МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри,
д-р техн. наук, проф.
_____ І. М. Журавська
« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Кластерна серверна система на базі Raspberry PI та Terragrunt

Спеціальність «Комп'ютерна інженерія»
123 – КМР.1 – 605.21710515

Студент _____С.Ю. Кушнір
підпис
«__» _____ 202__ р.

Керівник канд. фіз.-мат. наук, доцент _____С.В. Пузирьов
підпис
«__» _____ 202__ р.

Миколаїв – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОЗУ Оперативний запам’ятовуючий пристрій

ПЗ Програмне забезпечення

КВ Кілобайт

API Application Programming Interface ()

ARM ARM processor architecture family

CI/CD Continuous integration and continuous delivery process

CLI Command-line interface

EKS AWS Elastic Kubernetes Service

GB Гігабайт

GUI Graphical user interface

IAM AWS Identity and Access Management

IDE Integrated development environment

IaC Infrastructure as code

ВСТУП

Невід’ємна частина будь якої великої сучасної IT-інфраструктури – це інструменти IaC. Infrastructure as code. Інфраструктура як код (IaC) — це управління і розгортання інфраструктури за допомогою коду, а не за допомогою ручних процесів. За допомогою IaC створюються файли конфігурації, які містять специфікації інфраструктури, що полегшує редагування та розповсюдження конфігурацій. Це також гарантує, що кожного разу розгортається одне й те саме середовище

Актуальність: кластерні серверні системи вже стали фактично системами по-замовчування для сучасних IT компаній. Будь який великий IT продукт працює в тому чи іншому ступіні на основі кластерних систем. Основною їх перевагою є їх відмовостійкість, але одним з недоліків вважається складність налаштування, іншим - складність відтворення, також, актуальна проблемою є перенос інфраструктури з одного місця на інше, або частини інфраструктури. Система, розроблена протягом виконання дипломної роботи призначена за для того, щоб вирішити проблеми пов’язані з цими недоліками. За допомогою налаштування Raspberry PI як екзекютору для виконання коду та написання коду Terroform та обгортання його у Terragrunt будуть вирішені вище зазначені недоліки. Крім того, у період постійних відключень електроенергії стане у нагоді екзекютор коду, який має низьке енергоспоживання та може бути заживлений від павер банку.

Мета: розробити програмно-апаратний модуль на основі одноплатного комп’ютера, як екзекютору коду, та інструментів Infrastructure as Code для створення серверної кластерної системи.

Об’єкт: методи та технології, що використовуються у кластерних серверних системах, інструменти Infrastructure as code та екзекютори коду.

Предмет: апаратне забезпечення CI/CD екзекюторів для виконання коду та програмне забезпечення Infrastructure as code.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Дослідити інструменти Infrastructure as code, порівняти їх.
2. Проаналізувати екзекютори для виконання коду, їх архітектури та готові рішення від CI/CD систем
3. Розробити код та структуру каталогів для виконання коду Terragrunt.
4. Провести тести написаного коду та налаштувати мікрокомп’ютеру для виконання коду.
5. Приєднати одноплатний комп’ютер до CI/CD системи та автоматизувати процес розгортання та оновлення кластерної системи.
6. Провести тестування.

Результатом розробки має бути код у форматі HCL для Terraform, структура каталогів для Terragrunt, а також рекомендації для налаштування одноплатного комп’ютера.

Практичне значення програмно-апаратного модулю полягає у спрощенні процессу разгортання, перенесення кластерної IT інфраструктури або перенесення частини кластерної системи.

Дипломна робота складається з наступних частин: вступу, чотирьох розділів, висновку, списку використаної літератури, додатків.

Перший розділ – дослідження предмету та об’єкту. Визначення основних характеристик та відмінностей інструментів IaC.

Другий розділ присвячено розробці коду мовою HCL для Terraform та локальному тестуванню коду.

У третьому розділі буде розглянуто архітектуру мікрокомп’ютера Raspberry PI, встановленню Linux Based OS, налаштуванню Raspberry PI, приєднанню мікрокомп’ютера до CI/CD системи.

Четвертий розділ присвячений проведеню тестування, виконання коду з CI/CD системи та аналізу отриманих результатів.

Методи дослідження. У дипломній роботі для вирішення наукових завдань використані наступні методи досліджень:

експертної оцінки – при виборі теми і постановці мети дослідження та аналізі результатів;

дедукції – при виборі напрямків дипломного дослідження;

системного аналізу – при розробці технології наукових досліджень;

Апробація магістерської роботи відбулася під час Всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів **«Інформаційні технології та інженерія»**, (додаток Б) у Чорноморському національному університеті імені Петра Могили (Миколаїв, 7-10 лютого 2023 р.).

Публікації. За результатами кваліфікаційної роботи створено публікацію у збірнику матеріалів Всеукраїнської конференції [1].

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Описання технології IaC

Інфраструктура як код (IaC) – це управління інфраструктурою та її надання за допомогою коду, а не ручних процесів.

За допомогою IaC створюються конфігураційні файли, що містять специфікації інфраструктури, що спрощує редагування та розповсюдження конфігурацій. Це також гарантує, що щоразу розгортається те саме середовище. Систематизуючи та документуючи специфікації конфігурації, IaC (рис. 1.1) допомагає керувати конфігурацією та допомагає уникнути недокументованих, нерегламентованих змін конфігурації. [1]

Рисунок 1.1 Схема ідеї покладеної у базис інструментів IaC

Контроль версій є важливою частиною IaC, і конфігураційні файли повинні знаходитися під контролем вихідного коду, як і будь-який інший файл вихідного коду програмного забезпечення. Розгортання

інфраструктури у вигляді коду також означає, що можна розділити інфраструктуру на модульні компоненти (рис. 1.2), які можна комбінувати різними способами за допомогою автоматизації.

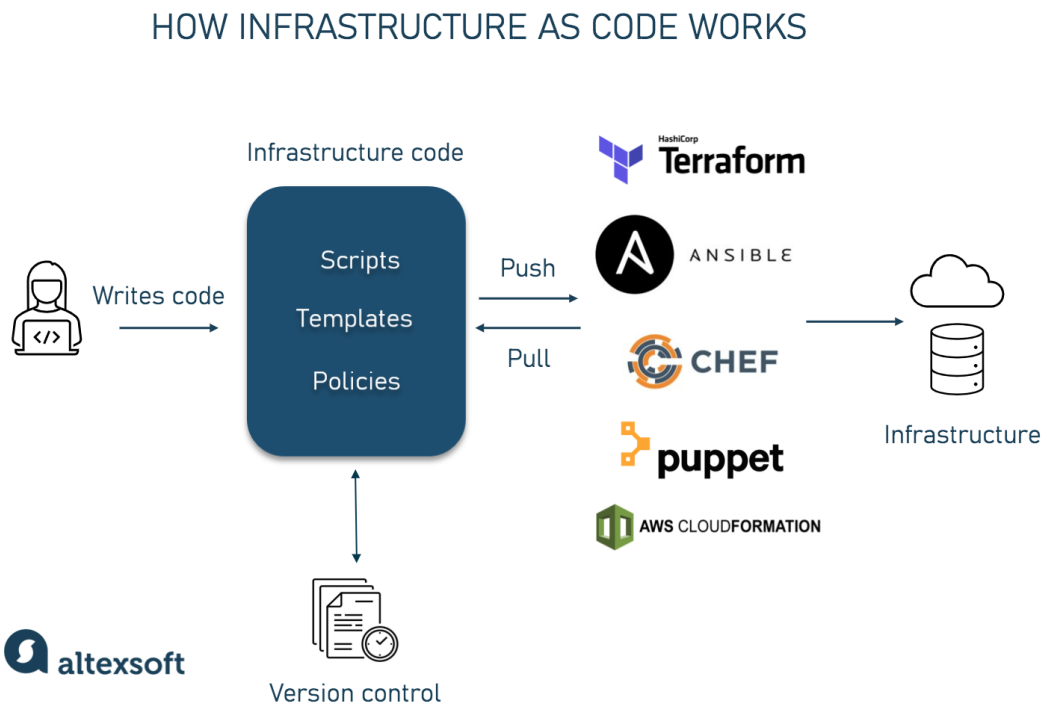


Рисунок 1.2 Схема роботи інструментів IaC

Автоматизація підготовки інфраструктури за допомогою IaC означає, що розробникам не потрібно вручну виділяти та керувати серверами, операційними системами, сховищем та іншими компонентами інфраструктури щоразу, коли вони розробляють або розгортають програму. Кодифікація інфраструктури дає шаблон для підготовки, і хоча це ще можна виконати вручну, інструмент автоматизації, такий як CI/CD системи можуть зробити це автоматично. Є два типи підходів при викростанні іструментів IaC:

1. декларативний підхід

2. імперативний підхід

Декларативний підхід (рис. 1.3) включає список реєстрації штатних систем із системних об'єктів, які стають доступними для управління. Тобто, при декларативному підході описується те, що необхідно отримати у результаті виконання коду.

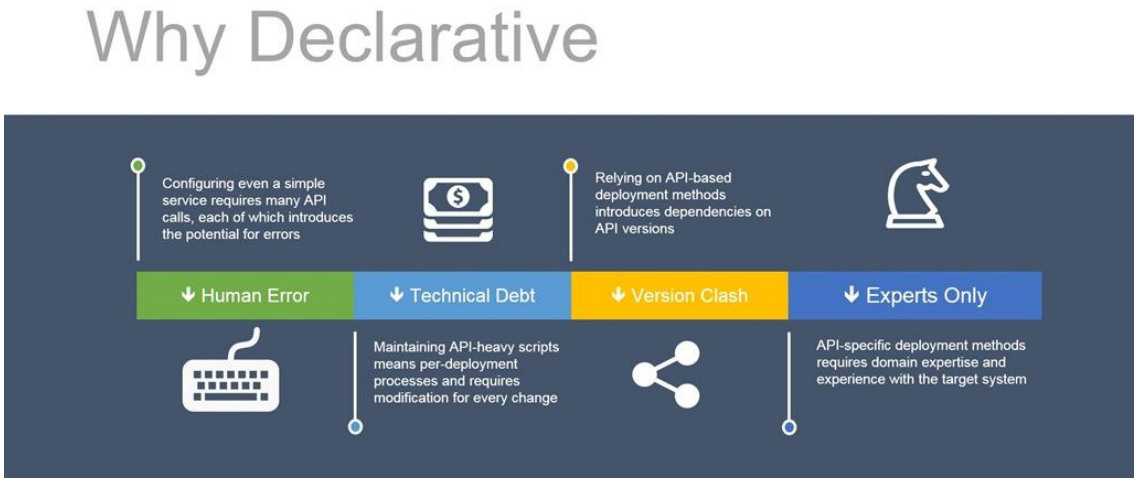


Рисунок 1.3 Схема декларативного підходу

Імперативний підхід - це підхід за яким описується те, як необхідно виконувати ту, чи іншу задачу, а також важливий порядок їх виконання.

Багато інструментів IaC використовують декларативний підхід та автоматично розгортають інфраструктуру. У випадку використання імперативного підходу, замість бажаного результату необхідно вказати та разібратись яким чином та у якому порядку інфраструктура буде змінюватись.

Підготовка інфраструктури історично була трудомістким та дорогим ручним процесом. Тепер управління інфраструктурою перейшло від фізичного обладнання в центрах обробки даних, хоча воно все ще може

бути компонентом організації, до віртуалізації, контейнерів та хмарних обчислень.

Завдяки хмарним обчисленням кількість компонентів інфраструктури зросла, щодня випускається все більше програм, і інфраструктура повинна мати можливість часто розгортатися, масштабуватись та вимикатися. Без інструментів IaC стає дедалі важче керувати масштабом сучасної інфраструктури.

IaC може допомогти організації керувати потребами IT-інфраструктури, а також підвищити узгодженість та зменшити кількість помилок та ручне налаштування. [1]

Переваги:

1. Зниження ціни
2. Збільшення швидкості розгортання
3. Зменшити кількість помилок
4. Поліпшення узгодженості інфраструктури
5. Усунення дрейфу конфігурації

IaC – важлива частина впровадження практики DevOps та безперервної інтеграції/безперервної доставки (CI/CD). IaC знімає велику частину роботи з підготовки розробників, які можуть виконати сценарій, щоб підготувати свою інфраструктуру до роботи. Таким чином, розгортання застосунків не затримується в очікуванні інфраструктури, а системні адміністратори не керують ручними трудомісткими процесами. CI/CD спирається на постійну автоматизацію та безперервний моніторинг протягом усього життєвого циклу програми, від інтеграції та тестування до постачання та розгортання.

Щоб середовище було автоматизованим, воно має бути послідовним. Автоматизація розгортання програм не працює, коли група розробників

розгортає програми або налаштовує середовища одним способом, а група експлуатації іншим. Об'єднання команд розробки та експлуатації за допомогою підходу DevOps (рис 1.4) призводить до меншої кількості помилок, ручних розгортань та невідповідностей.

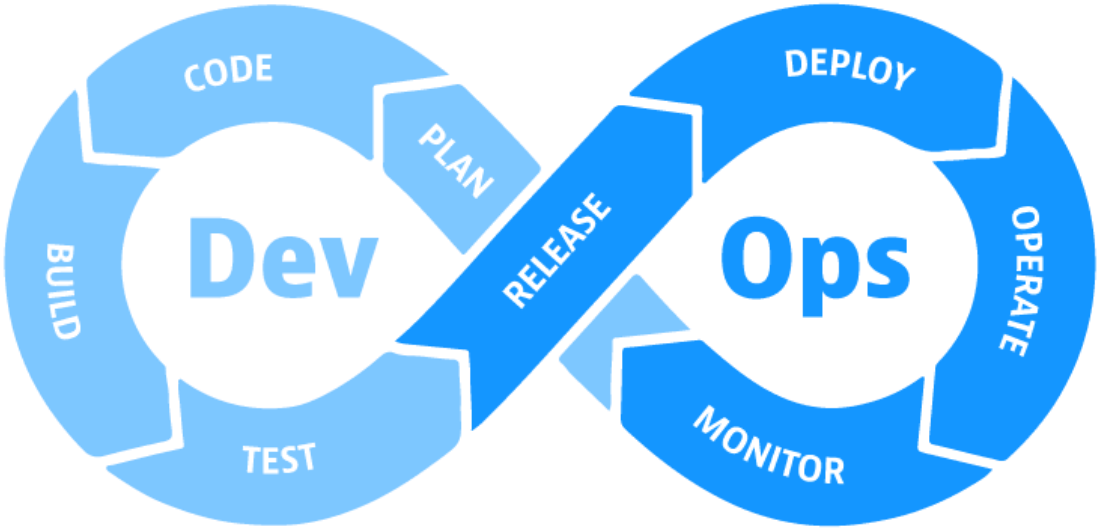


Рисунок 1.4 DevOps підхід до розробки

IaC допомагає узгодити розробку та експлуатацію, оскільки обидві команди можуть використовувати те саме опис розгортання програми, підтримуючи підхід DevOps. Один і той же процес розгортання слід використовувати для кожного середовища, включаючи робоче середовище. IaC створює те саме середовище кожного разу, коли воно використовується. IaC також позбавляє необхідності підтримувати окремі середовища розгортання з унікальними конфігураціями, які не можна відтворити автоматично, і забезпечує узгодженість робочого середовища.

Найкращі практики DevOps також застосовуються до інфраструктури IaC. Інфраструктура може проходити через той же CI/CD процес, що й додаток під час розробки програмного забезпечення, застосовуючи до коду інфраструктури ті ж таки тести та контроль версій.

1.2 Описання екзекюторів коду

Одна з частин дипломної роботи це побудова універсального екзекютора на основі одноплатного комп’ютера Raspberry PI 4.

Дипломну роботу виконано на основі прикладу Gitlab екзекютору. GitLab Runner це програма, яка працює з GitLab CI/CD для запуску завдань у конвеєрі.

GitLab Runner можна встановити в інфраструктурі, якою керується. Якщо це зробити, слід встановити GitLab Runner на машині, відмінної від тієї, на якій розміщений екземпляр GitLab, з міркувань безпеки та продуктивності. Коли використовуються окремі машини, можуть бути різні операційні системи та інструменти, такі як Kubernetes або Docker на кожній з них.

GitLab Runner має відкритий вихідний код та написаний на Go. Його можна запустити як один двійковий файл, ніяких вимог до мови не потрібно. [20]

Також, можна встановити GitLab Runner на кілька операційних систем, що підтримуються. Також можна встановити на будь яку платформу даний екзекютор, скомпілювавши код Go.

GitLab Runner також може працювати всередині контейнера Docker або у кластері Kubernetes.

