform використовує цей файл для інкрементального оновлення та видалення ресурсів.

Рисунок 1.11 - Приклад виводу команди ‘terraform apply’

Третім етапом є видалення ресурсів, для цього використовується команда ‘terraform destroy’ (рис. 1.12) при цьому видаляються всі ресурси, які були розгорнуті в хмарі, а також при цьому відповідні ресурси видаляються з файлу‘terraform.tfstate’.

Рисунок 1.12 - Приклад виводу команди ‘terraform destroy

1.3.3 Використання Docker

Docker це пакет програм написаних на об’єктно-орієнтованій мові Go, який використовується для створення, оркестрації та розробки та роботи з контейнерами. Він має архітектуру клієнт-сервер (рис. 1.13).

Контейнером називають ізольоване середовище для певного програмного забезпечення, яке складається зазвичай з ядра операційної системи, та необхідних для запуску цього програмного забезпечення бібліотек. Через свій маленький розмір зазвичай контейнери швидко запускаються, що являється зручним для розробки програмного забезпечення. Контейнер складається з двох частин, файлової системи яка доступна лише для читання, та частини яку можна змінювати. Частину, що доступна лише для читання ще називають Docker образ (англ. image). Саме під час створення образу встановлюються все необхідні для запуску певних програм залежності.

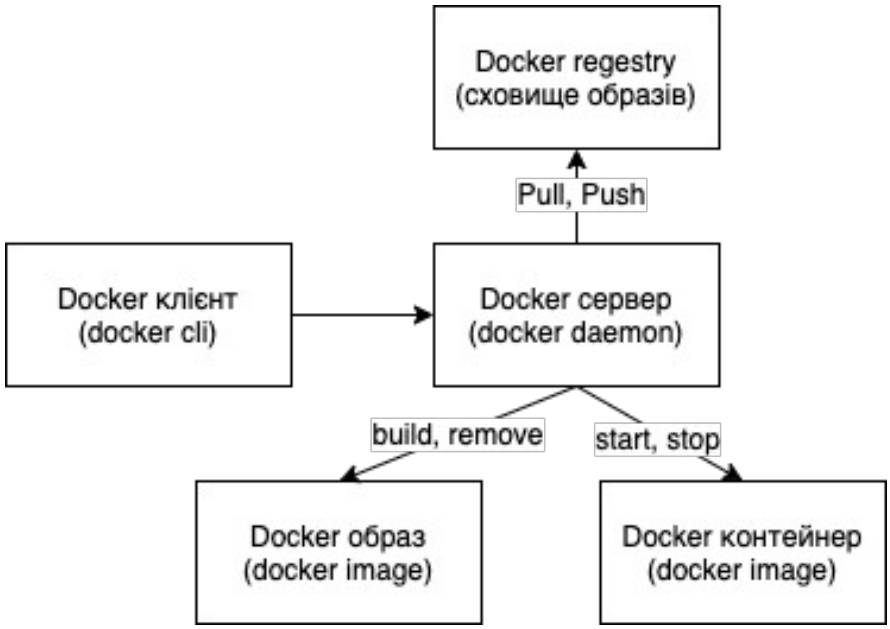

Рисунок 1.13 – Архітектура роботи програми Docker

На перший погляд, віртуальні машини та контейнери Docker можуть здатися схожими. Однак вони мають суттєві відмінності. Додатки, що працюють у віртуальних машинах, крім гіпервізора, потребують повного екземпляру операційної системи та будь-яких допоміжних бібліотек. Натомість контейнери мають спільний доступ до операційної системи хостом. Гіпервізор

можна порівняти з механізмом контейнерів (представлений як Docker на зображенні) в тому сенсі, що він керує життєвим циклом контейнерів. Важливою відмінністю є те, що процеси, що виконуються всередині контейнерів, подібні до власних процесів на хості, і не вносять ніяких накладних витрат, пов'язаних із виконанням гіпервізора. Крім того, програми можуть повторно використовувати бібліотеки та обмінюватися даними між контейнерами.

Для створення власного образу Docker необхідно перш за все створити файл ‘Dockerfile’ саме в ньому описуються базовий образ та кроки по встановленню бібліотек, та можливо запуск аплікації. Для цього потрібно виконати команду ‘docker build .’ в папці з файлом ‘Dockerfile’. Після того як образ буде зібрано (рис 2.2), ми готові до запуску Jenkins.