1.3Існуючі інструменти IaC

Для досягнення IaC часто можна використовувати інструменти автоматизації сервера та керування конфігурацією. Існують також рішення спеціально для IaC. [2]

Найпопулярніші інструменти IaC:

1. Chef
2. Puppet
3. Ansible
4. Saltstack
5. Terraform
6. AWS CloudFormation

Нижче кожен інструмент буде розглянутий більш детально та буде проаналізовано основні переваги та недоліки кожного інструменту.

1.4 Chef

Chef – це інструмент автоматизації, що дозволяє визначити інфраструктуру як код. Інфраструктура як код (IaC) означає, що керування інфраструктурою здійснюється шляхом написання коду (автоматизація інфраструктури), а не з використанням ручних процесів. Його також можна назвати програмованою інфраструктурою. Chef використовує суто Ruby, предметно-орієнтовану мову (DSL) для написання системних конфігурацій. Нижче наведено типи автоматизації, що виконуються Chef, незалежно від розміру інфраструктури:

1. Конфігурація інфраструктури
2. Розгортання застосунків
3. Віддалене керування конфігурацією

Chef, як і у Puppet працює за архітектурою Master-Slave, Chef має архітектуру Client-Server. Але Chef має додатковий компонент під назвою Workstation. Діагарама роботи інструменту Chef (рис 1.6).

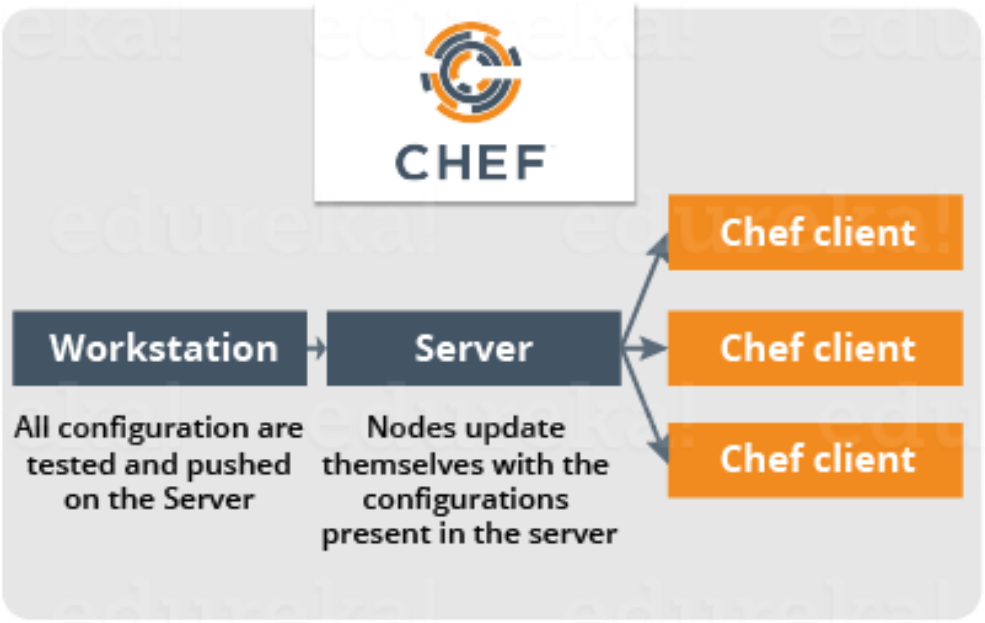


Рисунок 1.5 Діаграма роботи Chef

У Chef вузли динамічно оновлюються відповідно до конфігурацій на сервері. Це називається конфігурацією на запит, що означає, що не потрібно виконувати навіть одну команду на сервері Chef, щоб передати конфігурацію на вузли, вузли автоматично оновляться за допомогою конфігурацій, присутніх на сервері. [4]

Основні переваги Chef:

1. Chef підтримує кілька платформ, таких як AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows та Ubuntu. Додаткові клієнтські платформи включають Arch Linux, Debian та Fedora.
2. Chef можна інтегрувати з хмарними платформами, такими як Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure та Rackspace для автоматичної підготовки та налаштування нових машин.
3. У Chef є активна, розумна та швидкозростаюча підтримка спільноти.

4. Через зрілість та гнучкість Chef його використовують такі гіганти, як Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Університет Маршалла, Сократа, Міннесотський університет, Уортонська школа. Університет Пенсільванії, Бонобос, Splunk, Citi, DueDil, Disney і Cheezburger.

Основні недоліки Chef:

1) Імперативний підхід
2) Відсутність файлу стану або будь якого його аналогу
3) Менший фукціонал в порівнянні з Terraform
4) Складність написання коду, необхідні знання Ruby

1.5 Puppet

Puppet - це інструмент, який допомагає керувати та автоматизувати налаштування серверів. При використанні Puppet, визначається бажаний стан систем у інфраструктурі, якими необхідно керувати. [5]

Написання коду інфраструктури відбувається доменно-специфічною мовою Puppet (DSL) - Puppet Code - який можна використовувати з широким спектром пристроїв та операційних систем. Puppet Code є декларативним, що означає, що описується бажаний стан систем, а не кроки, необхідні для його досягнення.

Puppet автоматизує процес переведення систем у стан та підтримує системи у їх стані, описаному у коді. Puppet робить це через основний сервер Puppet та агент Puppet. Основний сервер Puppet – це сервер, на якому зберігається код, який визначає бажаний стан. Агент Puppet переводить код у команди, а потім виконує його у вказаних на сервері системах, що називається запуском Puppet. На наведеній нижче діаграмі показано (рис 1.6), як працює архітектура сервер-агент запуску Puppet.

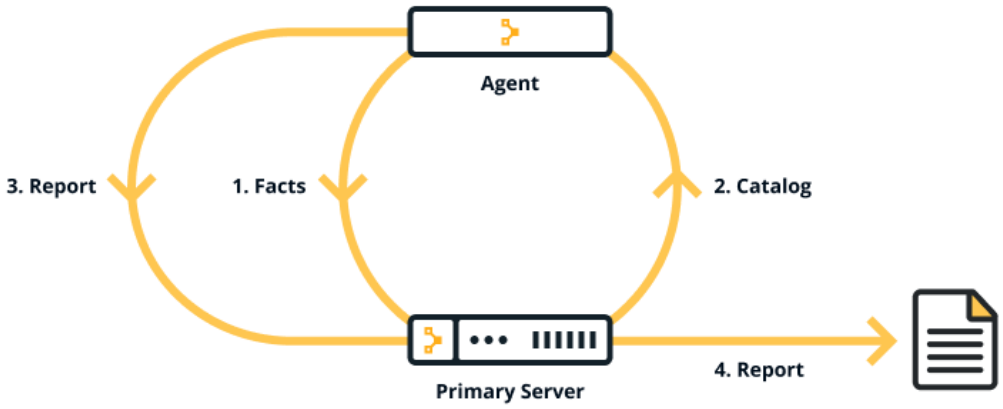


Рисунок 1.6 Архітектура сервер-агент запуску Puppet

Основні переваги Puppet:

1) Послідовність.
2) Автоматизація.

Основні недоліки:

1) Менший фукціонал в порівнянні з Terraform
2) Складна клієнт-серверна конфігурація
3) Специфічна мова програмування, яка не дуже широко розвинена

1.6 Ansible

Ansible - це інструмент IaC, який може прискорити розгортання програм. Ansible — це проект спільноти з відкритим вихідним кодом, який допомагає організаціям автоматизувати підготовку, керування конфігурацією та розгортання програм. В Ansible можна створювати плейбуки (написані мовою конфігурації YAML), щоб вказати необхідний стан для інфраструктури, Ansible виконує підготовку інфраструктури крок за кроком, описаним в плейбуці. [6]

Ansible отримує доступ до серверів за допомогою протоколу SSH. Архітектура Ansible (рис 1.7) приведено нижче.

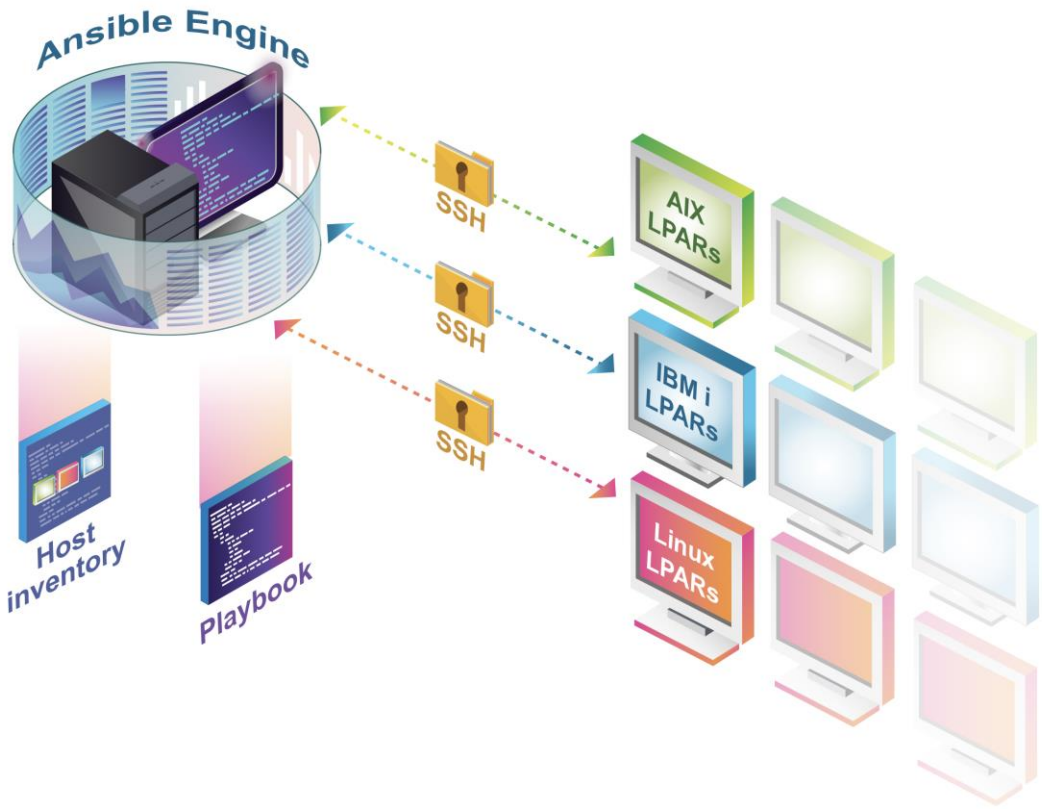


Рисунок 1.7 Архітектура Ansible

Основними перевагами Ansible є простота та зручність використання. Він також приділяє велику увагу безпеці та надійності з мінімальною кількістю рухомих частин. Він використовує OpenSSH для транспорту (з іншими транспортами та режимами вилучення в якості альтернативи) і використовує мову, яка легко читається, розроблена для швидкого початку роботи без тривалого навчання.

Основним недоліком вважається те, що Ansible меншою мірою призначкний для роботи як інструмент Infrastruckture as Code та більшою мірою, як інструмент автоматизації конфігурації, тому і має всі недоліки,

такі як імперативний підхід, відсутність файлу стану, відсутність автоматичного повернення до стану, описаному у коді, тощо.

1.7 Saltstack

Saltstack - також відомий як Salt, є інструментом керування конфігурацією та оркестрування (рис 1.8). Він використовує центральний репозиторій для надання нових серверів та іншої IT-інфраструктури, для внесення змін до існуючих та встановлення програмного забезпечення в IT-середовищах, включаючи фізичні та віртуальні сервери, а також у хмарі. SaltStack автоматизує повторювані завдання адміністрування системи та розгортання коду, за винятком ручних процесів, що може зменшити кількість помилок, що виникають при налаштуванні систем IT-організаціями. Salt використовується в організаціях DevOps, оскільки він отримує код розробника та інформацію про конфігурацію з центрального репозиторію коду, такого як GitHub або Subversion, і віддалено передає контент на сервери. [7] Користувачі Salt можуть писати власні скрипти та програми, а також завантажувати готові конфігурації, внесені іншими користувачами в загальнодоступний репозиторій.

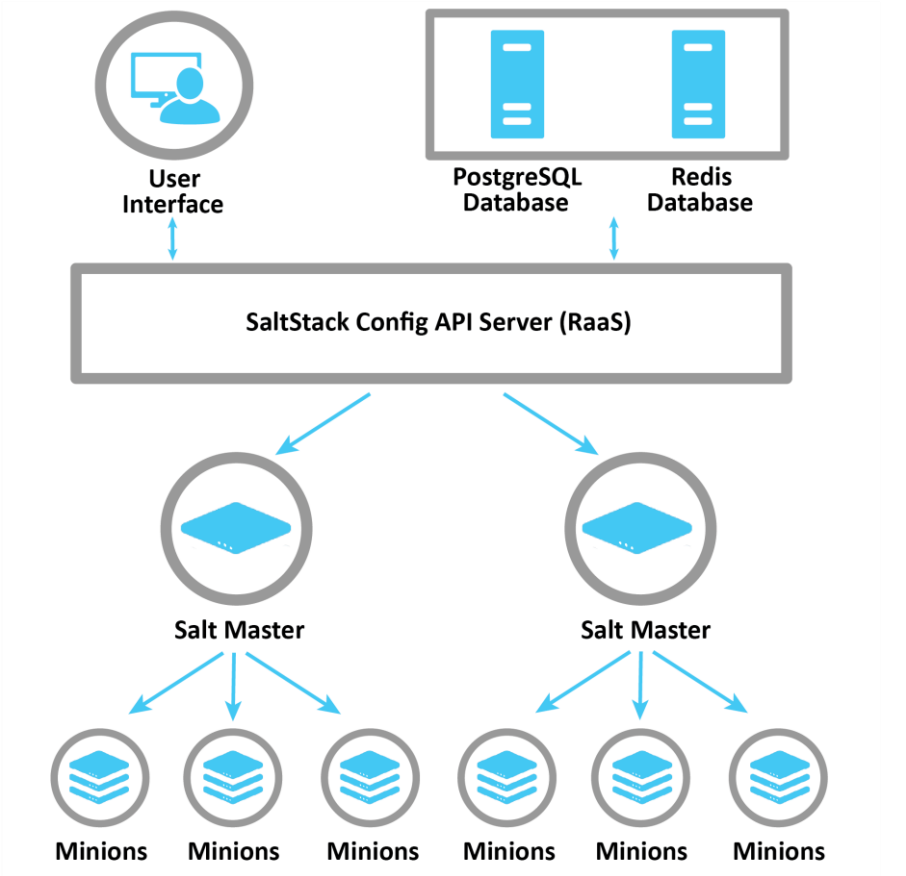


Рисунок 1.8 Архітектура конфігурації SaltStack

Технологія, що лежить в основі Salt та SaltStack Enterprise, має сильні та слабкі сторони залежно від навичок користувача, а також від розгортання, в якому діятиме користувач. Salt керується подіями та є модульним, гарантуючи, що IT-компоненти, що знаходяться під його контролем, зберігають свій цільовий стан. Він надає механізм зовнішньої аналітики, тим самим додаючи інтелект реагування на події, зокрема сторонні. Систему можна налаштувати у багаторівневій конфігурації, коли один minion контролює інших, щоб підвищити надмірність та балансування навантаження.

Salt написаний та використовує Python, який вже встановлений практично у всіх системах Linux. Однак графічний інтерфейс SaltStack

Enterprise не відрізняється широким набором функцій, тому більшість користувачів виконують завдання за допомогою інструмента командного рядка. Крім того, цільові стани не можна перевіряти у визначеному порядку. Це обмеження зменшує можливість програмування залежностей між системами.

1.8 HashiCorp Terraform

HashiCorp Terraform — це інструмент «інфраструктура як код», який дозволяє визначати як хмарні, так і локальні ресурси (рис. 1.8) в зручних для читання файлах конфігурації, які можна версіонувати, повторно використовувати і ділитися ними. Можна використовувати узгоджений робочий процес для підготовки та управління всією інфраструктурою протягом усього її життєвого циклу.

Рисунок 1.8 Архітектура Terraform

Terraform може керувати компонентами низького рівня, такими як обчислювальні ресурси, сховища та мережеві ресурси, а також компонентами високого рівня, такими як записи DNS та функції SaaS.

HashiCorp та спільнота Terraform вже написали тисячі провайдерів для керування безліччю різних типів ресурсів та послуг. Доступні всі

загальновідомі провайдери у реєстрі Terraform, включаючи Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog та багато інших.

Основний робочий процес Terraform складається з трьох етапів:

Write: на цьому етапі визначаються ресурси, які можуть мати декілька хмарних провайдерів і служб. Наприклад, можна створити конфігурацію для розгортання програми на віртуальних машинах у мережі віртуальної приватної хмари (VPC) з групами безпеки та балансувальником навантаження.

Plan: Terraform створює план виконання, який описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та конфігурації.