1.3.4 Огляд роботи з Visual Studio Code

Для роботи з кодом використовувалось середовище розробки Visual Strudio Code. Середовище розробки Visual Studio Code – це легковісний, але багатофункціональний редактор програмного коду, який підтримує роботу з більш ніж 30 мовами програмування та форматами файлів, включаючи C#, C++, JavaScript, TypeScript, Python, Go [11]. Це середовище має необхідні доповнення для роботи з Terraform, Jenkins та Docker, що значно пришвидшує створення необхідних конфігураційних файлів.

1.4 Постановка задачі дослідження

На підставі проаналізованих даних можна зробити висновок, що хмарні технології продовжують активно розвиватись, враховуючи потреби ринку. Вони дуже органічно вписуються в концепцію гнучких методології розробки програмного забезпечення та безперервної доставки. Це зумовить високий попит на спеціалістів, які володіють необхідними навичками для підтримки технологічних рішень зав’язаних на хмарних технологіях. Створення систем для автоматизації навчання та підготовки висококваліфікованих спеціалістів являється актуальною проблемою.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Створення облікового запису AWS

Створити обліковий запис в AWS можна безкоштовно, для цього потрібно мати поштову скриньку, номер телефону та кредитну чи дебетову картку. В AWS є спеціальна умова безкоштовного використання, яка дозволяє протягом року користуватися обмеженою кількістю сервісів в межах лімітів безкоштовно [12]. Для цієї роботи використовувався саме такий тип облікового запису. Важливо після реєстрації одразу налаштувати двофакторну автентифікацію, тому що можливі випадки зламів ненадійних паролів або публікацій паролів в публічний доступ. Після налаштування облікового запису необхідно створити нового користувача у AWS, та налаштувати доступ до цього користувача на локальному комп’ютері.

2.2 Архітектура прототипу системи автоматизації навчання

Прототип системи складається з трьох основних компонентів:

1. Terraform шаблони з описом завдання
2. Опис конфігурації Jenkins сервера
3. Декларативний опис конфігурації Jenkins конвеєра

При чому Jenkins конвеєр має універсальну структуру, його можна буде використовувати для декількох завдань, необхідно лише замінити декілька змінних. Jenkins сервер також достатньо конфігурувати лише один раз і потім використовувати вже готовий варіант. За необхідності можна модифікувати ‘Dockerfile’, де описано конфігурацію Jenkins, встановити додаткове програмне забезпечення тощо. Шаблони являються унікальними для кожного завдання проте, можливо будувати їх з використанням модульної системи, щоб повторно використати деякі фрагменти коду.

2.3 Алгоритм перевірки цілісності хмарної інфраструктури

Основною задачею системи є виявлення відмінностей між конфігурацією інфраструктури вирішеного завдання та невирішеного завдання. Для цього було використано Terraform. Головна ідея полягає в тому, щоб спочатку створити інфраструктуру вирішеного завдання, при цьому Terraform створить мета опис інфраструктури, яка була розгорнута в хмарі у ‘terraform.tfstate’ файлі. Ми робимо копію цього файлу, як еталонний стан інфраструктури. Потім ми видаляємо деякі ресурси, робота з якими являється змістом завдання. При цьому Terraform змінить опис інфраструктури, яка буда створена у ‘terraform.tfstate’ файлі. Ми також робимо копію цього файлу, як стан інфраструктури повністю невиконаного завдання. Таким чином у нас є еталонний опис стану повністю вирішеного, опис стану невирішеного завдання та створенні ресурси для завдання. Для перевірки завдання нам потрібно підмінити файл ‘terraform.tfstate’ на еталонний, де завдання є повністю вирішене, далі Terraform перевірить які ресурси існують у файлі ‘terraform.tfstate’ і є справді розгорнуті в хмарі та сформує звіт. Критерієм виконаного завдання є відсутність змін між еталонним описом та станом розгорнутої інфраструктури в хмарі.