Apply: після затвердження Terraform виконує запропоновані операції у правильному порядку, дотримуючись будь-яких залежностей ресурсів. Наприклад, якщо оновити властивості VPC і змінити кількість віртуальних машин у цьому VPC, Terraform відтворить VPC перед масштабуванням віртуальних машин. [10]

До основних переваг Terraform відносять гарну підтримку зі сторони розробників та спільноти, широке використання в багатьох компаніях світу, можливість роботи як з високорівневими системами, так і з низькорівневими.

До основних недоліків можна віднести відносно низьку гнучкість інструменту, в базовому його використанні можна користуватися лише зазначеним у провайдера функціоналом, код інфраструктури необхідно писати на специфічній мові програмування HCL, яка дуже схожа на JSON формат.

1.9 AWS CloudFormation

AWS CloudFormation — це сервіс, який допомагає моделювати та налаштовувати ресурси AWS, за для того, щоб компанії могли витрачати менше часу на керування ресурсами та більше часу на програми, що працюють у AWS. [11] Необхідно створити шаблони, що описують всі необхідні ресурси AWS (наприклад, інстанси Amazon EC2 або інстанси Amazon RDS DB), а CloudFormation подбає про надання та налаштування цих ресурсів (рис 1.9).

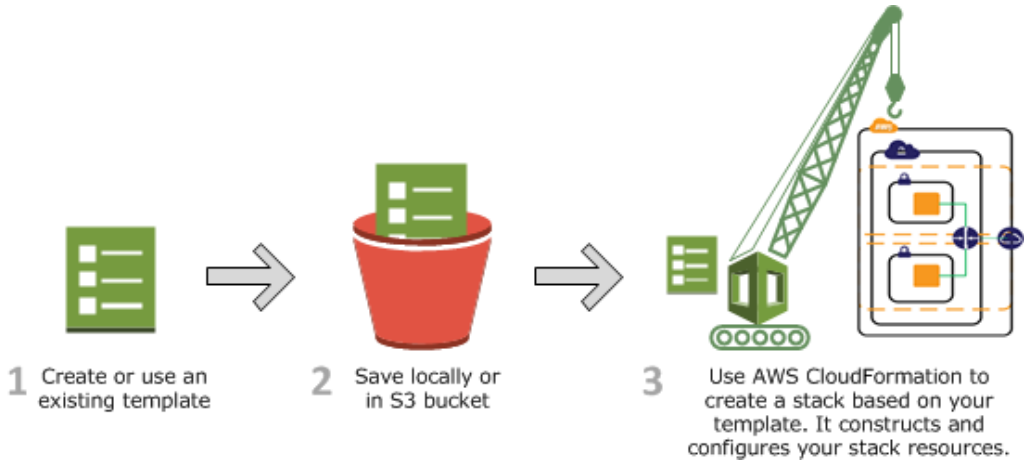


Рисунок 1.9 Архітектура CloudFormation

Використовуючи CloudFormation не потрібно індивідуально створювати та налаштовувати ресурси AWS та з'ясовувати, що від чого залежить; CloudFormation справляється із цим. Для масштабованого веб-додатку, який також включає серверну базу даних, можна використовувати групу Auto Scaling, балансувальник навантаження Elastic Load Balancing та екземпляр бази даних Amazon Relational Database Service. Один з варіантів використовувати кожну окрему службу для підготовки цих ресурсів, і після створення ресурсів вам доведеться

налаштувати їх для спільної роботи. Всі ці завдання можуть додати складності та часу ще до того, як ви запустите свою програму.

Натомість можна створити шаблон CloudFormation або змінити існуючий. Шаблон описує всі ресурси та їх властивості. Коли застосовується цей шаблон для створення стека CloudFormation, CloudFormation готує для групу Auto Scaling, балансувальник навантаження та базу даних. Після успішного створення стека ресурси AWS готові до роботи. Також можна легко видалити стек, що призведе до видалення всіх ресурсів у стеку.

До основних переваг CloudFormation можна віднести те, що він дуже добре пристосований для виконання коду та створення ресурсів AWS, його гнучкість та гарну підримку зі сторони розробників.

Головний недолік AWS CloudFormation це те, що цей інструмент працює лише з змарний провайдером Amazon Web Services (AWS). З ніяким іншим провайдером взаємодіяти не можна. Також, інша проблема в тому, що локально працювати з цим інструментом також не вийде, він розрахований лише на запуск в середовищі AWS.

Висновок до розділу 1

У першому розділі було проаналізовано інформацію про те, що таке IaC. IaC - це управління інфраструктурою та її надання за допомогою коду, а не ручних процесів. Також розгнуто підходи роботи інструментів Infrastructure as Code:

1. декларативний підхід
2. імперативний підхід

Окремим пунктом було розглянуто головні переваги IaC. До головних переваг можна віднести те, що в кінцевому випадку використання цих інструментів економить дуже багато часу за рахунок

спрощення розгортання та відновлення інфраструктури. А також зручність використання.

У розділі було наведено та проаналізовано найпопулярніші інструмени IaC. Аналізуючи дані у сухому залишку, можна зробити висновок, що найбільш оптимальним інструментом для виконання заданої дипломною роботи буде Terraform. Цей інструмент найбільш популярний, має найбільшу підтримку спільноти, а також, виходячи з теми дипломної роботи, це єдиний інструмент, який працює у зв’язці з Terragrunt.

Адже, Terragrunt є легкою оболонкою для виконання коду Terraform та необхідний для того, щоб зменшити кількість написаного коду Terraform та спростити його виконання. Під час розробки розділу сформовано вимоги до програмно-апаратного забезпечення.

РОЗДІЛ 2 РОЗРОБКА МОДУЛІВ IaC ТА СТРУКТУРИ КАТАЛОГІВ TERRAGRUNT

2.1. Початок роботи та встановлення необхідних інструментів

Інструменти IaC предбачають те, що інфраструктура буде розгортатися у змарному провайдері. Для виконання магістерської роботи був обраний провйдер Amazon Web Services (AWS), як один з найпопулярніших та найбільш поширених провайдерів.

Перед початком роботи над модулями Terraform необхідно підгнотовити робочу машину до написання коду та встановити все необхідне, а саме:

1. IDE для роботи
2. Розширення для IDE для зручності роботи
3. Terraform
4. Terragrunt
5. Створити аккаунт AWS
6. Згенерувати AWS API Keys

2.1.1. Встановлення IDE

Інтегроване середовище розробки (IDE) - це програма, яка допомагає розробникам ефективно розробляти програмний код. Це підвищує продуктивність розробників за рахунок об'єднання таких можливостей, як редагування, складання, тестування та збирання програмного забезпечення у простому середовищі. [12] Так само, як письменники використовують текстові редактори, а бухгалтери використовують електронні таблиці, розробники програмного забезпечення використовують IDE, щоб спростити свою роботу. Інтегровані середовища розробки (IDE) можна розділити на кілька різних категорій залежно від підтримуваної ними розробки застосунків та принципу їхньої роботи. Однак багато IDE можна віднести до кількох категорій. Нижче наведено деякі типи IDE:

Локальні IDE. Розробники встановлюють та запускають локальні IDE безпосередньо на своїх локальних комп'ютерах (рис 2.1). Їм також необхідно завантажити та встановити різні додаткові бібліотеки в залежності від їх потреб у розробці, вимог проекту та мови розробки.

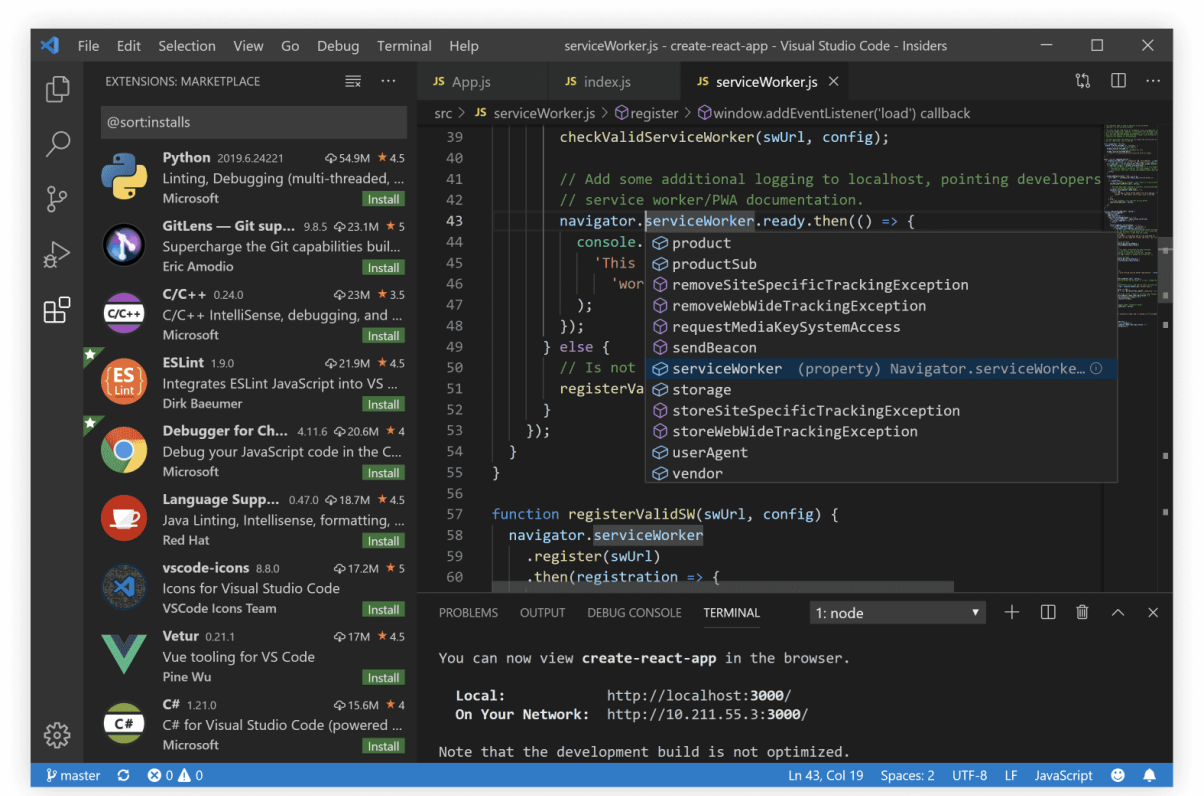


Рисунок 2.1 Локальна IDE

Хоча локальні IDE налаштовуються та не вимагають підключення до Інтернету після встановлення, вони створюють декілька проблем:

1. Їхнє налаштування може займати багато часу і бути складним.
2. Вони споживають ресурси локального комп'ютера і значно знизити продуктивність комп'ютера.

3. Відмінності у конфігурації між локальним комп'ютером та виробничим середовищем можуть призвести до помилок програмного забезпечення.

Хмарні IDE

Розробники використовують хмарні IDE для написання, редагування та компіляції коду безпосередньо у браузері, тому їм не потрібно завантажувати програмне забезпечення на свої локальні комп'ютери. Хмарні IDE мають низку переваг у порівнянні з традиційними IDE. [13] Ось деякі з цих переваг:

1. Стандартизоване середовище розробки
2. Команди розробників програмного забезпечення можуть централізовано налаштувати хмарну IDE для створення стандартного середовища розробки. Цей метод допомагає уникнути помилок, які можуть виникнути через відмінності в конфігурації локальних комп'ютерів.
3. Незалежність від платформи

Для виконання дипломної роботи було обрано одну з найпопулярніших IDE. Visual Studio Code. Вона є доступною у загальному доступі у мережі інтернет та є безкоштовною для всіх платформ. Visual Studio Code поєднує простоту редактора вихідного коду з потужними інструментами для розробників, такими як компіляція коду і налагодження IntelliSense. Простий цикл редагування-компіляції-налагодження - це менше часу на метушню з середовищем і більше часу на реалізацію ідей. Також VS Code дуже зручний для цілей, які планується досягнути під час виконання дипломоної роботи. [14]

Одна з головних переваг VS Code (рис. 2.2) - його невеликі системні вимоги до комп’ютера на якому встановлений та великий простір для налагодження та змінення.

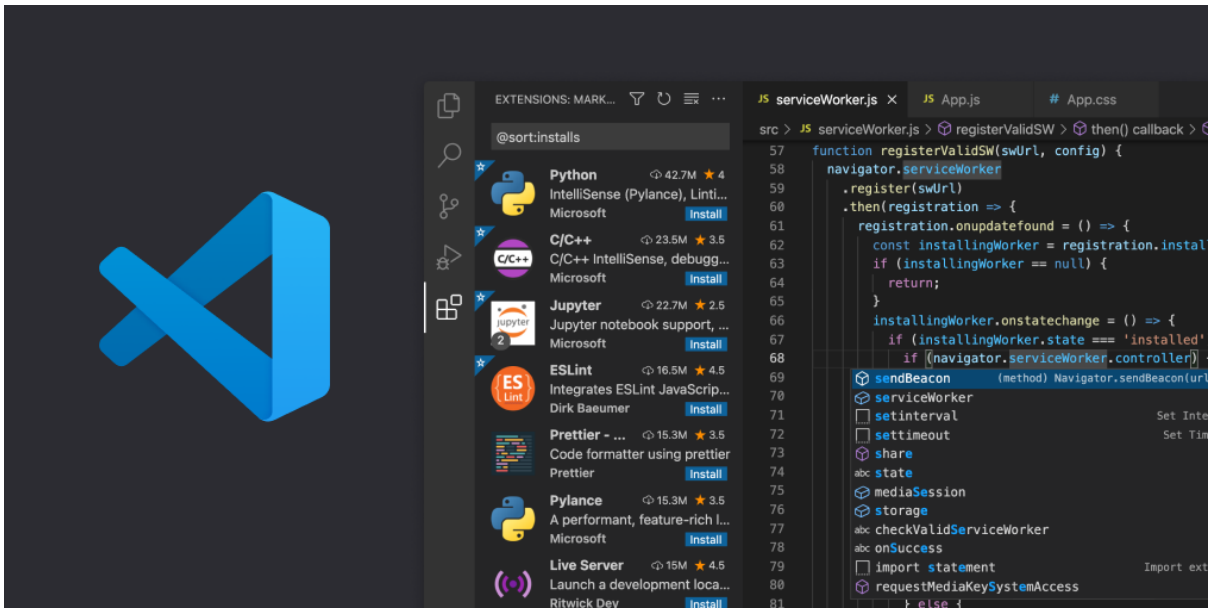


Рисунок 2.2 Visual Studio Code

2.1.2. Встановлення розширень для IDE

Функції, які Visual Studio Code включає за замовчуванням, - це тільки початок. Розширення VS Code дозволяють додавати в функціонал редактору мови, деьагери та інструменти для підтримки робочого процесу розробки.

Багата модель масштабування розширень для VS Code дозволяє авторам розширень підключатися безпосередньо до інтерфейсу користувача VS Code і вносити функціональні можливості через ті ж API, які використовуються в VS Code.

У VS Code можна переглядати та встановлювати розширення з VS Code. Для цього необхідно відкрити вкладку «Розширення», клацнувши значок «Розширення» (рис 2.3) на панелі дій збоку від VS Code. [15]

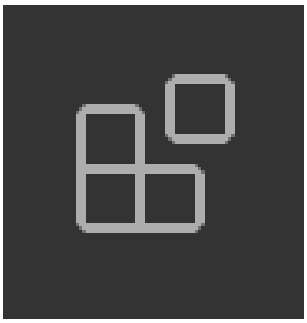


Рисунок 2.3 Іконка розширень у Visual Studio Code

Натиснувши вище зазначену іконку, можна отримати список найпопулярніших розширень VS Code на VS Code Marketplace. Для досягнення мети підрозділу необхідно у відкритому вікні натиснути «Пошук» та ввести «Terraform» (рис 2.4).