Для видалення ресурсів за змістом завдання використовувався вбудований функціонал Terraform: variables та count. За допомогою ‘variables’ ми можемо передавати параметри в наші файли конфігурацій інфраструктури, ‘count’ у свою чергу дозволяє вибирати кількість ресурсів яку потрібно створити. Поєднання ‘variables’ та ‘count’ дає змогу контролювати умови при яких ресурс буде створений або ні. Нижче наведено приклад використання (рис. 2.1).

Рисунок 2.1 - Використання умов для створення ресурсів у Terraform

В цьому прикладі використовуються так звані умовні вирази [13] (англ. Conditional Expressions), якщо змінна ‘create_instance’ дорівнює істині, то ресурс буде створений, якщо змінна ‘create_instance’ не дорівнює істині, то ресурс створюватись не буде. При чому якщо спробувати оновити

конфігурацію ресурсів, які вже розгорнуто в хмарі, то Terraform видалить цей ресурс. Цей підхід лежить в основі створення завдання для перевірки.

2.4 Конфігурація Jenkins

Розгортання Jenkins проводили локально з використанням контейнерів та програми Docker [14]. Для створення власної інсталяції Jenkins було створено власний Docker образ з необхідними для роботи залежностями в основі якого лежить офіційний образ від творців Jenkins. До нього було встановлено Jenkins плагіни для роботи за декларативними конвеєрами, Terraform та aws-cli для виконання певних конфігурацій в AWS.

Для того щоб запустити Jenkins необхідно створити образ для майбутнього контейнера. Для цього потрібно виконати команду ‘docker build -t jenkins_node .’ в папці з файлом ‘Dockerfile’. Після того як образ буде зібрано (рис 2.2), ми готові до запуску Jenkins.

```
➜  jenkins git:(main) ✗ docker build -t jenkins_node .
[+] Building 1.9s (11/11) FINISHED
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 37B                                           0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [internal] load metadata for docker.io/jenkins/jenkins:lts                1.7s
 => [auth] jenkins/jenkins:pull token for registry-1.docker.io                0.0s
 => [1/6] FROM docker.io/jenkins/jenkins:lts@sha256:3a441b1bcd2ce630b7bad3486e  0.0s
 => CACHED [2/6] RUN wget -O terraform.zip "https://releases.hashicorp.com/ter  0.0s
 => CACHED [3/6] RUN wget -O terragrunt https://github.com/gruntwork-io/terrag  0.0s
 => CACHED [4/6] RUN apt update && export DEBIAN_FRONTEND=noninteractive && ap  0.0s
 => CACHED [5/6] RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_6  0.0s
 => CACHED [6/6] RUN jenkins-plugin-cli --plugins pipeline-model-definition sc  0.0s
 => exporting to image                                                        0.0s
 => => exporting layers                                                       0.0s
 => => writing image sha256:878cb6b6f5e29a9f1b94c5b76f79a2374804eb1765ebf7775f  0.0s
 => => naming to docker.io/library/jenkins_node                               0.0s
```

Рисунок 2.2 - Збірка контейнера

Запуск контейнера виконується командою ‘docker run -p 8080:8080 -p 50000:50000 -v $HOME/.aws/:/var/jenkins_home/.aws/ -v

jenkins_home:/var/jenkins_home jenkins_node’ (рис 2.3), після чого нам буде локально доступний веб-інтерфейс Jenkins за посиланням http://localhost:8080.

```
➜  jenkins git:(main) ✗ docker run -p 8080:8080 -p 50000:50000 -v $HOME/.aws/:/var/je
nkins_home/.aws/ -v jenkins_home:/var/jenkins_home jenkins_node
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
2021-05-18 22:06:17.781+0000 [id=1]     INFO    org.eclipse.jetty.util.log.Log#initia
lized: Logging initialized @594ms to org.eclipse.jetty.util.log.JavaUtilLog
2021-05-18 22:06:18.025+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginnin
g extraction from war file
2021-05-18 22:06:19.593+0000 [id=1]     WARNING o.e.j.s.handler.ContextHandler#setCon
textPath: Empty contextPath
2021-05-18 22:06:19.730+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#doSta
```

Рисунок 2.3 - Запуск контейнера

Після переходу за посиланням потрібно ввести пароль адміністратора для того щоб розблокувати Jenkins. Знайти цей пароль можна в виводі команди запуску контейнера (рис. 2.4).

```
2021-05-18 22:06:31.099+0000 [id=31]    INFO    jenkins.install.SetupWizard#init:

*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password gene
rated.
Please use the following password to proceed to installation:

b0d3f90bd5384c55a7f815a8904ea984

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************

2021-05-18 22:06:39.803+0000 [id=31]    INFO    jenkins.InitReactorRunner$1#onAttaine
```

Рисунок 2.4 - Пароль для ініціалізації Jenkins

Далі необхідно встановити рекомендовані плагіни, та користувача. Потім створюємо новий конвеєр (pipeline) використовуючи код із Додатку Б.

2.5 Структура Jenkins конвеєру

Основними задачами конвеєру є створення тестової інфраструктури у хмарному середовищі AWS, підготувати завдання для студента, виконати перевірку що до правильності виконання, видалення тестової інфраструктури у хмарному середовищі AWS. Конвеєр побудований таким чином що дозволяє необмежену кількість разів проводити перевірку правильності виконання завдання, проте має функцію обмеження часу, за який студент повинен виконати завдання, якщо студент не встигає, то вся тестова інфраструктура буде видалена, а завдання не зараховане. Схематично конвеєр складається з 5 блоків (рис. 2.5). Проте блок №4 може бути повторно викликаний необмежену кількість разів.

Рисунок 2.5 - Схема конвеєру для перевірки завдання

У веб-консолі Jenkins конвеєр зображено трохи по іншому (рис. 2.6), тому що вона не підтримує правильне відображення складних конвеєрів з паралельним виконанням кроків, або необмежену кількістю послідовних блоків.

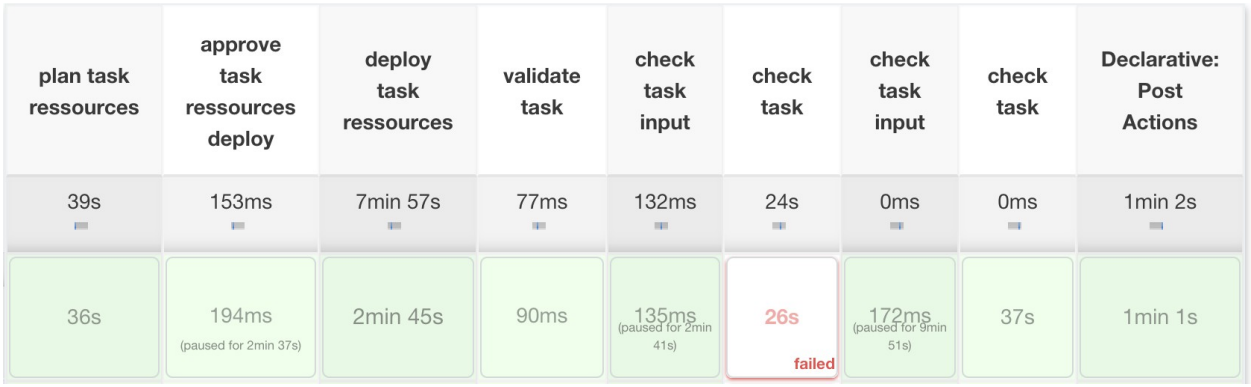

Рисунок 2.6 - Зображення конвеєру в веб-консолі Jenkins

2.6 Структура завдання з конфігурації доступу до віртуальної машини

На даний момент існує тільки один приклад завдання, яке може перевіряти конвеєр: це налаштування доступу до віртуальних машин в AWS. В рамках цього завдання створюється вся необхідна інфраструктура включаючи, створення віртуальних мереж та підмереж.

Суть задачі полягає в тому щоб студент ознайомився з документацією AWS, та створив робочу конфігурацію за певну кількість кроків в нашому випадку це 2 кроки. До кроків які необхідно оформити, використаємо наш алгоритм для перевірки цілісності інфраструктури (рис. 2.7).

Рисунок 2.7 - Основні ресурси завдання

Тут було використано прийом для створення умови для створення ресурсів, для їх видалення щоб створити зміст завдання. Таким чином ми маємо два виконання команди ‘terraform apply’ присвоївши змінній ‘`deploy_task_resources`’ істину. І для ці ресурсів створюються і потрапляють у ‘terraform.tfstate’ файл. Ми робимо копію цього файлу як еталон для порівняння та запускаємо ‘terraform apply’ другий раз. При цьому ‘`deploy_task_resources`’ виставляємо в не істину, і Terraform видалить ці ресурси. Після чого завдання готове до виконання студентом.