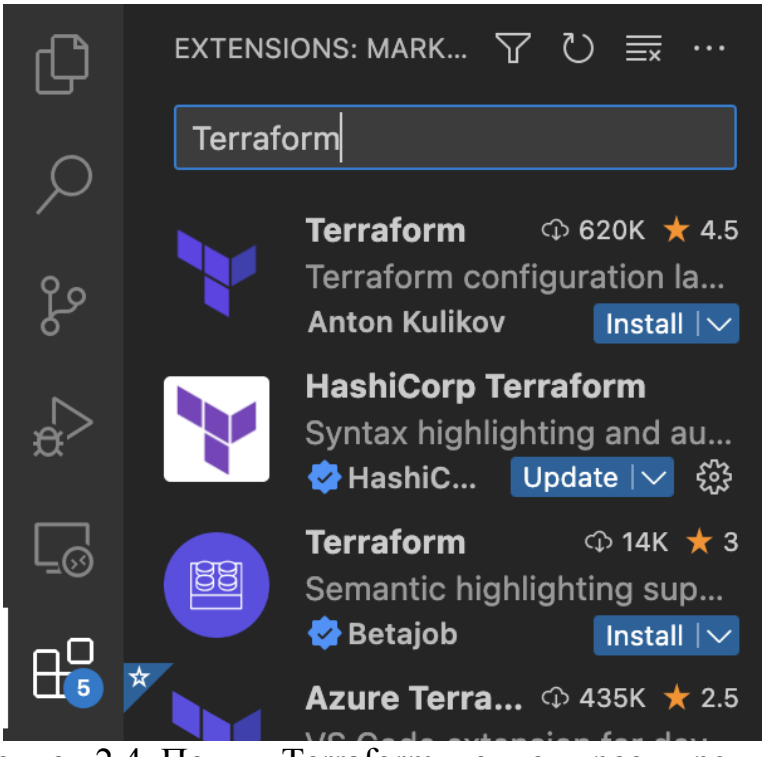


Рисунок 2.4 Пошук Terraform у списку розширень

Необхідно обрати розширення та натиснути «Встановити». Для виконання дипломної роботи було обрано розширення «HashiCorp Terraform». Розширення HashiCorp Terraform (рис. 2.5) для Visual Studio Code (VS Code) з мовним сервером Terraform додає функції редагування для файлів Terraform, такі як підсвічування синтаксису, IntelliSense, навігація за кодом, форматування коду, провідник модулів та багато іншого.

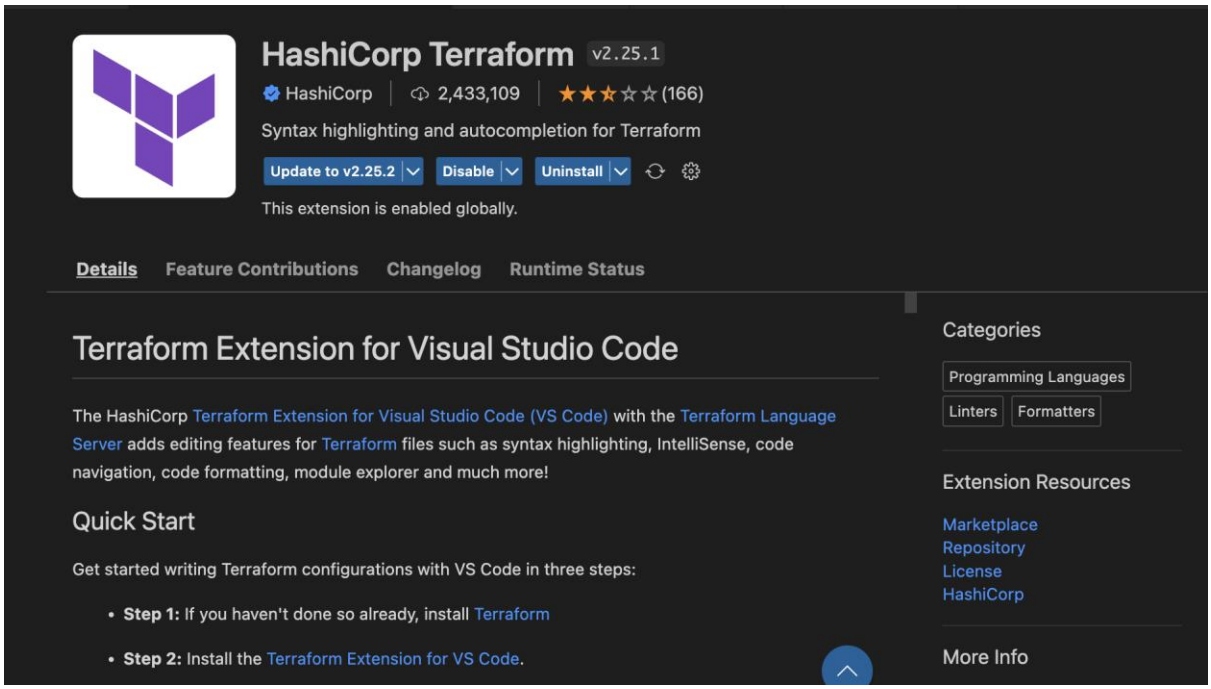


Рисунок 2.4 Розширення HashiCorp Terraform

2.1.3. Встановлення Terraform

Для виконання дипломної роботи буде використано Linux Based систему MacOS від Apple, з архітектурою процессору arm64. Тому, виходячи з наявного обладнання, нижче наведено інструкція для встановлення інструменту Terraform для зазначеного обладнання.

HashiCorp Terraform — це інструмент «інфраструктура як код», який дозволяє визначати як хмарні, так і локальні ресурси в зручних для читання файлах конфігурації, які можна версіонувати, повторно використовувати і ділитися ними. Можна використовувати узгоджений робочий процес для підготовки та управління всією інфраструктурою протягом усього її життєвого циклу. [15]

Для встановлення Terraform локально, необхідно виконати наступні кроки:

Встановити Homebrew, вписавши в пошукову строку Google запит «mac os install homebrew», вікрити термінал та виконати команду встановлення інструменту (рис 2.5).

Рисунок 2.5 Команда встановлення Homebrew

По-перше, необхідно встановити tap HashiCorp, сховище для всіх пакетів HashiCorp у Homebrew. Зробити це можна за допомогою команди «brew tap hashicorp/tap» (рис 2.6).

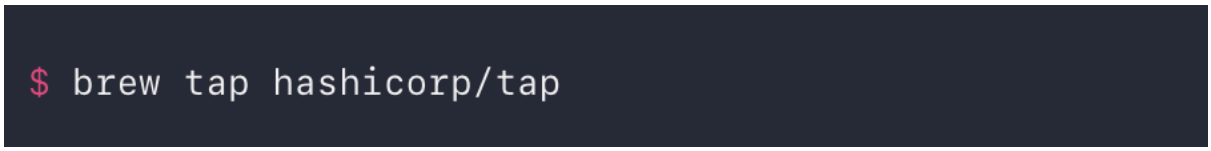


Рисунок 2.6 Команда встановлення сховища пакетів HashiCorp у Homebrew

Наступним кроком необхідно виконати саме встановлення самого Terraform. Для цього необхідно виконати команду «brew install hashicorp/tap/terraform» (рис. 2.7). За допомогою цієї команди інструмент Homebrew встановить все необхідне на комп’ютер, тобто залежності та встановить інструмент, який було необхідно встановити на початку.

```
$ brew install hashicorp/tap/terraform
```

Рисунок 2.7 Встановлення Terraform за допомогою brew

Після виконання всіх вище зазначених кроків необхідно виконати перевірку того, что все встановлено правильно. Це можна зробити, виконавши команду «terraform -help» (рис 2.8). Ця команда виведе на екран сторінку з інструкцією до використання Terraform, а також продемострує те, що сам інструмент знаходить у робочому стані. Це відбувається тому, що інструкція є в самому Terraform та ми, викликаємо вже встановлений інструмент та продивляємося інструкцію. Ти самим верифікуємо те, що інструмент готовий до роботи.

```
> terraform --help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure
```

Рисунок 2.8 Перевірка працездатності встановленого інструменту

У результаті виконання команди було виведено основні команди, з яким необхідно ознайомитись заздалегідь перед початком роботи. Це основні команди з якими буде проходити взаємодія.

Init: необхідно виконати перед першим запуском, виконує встановлення всіх залежностей, описаних у коді, встановлення провайдеру, тощо.

Plan: Terraform створює план виконання, який описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та конфігурації.

Apply: після затвердження Terraform виконує запропоновані операції у правильному порядку, дотримуючись будь-яких залежностей ресурсів. Наприклад, якщо оновити властивості VPC і змінити кількість віртуальних машин у цьому VPC, Terraform відтворить VPC перед масштабуванням віртуальних машин.

Destroy: Виконує зворотню процедуру від apply. Видаляє всі створені ресурси та очищує файл стану.

2.1.4. Встановлення Terragrunt

Terragrunt - це тонка обгортка, яка надає додаткові інструменти для збереження конфігурацій у без повторення коду, роботи з кількома модулями Terraform і керування віддаленим станом. Terragrunt використовується таким же чином, як і Terraform. [2] Основні команди залишаються незмінними, але основна відмінність у тому, що Terragrunt допомагає оформити код інфраструктіри таким чином, щоб не повторювати одні й ті самі модулі по декілька разів, а також допомагає спростити читання вже готового коду та структури інфраструктури, крім того, допомагає використовувати готовий модуль у багатьох різноманітних випадках, змінюючи модуль лише за допомогою змінних, описаних у Terragrunt. Дуже зручний інструмент, який допомагає

створювати декілька різноманітних оточень (рис. 2.9) за допомогою одного і того самого модулю Terraform.

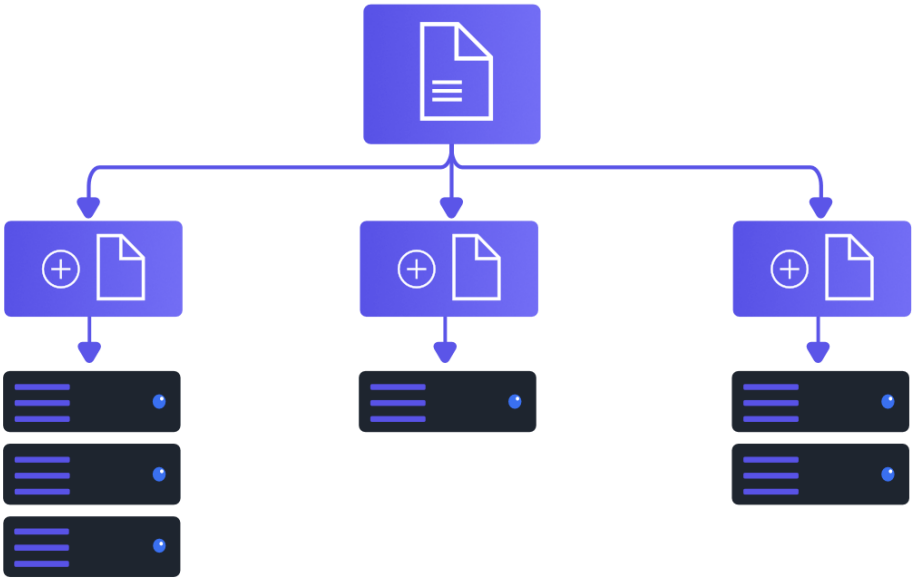


Рисунок 2.9 Схема структури Terragrunt

Встановлення Terragrunt відбувається за допомогою інструменту Homebrew. Для цього необхідно виконати просту команду «brew install terragrunt» (рис. 2.10).

Рисунок 2.10 Встановлення Terragrunt

Так само, як і у випадку з встановленням Terraform необхідно провести перевірку, чи правильно встановився інструмент та виконати команду «terragrunt -help» (рис. 2.11).

```
> terragrunt --help
DESCRIPTION:
   terragrunt - Terragrunt is a thin wrapper for Terraform that provides extra tools for working with multiple
   Terraform modules, remote state, and locking. For documentation, see https://github.com/gruntwork-io/terragrunt/.

USAGE:
   terragrunt <COMMAND> [GLOBAL OPTIONS]

COMMANDS:
   run-all                Run a terraform command against a 'stack' by running the specified command in each subfolder.
   terragrunt-info        Emits limited terragrunt state on stdout and exits
   validate-inputs        Checks if the terragrunt configured inputs align with the terraform defined variables.
   graph-dependencies     Prints the terragrunt dependency graph to stdout
   hclfmt                 Recursively find hcl files and rewrite them into a canonical format.
   aws-provider-patch     Overwrite settings on nested AWS providers to work around a Terraform bug (issue #13018)
   render-json            Render the final terragrunt config, with all variables, includes, and functions resolved, as j
cy Agent, or for debugging your terragrunt config.
   *                      Terragrunt forwards all other commands directly to Terraform
```

Рисунок 2.11 Перевірка працездатності встановленого Terragrunt

Terragrunt має той самий набір основних команд, що й Terraform:

Init: необхідно виконати перед першим запуском, виконує встановлення всіх залежностей, описаних у коді, встановлення провайдеру, тощо.

Plan: Terraform створює план виконання, який описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та конфігурації.

Apply: після затвердження Terraform виконує запропоновані операції у правильному порядку, дотримуючись будь-яких залежностей ресурсів. Наприклад, якщо оновити властивості VPC і змінити кількість віртуальних машин у цьому VPC, Terraform відтворить VPC перед масштабуванням віртуальних машин.

Destroy: Виконує зворотню процедуру від apply. Видаляє всі створені ресурси та очищує файл стану.

Велика відмінність полягає лише в тому, що Terragrunt має підкоманду «run-all». Сутність команди полягає у тому, що Terragrunt

проходить рекурсивно по всім папкам, розташованим по дереву каталогів вниз та виконує, команду яка розташована після підкоманди «run-all». Наприклад: «terragrunt run-all plan». У цьому випадку, інструмент виконає команду «plan» у всіх директоріях, які знаходяться нижче по дереву. Детальніше буде розглянуто у інших розділах.

2.1.5. Створення аккаунту AWS

Для того, щоб рухатися далі, згідно поставлених задач у даній дипломній роботі, необхідно створити AWS аккаунт, в якому будуть виконуватися всі дії написані у коді Terraform. Amazon Web Services (AWS) - це одна з найкомплексніших і найпоширеніших хмарних платформа у світі, що пропонує понад 200 повнофункціональних сервісів із центрів обробки даних по всьому світу. Мільйони клієнтів викростовують платформу для розгортаня своєї інфраструктури, у тому числі найстаріші стартапи, найбільші підприємства та провідні державні установи. [16]

У AWS значно більше сервісів і більше функцій у таких сервісах, ніж у будь-якого іншого постачальника хмарних послуг - від інфраструктурних технологій, таких як продуктивність, зберігання та дані, до нових технологій, таких як машинне навчання та штучний інтелект, data lakes та аналітики, а також Інтернет речей.

Це дозволяє швидше, простіше і скоротити витрати на виявлення у хмарі послуги і створити практично все, що тільки можна собі уявити в IT інфраструктурі.

Для того, щоб створити AWS аккаунт, необхідно виконати наступні кроки:

1. Ввідкрити домашню сторінку Amazon Web Services (рис 2.12).

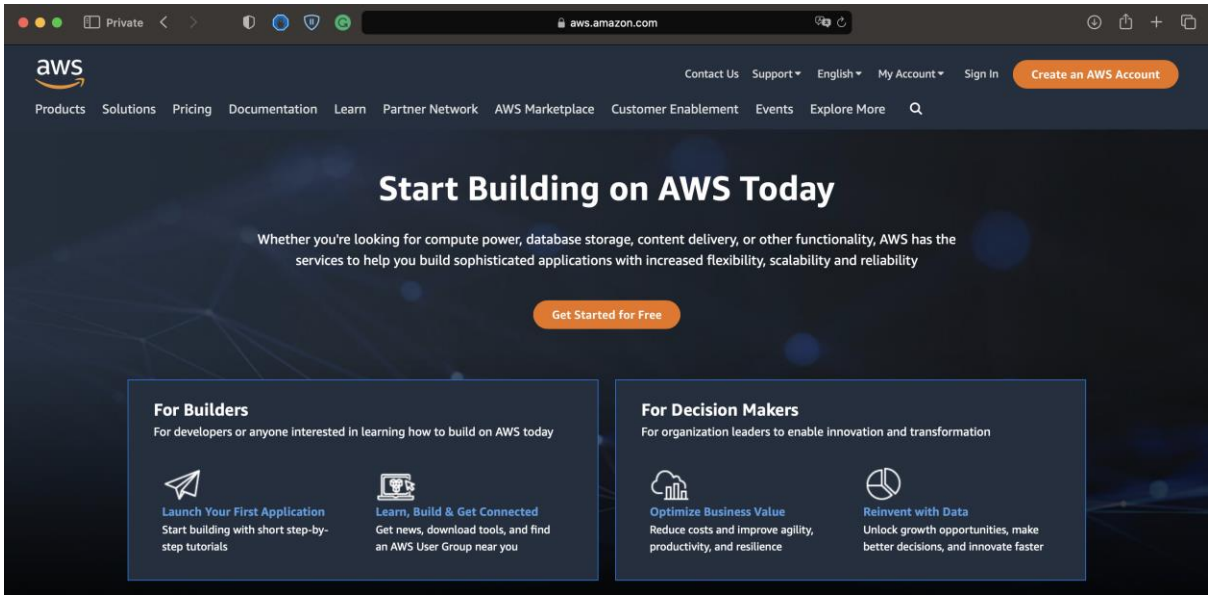


Рисунок 2.12 Домашня сторінка AWS консолі

2. Настиснути кнопку «Create an AWS Account» (рис 2.13).

Рисунок 2.13 Копка створення аккаунту AWS

3. Ввести дані облікового запису, а потім вибрати «Продовжити». Необхідно переконатися, що дані облікового запису правильно введені, особливо треба звернути увагу на адресу електронної пошти. Якщо ввести адресу електронної пошти неправильно, то неможливо буде доступ до нового облікогово запису AWS (рис 2.14). Так як головним ідентифікатором облікового запису є адреса електронної пошти.

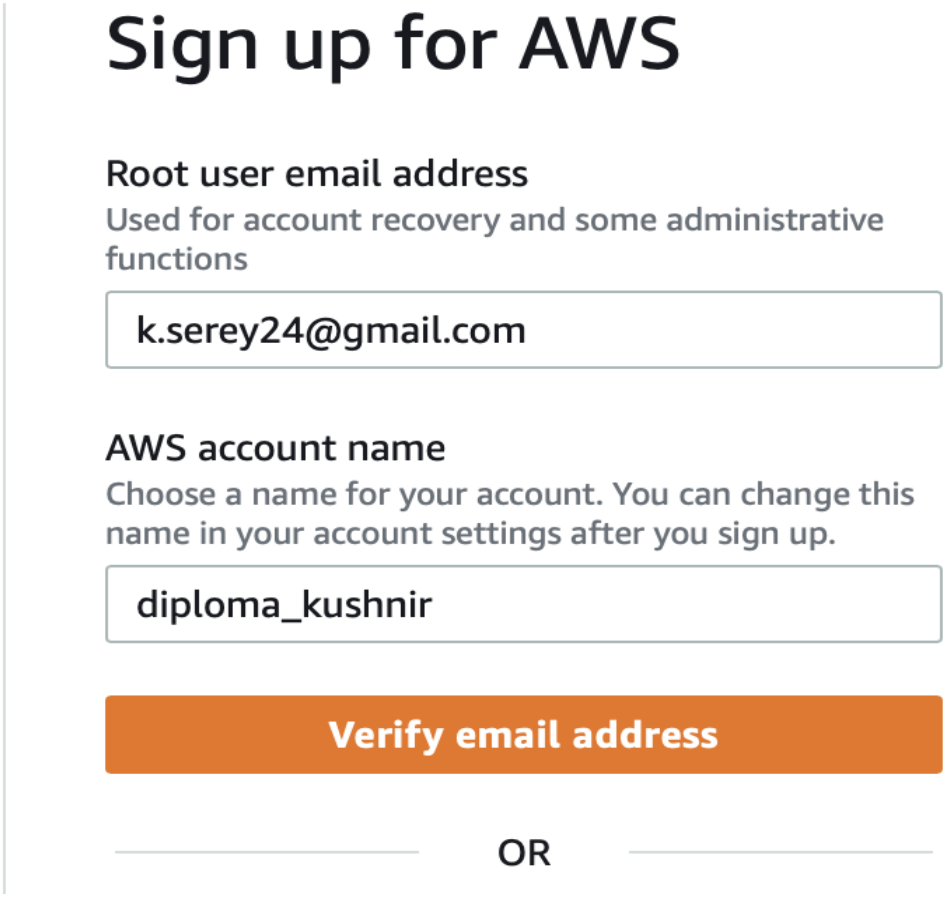


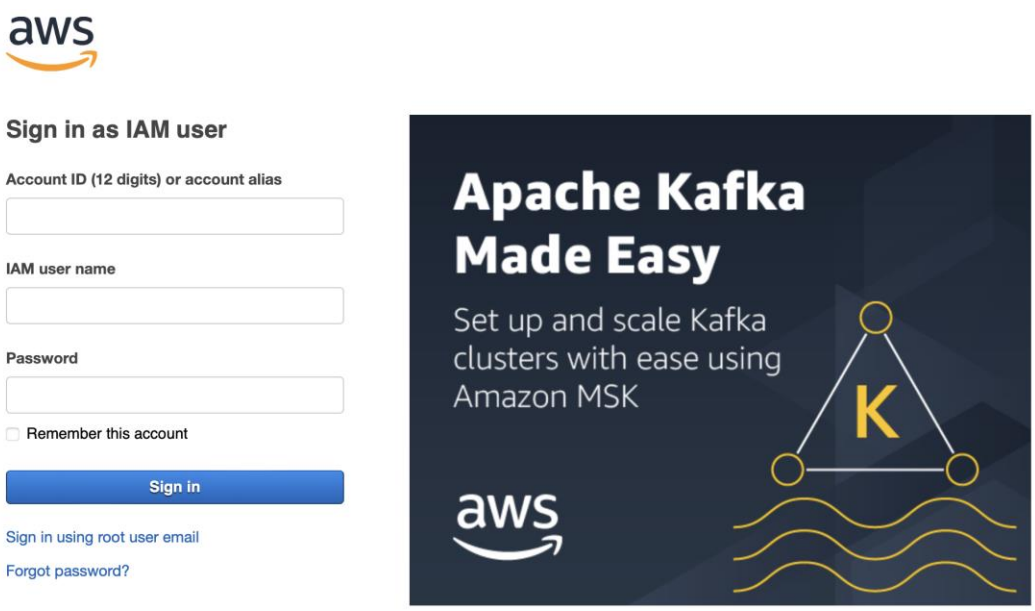
Рисунок 2.14 Введення данних облікового запису

4. Необхідно вибрати Особистий або Професійний аккаунт.
5. Необхідно ввести компанію чи особисту інформацію.
6. Важливий
7. Необхідно прочитати та прийняти Клієнтську угоду AWS.
8. Натиснути «Створити обліковий запис та продовжити».
9. На сторінці «Інформація про платеж» ввести інформацію про спосіб оплати, а потім вибрати «Підтвердити та додати».
10. Далі потрібно підтвердити номер телефону. Ввести код країни або регіону зі списку та ввести номер телефону, за яким можна буде зв'язатися в найближчі кілька хвилин.

11. Ввести код, який відображається в CAPTCHA, а потім надіслати повідомлення на доступний телефон.
12. Після отримання PIN-коду на телефон, необхідно ввести його і настичнути кнопку «Продовжити».
13. На сторінці «Вибір плану підтримки» необхідно обрати один із доступних планів підтримки AWS.

Після виконання всіх вище зазначених кроків необхідно зачекати, поки новий обліковий запис буде активовано. Зазвичай це займає кілька хвилин, але може тривати до 24 години.

Після активації аккаунту необхідно зайти на сторінку входу (рис



2.15) до AWS та виконати логін в головний аккаунт.

Рисунок 2.15 Сторінка входу до AWS аккаунту

Після виконання входу до аккаунту, повинно з'явитися головна консоль (рис 2.16). Якщо, консоль стала доступна, то реєстрацію можна вважати виконано успішно та правильно.

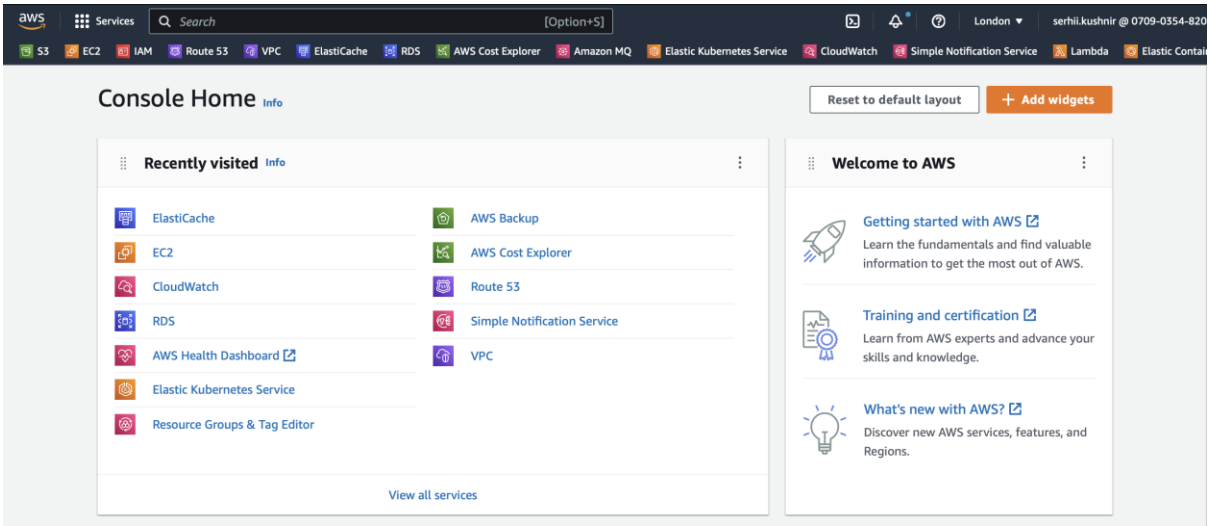


Рисунок 2.16 Головна консоль AWS аккаунту

2.1.6. Створення API ключів для AWS

У AWS дуже широкий функціонал, створений для великих компаній, тому необхідно лімітувати права для створення та управління ресурсами в організації. В AWS цю роль виконує функціонал IAM (Identity and Access Management).

AWS Identity and Access Management (IAM) – це веб-сервіс, який допомагає безпечно контролювати доступ до ресурсів AWS. За допомогою IAM можна централізовано керувати дозволами, які контролюють доступ користувачів до ресурсів AWS.

IAM використовується, щоб контролювати, хто пройшов автентифікацію (увійшов до системи) і авторизований (має дозволу) для використання ресурсів.

Під час створення облікового запису AWS створюється одне посвідчення для входу, яке має повний доступ до всіх сервісів та ресурсів AWS в обліковому записі.

Це посвідчення називається кореневим користувачем облікового запису AWS, яке було створенно протягом попереднього підрозділу, і доступ до нього здійснюється шляхом входу з адресою електронної пошти та паролем, які було використано для створення облікового запису.

Виходячи з вище зазначеної інформації інструмент IAM необхідно використати для створення API ключів за для доступу до аккаунту через Terraform.

Для створення ключів необхідно:

1. Вибрати пошукову строку верхній частині головної консолі управління AWS (рис 2.17) та ввести запит IAM, після чого натиснути на строку, що з’явилася.

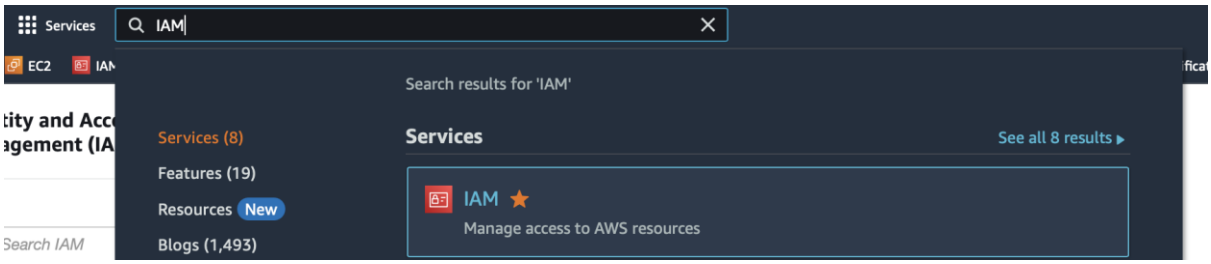


Рисунок 2.17 Пошук IAM

2. У лівій частині екрану необхідно знайти вкладку «Access management» та в цій вкладці обрати «Users» (рис 2.18).

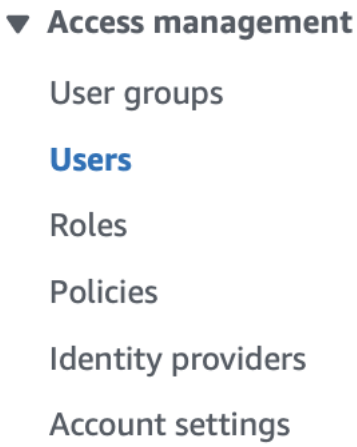


Рисунок 2.18 Вкладка «Users»

3. У відкритому вікні обрати свій обліковий запис та натиснути на нього.
4. Обрати вкладку «Security credentials»
5. Пролистати вниз до частини «Access keys» та настинути «Create access key» (рис. 2.19).

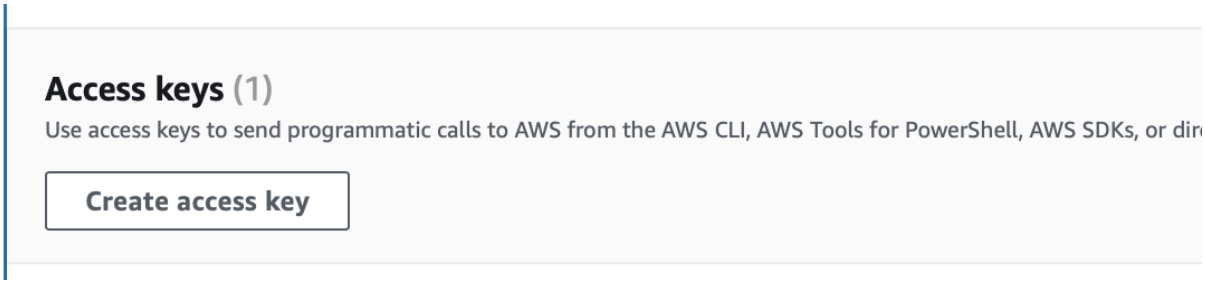


Рисунок 2.19 Створення «Access key»

Після створення ключа його ID та Secret необхідно зберегти у безпечному місці, для того, щоб використати у майбутньому.

2.2. Написання модулів Terraform

Для написання модулів, перш за все необхідно створити репозиторій на Gitlab (дипломна робота буде виконуватись на основні платформи системи контроля версій Gitlab).

GitLab – це веб-репозиторій Git, який надає безкоштовні відкриті та приватні репозиторії, можливості відстеження проблем. Це повна платформа DevOps, яка дозволяє професіоналам виконувати всі завдання у проекті - від планування проекту та управління вихідним кодом до моніторингу та забезпечення безпеки. Крім того, платформа дозволяє командам співпрацювати та створювати найкраще програмне забезпечення.

Після створення репозиторію його необхідно завантажити на комп’ютер та відкрити у VS Code (рис 2.20).

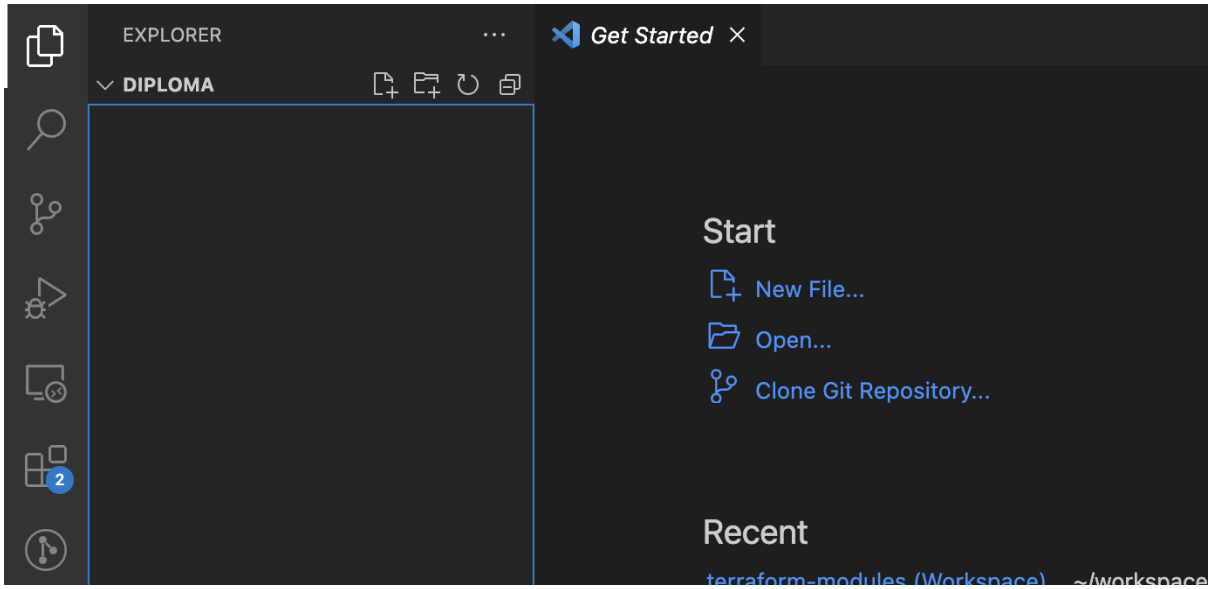


Рисунок 2.20 Відкритий репозиторій у VS code

На даному утапі репозиторій поки що пустий, для початку роботи необхідно створити дерево директорій (рис 2.21), яке буде

використовуватись для розгортання та перевикористання коду терраформ. Дерево директорій буде складатися на 1 рівні з 3 папок:

1. Diploma - папка в якій буде створений конфіг та структура оточення. Буде розглянута проятгом наступного розділу.
2. Environment - папка, яка буде служити для з’єднання папки з модулями та папки з налаштуваннями оточення. Так само буде розглянуто проятгом 3 розділу.
3. Modules - основна папка для роботи у цьомо розділі. Папка буде зберігати всі модулі, що будуть написані протягом цього розділу.