Коли завдання виконане для його перевірки ми виконуємо ‘terraform plan’ присвоївши змінній ‘`deploy_task_resources`’ істину та попередньо підмінивши ‘terraform.tfstate’ на копію з минулого кроку. Якщо Terraform видає, що додаткові зміни не потрібні то задача рахується виконаною. Повний конфігураційних файлів для завдання знаходить в Додатку В.

ВИСНОВКИ

В рамках роботи було розглянуто хмарні технології, їх структуру та деталі реалізації. Проаналізовано дослідження компанії Garthner, щодо лідерів на ринку, на основі чого було обрано хмарного провайдера AWS для виконання практичної частини.

Дано визначення поняттям гнучких методологій та безперервної доставки. Проведено огляд інструментів для виконання роботи та вибрано такі інструменти як Jenkins та Terraform для реалізації бізнес логіки системи.

Розроблено алгоритм для перевірки цілісності інфраструктури. На основі якого створено прототип системи для автоматизації навчання та підготовки спеціалістів в області хмарних технологій з використанням продуктів з відкритим програмним кодом Docker, Jenkins та Terraform.

Перелік питань поставлених для виконання роботи було опрацьовано у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Weintraub, E., Cohen, Y. Cost Optimization of Cloud Computing Services in a Networked Environment. International Journal of Advanced Computer Science and Applications. 2015. Vol. 6, No. 4.

2. Meikshan, V. I., Teslya, N. B., Istratova, E. E., et al. Cost Optimization for Data Storage Using Cloud Technologies: Frontier Information Technology and Systems Research in Cooperative Economics: / за ред. A. V Bogoviz, A. E. Suglobov, A. N. Maloletko, et al. Cham, Springer International Publishing, 2021.

3. The NIST Definition of Cloud Computing: Application Performance Management (APM) in the Digital Enterprise. Elsevier, 2017.

4. Wright, D., Gill, B., Ji, K., et al. Magic Quadrant for Cloud Infrastructure and Platform Services: [Електронний ресурс] - Режим доступу до ресурсу: https://www.gartner.com/doc/reprints?id=1-1ZDZDMTF&ct=200703&st=

5. Dingsøyr, T., Dybå, T., Moe, N. B. Agile Software Development: An Introduction and Overview: Agile Software Development: An Introduction and Overview: / за ред. T. Dingsøyr, T. Dybå, N. B. Moe. Berlin, Heidelberg, Springer, 2010.

6. Humble, J., Read, C., North, D. The deployment production line: *AGILE 2006 (AGILE'06)*, 06. C. 6 pp. – 118.

7. Humble, J., Farley, D. Continuous delivery: reliable software releases through build, test, and deployment automation: Upper Saddle River, NJ: Addison-Wesley, 2010.

8. Kim, G., Debois, P., Willis, J., et al. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations: Portland, OR: IT Revolution Press, 2016.

9. Pulumi - Modern Infrastructure as Code: [Електронний ресурс] - Режим доступу до ресурсу: https://www.pulumi.com/ (Дата звернення: 18.05.21).

10. Stack Overflow Trends: [Електронний ресурс] - Режим доступу до ресурсу: https://insights.stackoverflow.com/trends?tags=r%2CstatisticsStack Overflow Trends, (Дата звернення: 16.05.21).

11. Documentation for Visual Studio Code: [Електронний ресурс] - Режим доступу до ресурсу: https://code.visualstudio.com/docs (Дата звернення: 18.05.21).

12. AWS Free Tier: [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/free (Дата звернення: 10.05.21).

13. Conditional Expressions - Configuration Language - Terraform by HashiCorp: [Електронний ресурс] - Режим доступу до ресурсу: https://www.terraform.io/docs/language/expressions/conditionals.html (Дата звернення: 18.05.21).

14. Docker Documentation | Docker Documentation: [Електронний ресурс] - Режим доступу до ресурсу: https://docs.docker.com/ (Дата звернення: 18.05.21).