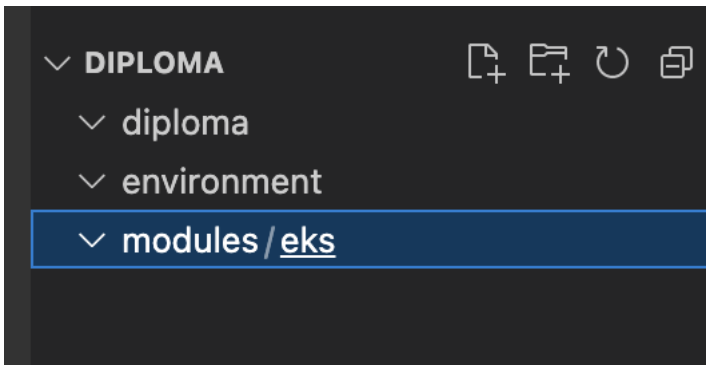


Рисунок 2.21 Базова структура папок для Terragrunt

2.2.1. Написання модулю EKS

Як видно на рисунку, папка «modules» вже має одну підпапку всередині. Ця папка буде вміщати у собі модуль для створення AWS EKS.

Amazon Elastic Kubernetes Service (Amazon EKS) – це керований сервіс Kubernetes, який спрощує запуск Kubernetes на AWS та локально. Kubernetes - це система з відкритим вихідним кодом для автоматизації розгортання, масштабування та керування контейнерними програмами.

Amazon EKS сертифікований як сумісний із Kubernetes, тому існуючі програми, що працюють у вихідному Kubernetes, сумісні з Amazon EKS.

Amazon EKS автоматично керує доступністю та масштабованістю вузлів площини управління Kubernetes, які відповідають за планування контейнерів, керування доступністю додатків, зберігання даних кластера та інші ключові завдання.

Amazon EKS дозволяє запускати програми Kubernetes як у Amazon Elastic Compute Cloud (Amazon EC2), так і в AWS Fargate. З Amazon EKS можна скористатися всіма перевагами продуктивності, масштабованості, надійності та доступності інфраструктури AWS, а також інтеграції з мережевими сервісами та сервісами безпеки AWS, такими як балансувальники навантаження додатків (ALB) для розподілу навантаження, AWS Identity and Access. Інтеграція управління (IAM) з управлінням доступом на основі ролей (RBAC) та підтримка віртуальної приватної хмари AWS (VPC) для мереж модулів.

Наступним кроком необхідно створити структури файлів для модуля Terraform (рис. 2.22).

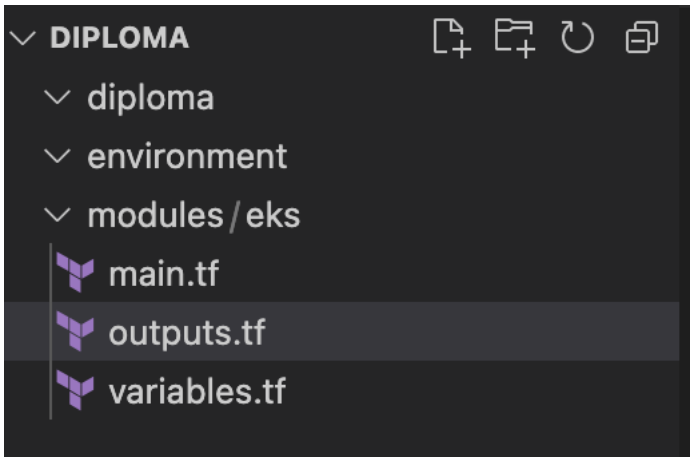


Рисунок 2.22 Структура файлів Terraform

Структура файлів для модуля Terraform було створено. Вона складається з 3 основних файлів. Авжеж, файлів може бути і більше, але на вище зазначеному рисунку показаний мінімально необхідні файли:

1. main.tf основний файл у якому буде написаний код Terraform.
2. outputs.tf файл у якому будуть зазначені змінні, які необхідно буде зберегти у файлі стану. Їх таком можна буде використовувати, як данні в залежних модулях.
3. variables.tf файл у якому додаються змінні для коду в файлі main.tf.

Перш за все необхідно зазначити версію Terraform для майбутнього модулю, а також провайдери, які будуть використовуватись протягом застосування модулю. У випадку з модулем EKS буде використано 4 провайдери та необхідна версія Terraform повинна бути не меншою за 1.0.0 (рис 2.23).

```
terraform {
  required_version = "> 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.72.0"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "2.10.0"
    }
    http = {
      source  = "terraform-aws-modules/http"
      version = "2.4.1"
    }

    tls = {
      source  = "hashicorp/tls"
      version = "3.1.0"
    }
  }
}
```

Рисунок 2.23 Провайдери модуля EKS

Провайдери у модулю EKS:

1. AWS провайдер Amazon Web Services. Основний провайдер для взаємодії з хмарним провайдером.
2. Kubernetes допоміжний провайдер для взаємодії з вже готовим кластером.
3. http допоміжний провайдер для генерації сертифікату доступу до кластеру.
4. tls допоміжний провайдер для генерації сертифікату доступу до кластеру.

Всі інструменти для роботи з усіма зазначеними провадерами завжди доступні у базі знань Terraform Terraform registry.

У модулі використовується декілька типів блоків з данними:

1. data отримує данні з провайдеру або з попередто виконаного модулю, або ресурсу.
2. Resource блок для описання ресурсу, який необхідно створити.
3. Locals блок, у якому встановлюються додатків змінні, яких не вистачає, або у якому проводяться додаткові маніпуляції зі змінними.
4. Module виконує ту саму функцію, що і ресурс, але може містити у собі декілька ресурсів об'єднаних у блок.

Найцікавіший блок у даному модулі це модуль «eks».

Кожна конфігурація Terraform має принаймні один модуль відомий як кореневий модуль, який складається з ресурсів, визначених у файлах .tf в основному робочому каталозі.

Модуль може викликати інші модулі, що дозволяє включати ресурси дочірнього модуля конфігурацію коротко. Модулі також можна викликати кілька разів, або в рамках однієї конфігурації або в окремих конфігураціях, що дозволяє упаковувати і повторно використовувати конфігурації ресурсів.

Це блок, у якому створюється саме кластер EKS з найменшими налаштування. У цьому блоці вказується:

1. source базовий ресурс провайдера, за допомогою якого будуть генеруватися виклики API. Всі модулі потребують вихідного аргументу, який є метааргументом, визначеним Terraform. Його значенням є шлях до локального каталогу, що містить файли конфігурації модуля, або шлях до віддаленого джерела модуля, який Terraform повинен завантажити і використовувати. Це значення має бути літеральним рядком без послідовностей шаблонів; довільні вирази не допускаються. Додаткові відомості про можливі значення цього аргументу див. у розділі Джерела модулів.
2. Version версія базового ресурсу провайдера.
3. cluster_name назва майбутнього кластеру.
4. cluster_version версія EKS Engine.
5. subnet_ids внутрішній ID підмережі VPC в мережі аккаунту AWS.
6. cluster_endpoint_private_access налаштування створення внутрішньої точки доступу до кластеру.
7. cluster_endpoint_public_access налаштування створення зовнішньої точки доступу до кластеру.
8. enable_irsa налаштування шифрування доступу до кластеру.
9. cluster_addons додаткові модулі для передвстановлення у кластер.
10. Tags мітки для ресурсів кластеру. Можуть бути корисними у випадку, коли в аккаунті велика кількість ресурсів та їх потрібно віддуляти один від одного по групам.
11. vpc_id ID внутрішньої інтернет мережі аккаунту AWS.
12. manage_aws_auth_configmap налаштування керування ролей доступу до кластеру EKS.
13. aws_auth_roles IAM ролі, які матимуть доступ до кластеру EKS.

Приклад заповнення модулю «EKS» (рис. 2.24).

```
module "eks" {
  source                          = "terraform-aws-modules/eks/aws"
  version                         = "18.20.0"
  cluster_name                    = var.eks_cluster_name
  cluster_version                 = var.cluster_version
  subnet_ids                      = var.private_subnets
  cluster_endpoint_private_access = true
  cluster_endpoint_public_access  = true
  enable_irsa                     = true

  cluster_addons = {
    vpc-cni = {
      resolve_conflicts = "OVERWRITE"
    }
  }

  tags = {
    "Name"                   = var.eks_cluster_name
    "Environment"            = var.environment
    "karpenter.sh/discovery" = var.eks_cluster_name
    provider_name            = "aws"
    department_name          = var.department_name
    project_env              = var.project_env
  }

  vpc_id = var.vpc_id

  manage_aws_auth_configmap = true

  aws_auth_roles = [
```

Рисунок 2.24 Модуль EKS

Повний код написаного модуля буде представлений у додатку.

2.2.2. Написання модулю группи серверів

На жаль, створення серверів EKS не є цілком автоматизованим хмарним провайдером AWS. Тому, за для нормальної роботи кластеру, необхідно створити групи серверів на яких будуть виконуватись певні сервісні поди, або певні користувацькі поди.

Виходячи з даної інформації було прийняте рішення написати додатковий допоміжний модуль для автоматизації створення группи серверів. Модуль буде названо «node_groups» та додано до структури каталогів (рис 2.25).

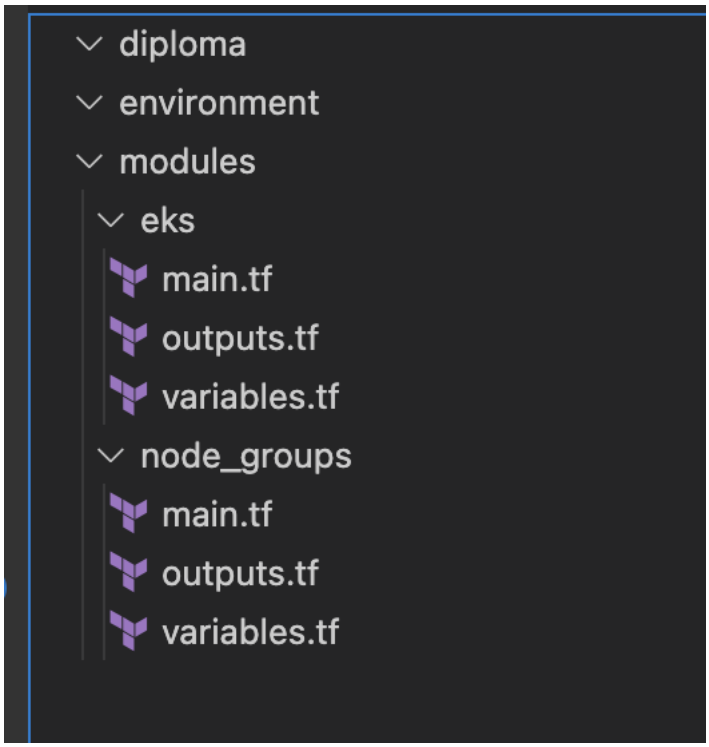


Рисунок 2.25 Модуль «node_groups»

Після додавання нового модулю до структури каталогів було створенно структуру файлів по замовчуванням, так само, як і в попередньому модулі, а саме:

1. main.tf основний файл у якому буде написаний код Terraform.
2. outputs.tf файл у якому будуть зазначені змінні, які необхідно буде зберегти у файлі стану. Їх таком можна буде використовувати, як данні в залежних модулях.
3. variables.tf файл у якому додаються змінні для коду в файлі main.tf.

Але, після того, як було почато написання основного файлу (рис. 2.26). Було прийнято рішення про те, що необхідно додадати допоміжні

файли у структуру. Допоміжні файли у структуру можна просто додати на тому ж рівні, що і головні файли та вони будуть виконані з тим самим пріоритетом, що і основні файли.

```
terraform {
  required_version = "> 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.72.0"
    }
  }
}

data "aws_iam_role" "nodes_iam_role" {
  name = var.nodes_iam_role_name
}
```

Рисунок 2.26 Написання модулю «node_groups»

У даному модулі буде використовуватись лише один провайдер AWS та версія Terraform повинна бути не меншою за 1.0.0. Також, використовуються данні про IAM роль, яка буде використана для новостворених інстансів.

Інстанс це персональна назва віртуальних серверів в Amazon Web Services. Деякі провайдери називають свої віртуальні сервери, які вони надають користувачам якимось персональними назвами, скоріше за все це зроблено за для меркетингових вигод.

Наступним кроком необхідно розробити список групп, які будуть використовуватись в рамках дипломної роботи. Для дипломної роботи

буде використано лише 2 групи серверів, одна обов’язкова система, а інша «робоча»:

1. System
2. Worker

У групі «System» будуть розташовані поди та сервіси, які необхідні для нормального функціонування EKS.

У групі «Worker» будуть розташовані всі інші сервіси, які будуть запущені користувачем.

Тому, треба додати наступні файли до структури файлів модулю (рис. 2.26):

1. system-node_group.tf
2. worker-node_group.tf

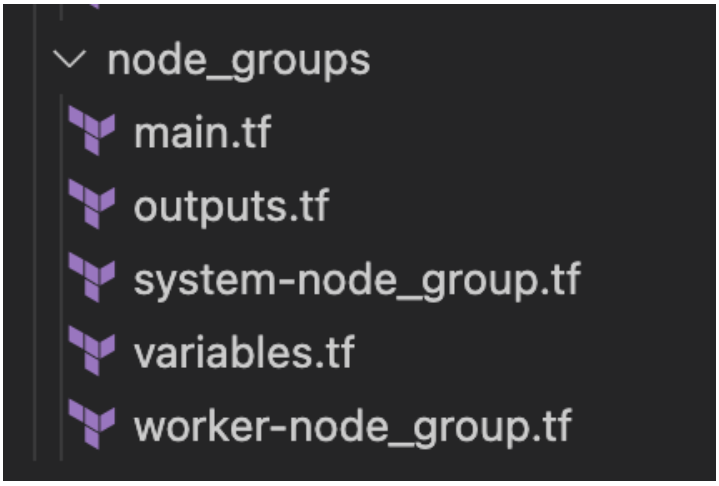


Рисунок 2.26 Додткові файли в стурктурі модулю

У кожному з додаткових файлів буде прописано єдиний ресурс, який будет визначати групу серверів та їх налаштування:

count початкова кількість інстансів у групі.

cluster_name назва кластеру до якого відносить группа, назва повинна бути однаковою з тією, що вказувалась при створенні кластеру.

node_group_name унікальна назва групи.

node_role_arn роль, яка буде прив’язана до кожного інстансу. Була вказана в основному файлі та регулює права доступу до ресурсів AWS.

subnet_ids ідентифікатор внутрішньої підмережі AWS,

ami_type тип ОС, яка буде використовуватись для новостворених інстансів.

capacity_type план покупки нових серверів. Може бути «ON_DEMAND» або «SPOT».

disk_size розмір корневого диску для системи.

instance_types визначає тип та розмірність інстансу або інстансів, якщо позначено використання декількох типів, який буде використано. Значення очікується у виді массиву.

Tags користувацький ідентифікатор ресурсу в аккаунті AWS. Значення очікується у вигляді масиву.

Labels користувацький ідентифікатор в середині EKS. EKS знаходить та взаємодіє з усіма ресурсами за допомогою labels. Значення очікується у вигляді масиву.

scaling_config параметри масштабування групи серверів. Однією з головних переваг EKS вважається масштабування інфраструктури, як в більшу, так і в меншу сторону, що дозволяє отримати гарну стабільність системи без великих фінансоіих витрати в порівнянні з статичною інфраструктурою.

Нижче наведено приклад створення групи серверів для групи під назвою «System» (рис. 2.27). Повний код створення всіх ресурсів буде приведений у додатку.

```
resource "aws_eks_node_group" "k8s_system" {
  count           = var.nodes.k8s_system.count
  cluster_name    = var.eks_cluster_name
  node_group_name = "${var.project_name}-k8s-system"
  node_role_arn   = data.aws_iam_role.nodes_iam_role.arn
  subnet_ids      = var.private_subnets

  ami_type      = "BOTTLEROCKET_ARM_64"
  capacity_type = "ON_DEMAND"
  disk_size     = "20"

  instance_types = [
    "m6g.medium"
  ]
```

Рисунок 2.27 Ресурс створення групи серверів «System»

2.3. Тестування написаного коду

Перед тестування написаного коду необхідно створити структуру файлів Terragrunt. Terragrunt є легкою оболонкою для виконання коду Terraform та необхідний для того, щоб зменшити кількість написаного коду Terraform та спростити його виконання. Тому, необхідно створити декілька підпапок та файлів, які будуть основними в репозиторії та обхідними для виконання коду.

Файл terragrunt.hcl розташований у корні репозиторія це базовий файл з налаштуваннями самого Terragrunt, з вказанням базового провайдеру, розташування файлу стану, тощо (рис. 2.28).

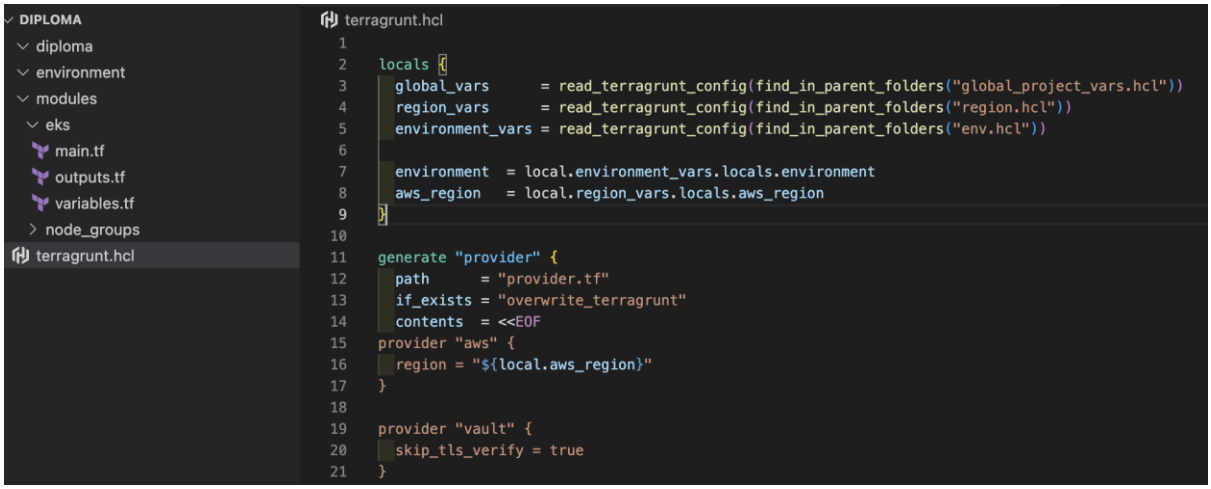


Рисунок 2.28 Створення кореневого файлу terragrunt.hcl

Також у зазначеному файлі вказані основні змінні, які будуть використані проятгом запуску модулю.

Після створення основго файлу, необхідно створити структуру та виконуваний файл для створення кластеру. У дипломній роботі буде створено «staging» кластер. Це означає, що буде створено тестовий кластер. Для цього необхідно в директорії «diploma» створити наступне дерево директорій. Директорії будуть вказані зверху вниз по структурі:

1. Директорія з вказанням регіону для виконання коду. Регіон можна вибрати зі списку доступних у провайдера. В дипломній роботі буде використано регіон «eu-west-1».
2. Директорія з вказанням оточення. Дипломна робота буде виконана в оточені staging.

Крім зазначених вище дирокторій, необхідно створити додаткові файли зі змінними для виконання коду. Саме ці маніпуляції і допомогають перевикористовувати код Terraform багато разів. Багато змінних великий простір для зміни виконуваного коду без його переписання.

Нижче будуть приведені файли, які необхідно створити та внести данні до цих файлів:

1. «diploma/global_project_vars.hcl» файл у розширені HCL. Зі змінними, які будуть відноситись до всього дипломного проєкту (рис 2.29). Оголошена лише одна змінна «project_name»

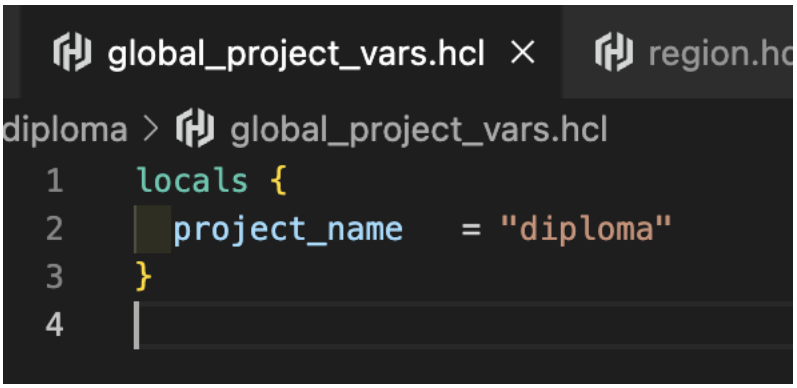


Рисунок 2.29 Заповнення файлу global_project_vars.hcl

2. «diploma/eu-west-1/region.hcl» файл у розширені HCL за змінними, які відносяться лише до конкретного регіону виконання коду. Файл заповнюється таким самим чином, як і попередній, за винятком назви змінної. Змінна повина мати назву «aws_region».
3. «diploma/eu-west-1/staging/env.hcl» файл у розширені HCL за змінними, які відносяться лише до конкретного регіону виконання коду. Файл заповнюється таким самим чином, як і попередній, за винятком назви змінної. Змінна повина мати назву «environment».

Вся структура повинанна мати вигляд дерево з директоріями та файлами (рис. 2.30).

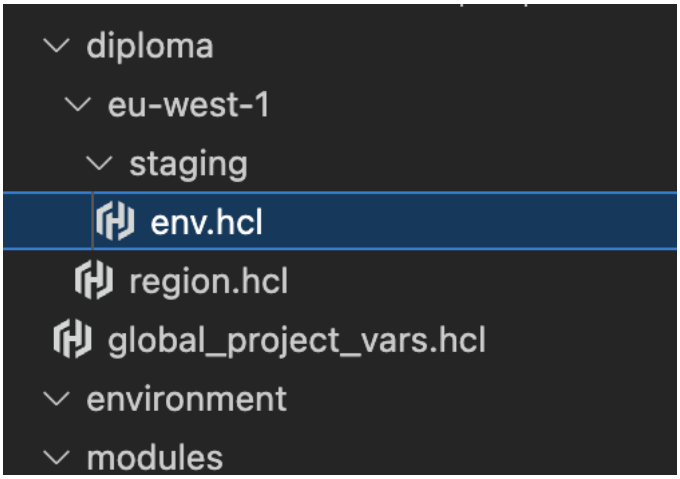


Рисунок 2.30 Дерево каталогів та файли конфігурації

Після чого у директорії «staging» необхідно створити піддиректорії, з назвами модулів, які будуть використані та створити у цих піддиректоріях виконувані файли для Terragrunt під назвою terragrunt.hcl.

Після чого, необхідно зазначити файли Terragrunt які будуть саме звантажувати модулі Terraform з директорії modules. Ці файли повинні лежати в дирекорії «environment» та мати у собі конфігурацію, передачу змінних та шлях до розташування модулів Terraform.

Після виконання всіх вище зазначених кроків структура катологів повианна мати дещо більший розмір, аніж це було на початку розділу. А саме, повинна бути директорія зі структурою для запуску модулів та змінними, які будуть використані, повинен бути каталог з модулями, який містить у собі піддиректорії з назвами модулів та структурою файлів Terraform.

Повинна бути директорія під назвою «environment», яка буде виконувати роль зв’язування двох інструментів: Terrafrom та Terragrunt.

Кінцевий вигляд структури каталогів повинен бути наступним (рис. 2.31).

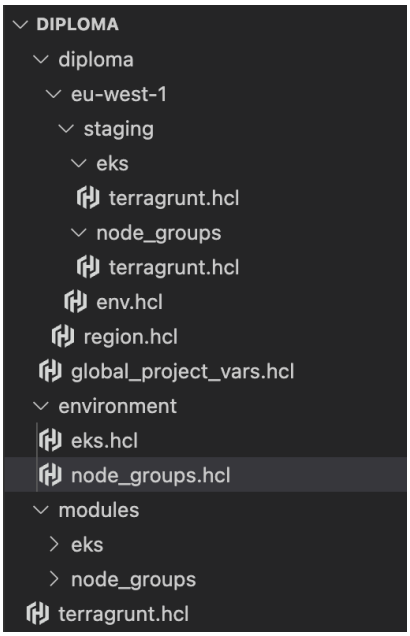


Рисунок 2.31 Кінцевий вигляд структури каталогів

Висновок до розділу 2

Під час розробки розділу було встановлено основні інструменти, які використовуються протягом дипломної роботи на локальний комп’ютер для тестування. А саме: Terraform та Terragrunt.

Основною частиною розділу став розділ написання коду Terraform. Під час написання розділу було написано код Terraform мовою HCL для двох модулів:

1. eks модуль створення EKS кластеру в хмарному провайдері AWS
2. node_groups модуль створення груп серверів для EKS кластеру.
3. Проаналізувавши вимоги до кластеру, було прийнято рішення створити 2 додаткових групи серверів:
4. system обов’язкова група серверів для додаткових системних служб кластеру.

5. Worker користувацька група в якій будуть запускатися всі користувацькі сервіси.

Кожна група була відділена одна від одної за допомогою labels.

РОЗДІЛ 3 НАЛАШТУВАННЯ АПАРАТНОЇ ЧАСТИНИ ТА РОЗРОБКА CI/CD ПАЙПЛАЙНУ

3.1. Вибір цільової платформа для виконання коду

Raspberry Pi (рис 3.1) це недорогий комп'ютер розміром із кредитну картку, який підключається до комп'ютерного монітора або телевізора та використовує стандартну клавіатуру та мишу.

Рисунок 3.1 Raspberry Pi 4 у корпусі

Це функціональний маленький пристрій, який дозволяє людям різного віку вивчати обчислювальну техніку і вчитися програмувати такими мовами, як Scratch і Python. [18] Він здатний робити все, що здатний робити звичайний настільний комп'ютер, від роботи в Інтернеті та відтворення відео високої чіткості до створення електронних таблиць, обробки текстів та ігор. А також виконання будь якого коду, сумісного з архітектурою процесору мікрокомп’ютера.

Raspberry Pi має можливість взаємодіяти із зовнішнім світом та використовувався у широкому спектрі проектів цифрових виробників, від музичних машин до метеостанцій. Принципова схема Raspberry PI (рис. 3.2)

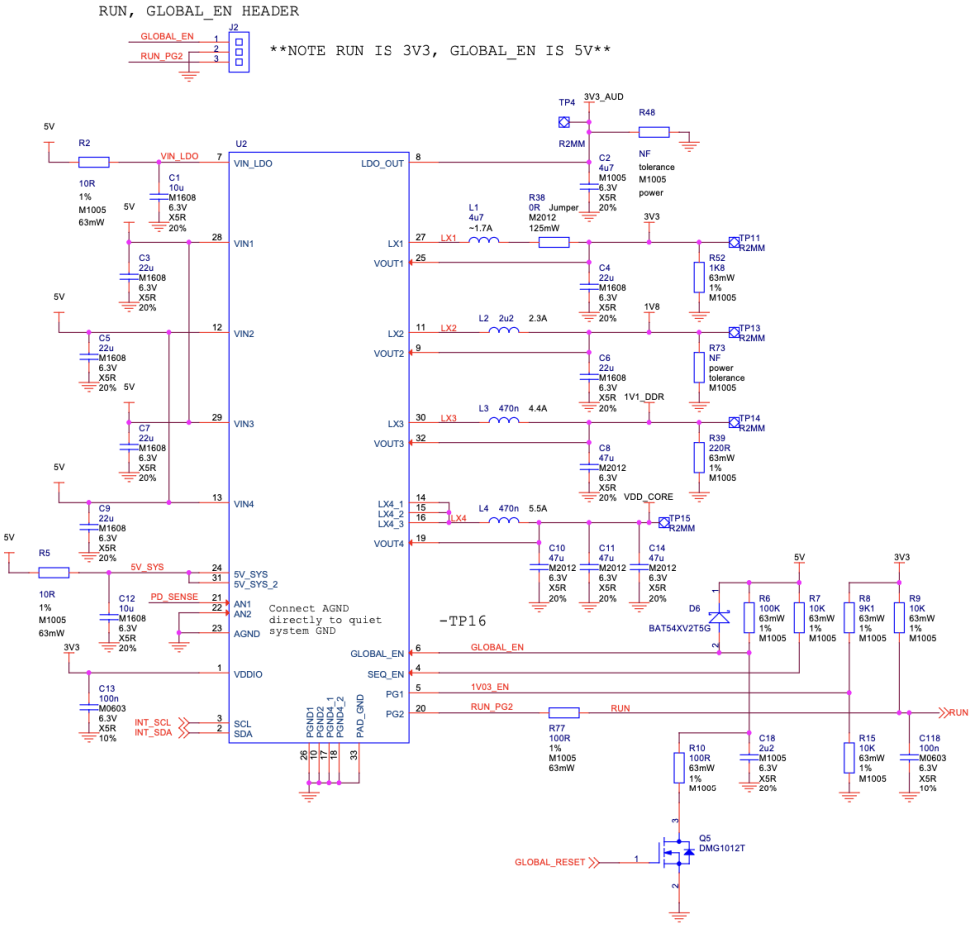


Рисунок 3.2 Принципова схема Raspberry PI

Raspberry Pi має архітектуру процесору ARM. Що накладає деякі обмеження у порівнянні з звичною архітектурою домішніх комп’ютерів, але для виконання коду Terraform у парі з Terragrunt це не має великого значення, адже зазначені інструменти є мільтиплатформеними. Крім

цього, локальний комп’ютер, який використовувався для тестів на базі MacOS також має архітектуру процесору arm64.

Кілька років тому підприємствам, яким була потрібна першокласна продуктивність CPU, доводилося боротися з високопродуктивними процесорами x86 або використовувати дорогі чіпи PowerPC. Тоді залежність галузі від чіпів x86, здавалося, зростала. І коли майже п'ятнадцять років тому Apple оголосила про намір використовувати процесори x86 для своїх комп'ютерів Mac, багато спостерігачів дійшли висновку, що ера процесорів, відмінних від x86, закінчилася.

CPU на базі ARM це сімейство процесорів, заснованих на архітектурі комп'ютера зі скороченим набором команд (RISC). Arm Holdings Ltd британська компанія розробляє архітектуру та передає її за ліцензією іншим постачальникам, які, у свою чергу, розробляють свої власні процесори на основі цих архітектур.

Дизайн ARM зазнав кількох ітерацій. Перша архітектура, широко відома як ARM1, використовувала 32-бітові регістри з 26-бітним адресним простором. Це обмежило основну пам'ять архітектури до 64 МБ. Пізніше Arm Holdings випустила серію ARMv3, яка збільшила продуктивність процесора. Додаткові ітерації до серії ARMv7 залишалися 32-бітовими, кожен процесор мав 15 регістрів загального призначення. [19]

Arm Holdings представила ARM64, також звану ARMv8-A, у 2011 році для розширення підтримки 64-бітових обчислень. На відміну від ARM32, який має 15 регістрів загального призначення, архітектура ARM64 використовує 31 регістр, кожен з яких має ширину 64 біти. Таким чином, його регістри можуть обробляти більші числа та містити більше адрес пам'яті.

Hackboard 2 (рис 3.3) - це новий одноплатний комп'ютер (SBC), розроблений з нуля, щоб стати одним із найпотужніших і найдоступніших

комп'ютерів на базі процесорів Intel з підтримкою Windows. Завдяки додатковому підключенню 4G або 5G він ідеально підходить для студентів, вчителів, сімей, виробників та аматорів. На відміну від Raspberry Pi, Hackboard 2 заснований на потужному процесорі Intel, здатному працювати під керуванням Windows. [24]

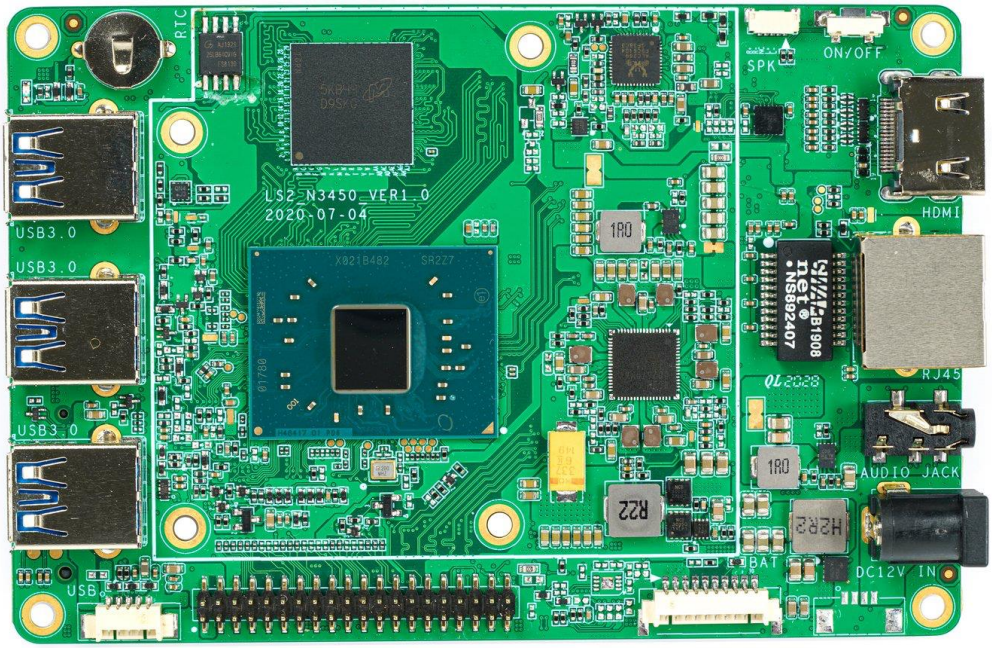


Рисунок 3.3 Hackboard 2

Порівнявши характеристики, та проаналізувавши майбутні задачі, було обрано платформу Raspberry PI з наступними характеристиками:

1. RAM: 2GB
2. CPU: 4 ARM cores
3. Disk: 32GB
4. OS: Linux based OS

3.2. Вибір базової операційної системи

Операційна система (ОС) - це програма, яка після початкового завантаження в комп'ютер завантажувальною програмою керує іншими прикладними програмами на комп'ютері. Прикладні програми використовують операційну систему, надсилаючи запити послуги через певний інтерфейс прикладних програм (API). Крім того, користувачі можуть безпосередньо взаємодіяти з операційною системою через інтерфейс користувача, такий як інтерфейс командного рядка (CLI) або графічний інтерфейс користувача (GUI). [22]

Для виконання дипломної роботи необхідна операційна система на базі Linux (рис. 3.4).

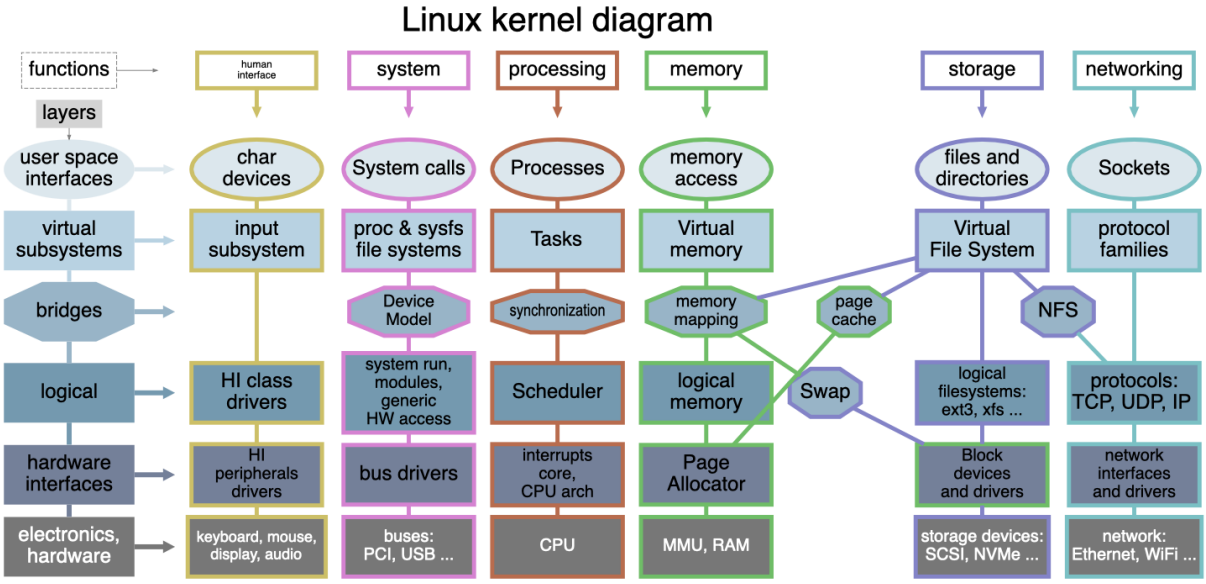


Рисунок 3.4 Linux kernel

Для виконання дипломної роботи необхідно обрати будь який дистрибутив операційної системи на базі Linux, який підтримує архітектуру проуцесору ARM. Наприклад:

1. Ubuntu
2. CentOS

3. Debian
4. Arch Linux

Ubuntu це дистрибутив Linux (рис 3.5), що базується на Debian і складається в основному з безкоштовного програмного забезпечення з відкритим вихідним кодом. [22]

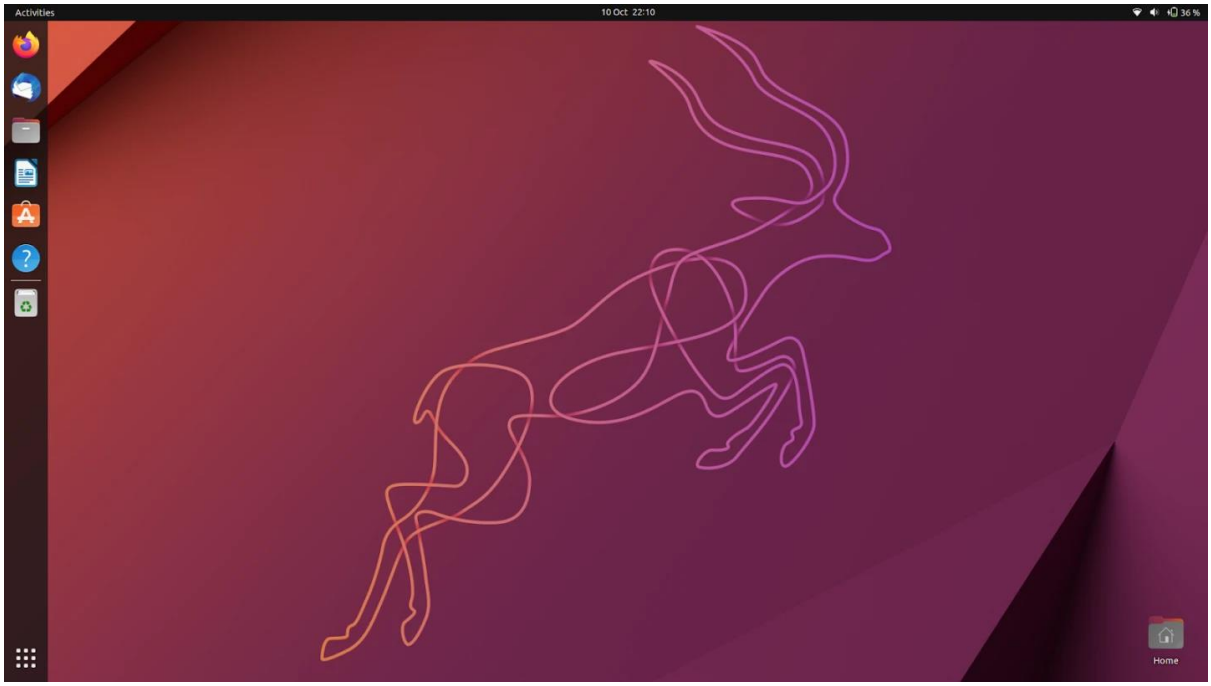


Рисунок 3.5 Робочій стіл Ubuntu

Ubuntu офіційно випущений у трьох версіях: Desktop, Server, та Core для пристроїв Інтернету речей та роботів. Всі випуски можуть працювати на комп'ютері окремо або на віртуальній машині. Ubuntu популярна операційна система для хмарних обчислень за допомогою OpenStack. Робочий стіл Ubuntu за замовчуванням змінився з внутрішньої Unity на GNOME майже через 6,5 років у 2017 році після випуску версії 17.10.

Ubuntu випускається кожні шість місяців, а випуски з довгостроковою підтримкою (LTS) – кожні два роки. Станом на жовтень

2022 року останнім випуском є 22.10 (Kinetic Kudu), а поточним випуском з довгостроковою підтримкою - 22.04 (Jammy Jellyfish).

CentOS (рис 3.6) - це операційна система з відкритим вихідним кодом, заснована на вихідному коді Red Hat Enterprise Linux і побудована на ядрі Linux, вперше представленому в 2004 році. [25]

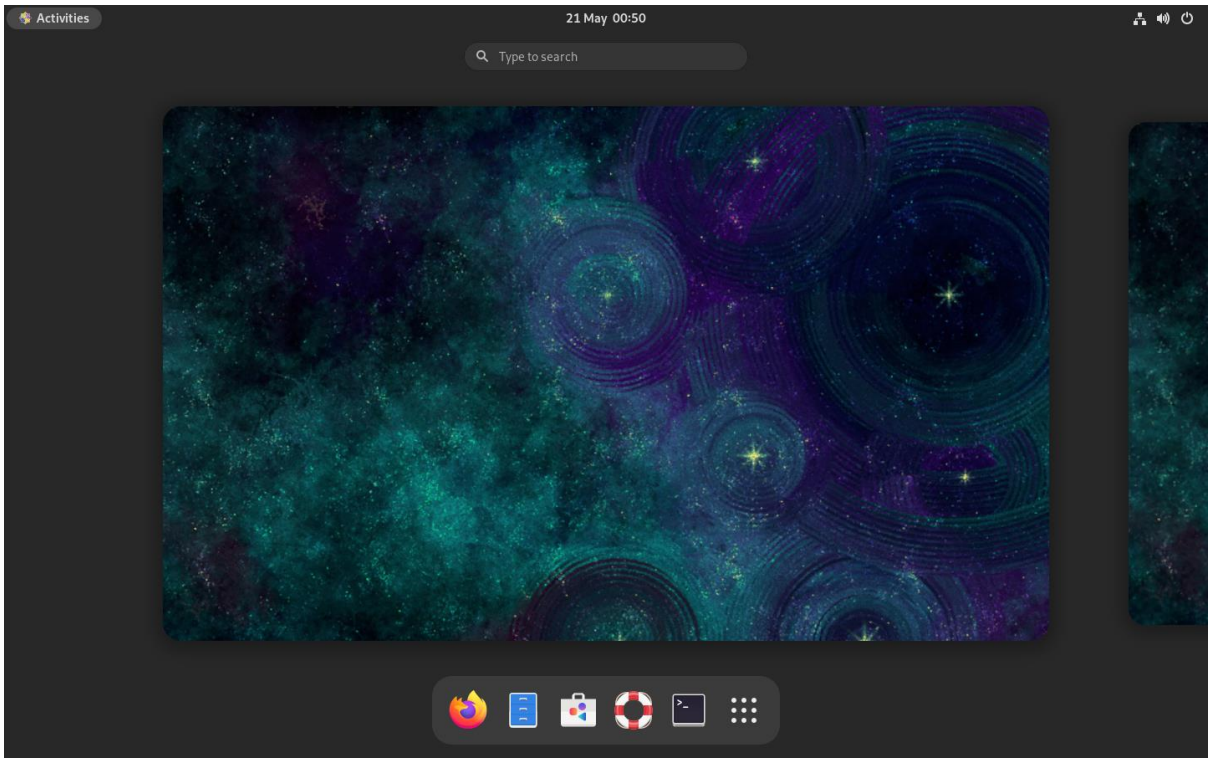


Рисунок 3.7 Робочій стіл CentOS

Це стабільне програмне забезпечення з високим рівнем безпеки та різними варіантами панелей.

CentOS використовується в багатьох проектах, від малого бізнесу до високопродуктивних корпоративних IT-додатків, через його стабільність, високу безпеку та численні опції панелі управління. Крім цього, CentOS має зростаючу спільноту, яка сприяє розвитку дистрибутива Linux,

створюючи вікі-контент, надаючи технічну підтримку та знаходячи виправлення помилок.

Для виконання дипломної роботи було обрано систему CentOS. Як одну з найпопулярніших систем на базі Linux. Для встановлення CentOS на Raspberry Pi 4 необхідно мати карту пам’яті мінімум на 16 GB. Але для дипломної роботи було обрана карта пам’яті на 32 GB. А також комп’ютер або ноутбук з адаптером для читання та записування данних на карту пам’яті.

Спочатку необхідно вставити картку microSD у комп'ютер.

Щоб завантажити CentOS для Raspberry Pi необхідно перейти на офіційний сайт CentoOS [21] (рис 3.8).

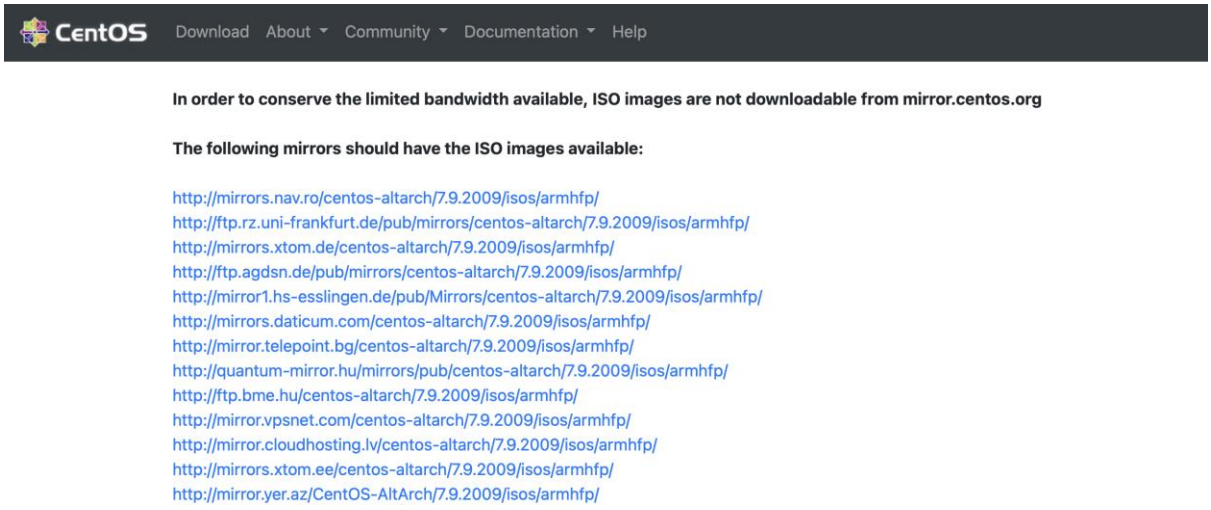


Рисунок 3.8 Офіційний сайт CentOS

Можна обрати будь яке посилання, але найкраще за все обирати найперше посилання. Це оптимальне посилання для регіону.

Після переходу по посиланню та відкриття дзеркального веб-сайту CentOS (рис. 3.9), необхідно обрати версію для Raspberry PI та

завантажити її. У дипломній роботі використовується версія CentOS-Userland-7-armv7hl-RaspberryPI-Minimal-4-2009-sda.raw.xz.

Index of /centos-altarch/7.9.2009/isos/armhfp

Name	Last modified	Size	Description
Parent Directory		-	
CentOS-Userland-7-armv7hl-generic-GNOME-2009-sda.raw.xz	2020-10-31 01:13	1.0G	
CentOS-Userland-7-armv7hl-generic-KDE-2009-sda.raw.xz	2020-10-31 01:25	1.0G	
CentOS-Userland-7-armv7hl-generic-Minimal-2009-sda.raw.xz	2020-10-31 01:33	559M	
CentOS-Userland-7-armv7hl-generic-Minimal-lpae-2009-sda.raw.xz	2020-10-31 01:38	550M	
CentOS-Userland-7-armv7hl-RaspberryPI-GNOME-2009-sda.raw.xz	2020-10-30 03:47	751M	
CentOS-Userland-7-armv7hl-RaspberryPI-KDE-2009-sda.raw.xz	2020-10-30 03:57	782M	
CentOS-Userland-7-armv7hl-RaspberryPI-Minimal-2009-sda.raw.xz	2020-10-30 04:15	305M	
CentOS-Userland-7-armv7hl-RaspberryPI-Minimal-4-2009-sda.raw.xz	2020-10-30 04:07	307M	
CentOS-Userland-7-armv7hl-RootFS-Minimal-2009-sda.raw.xz	2020-10-31 01:44	503M	
sha256sum.txt	2020-10-31 01:47	1.1K	
sha256sum.txt.asc	2020-11-06 16:53	1.9K	

Рисунок 3.9 Дзеркальний веб-сайт CentOS

Щоб записати образ на карту MicroSD, необхідно використати програмне забезпечення BalenaEtcher.

Відкривши BalenaEtcher на пристрої, необхідно настинути кнопку «Select Image». (рис. 3.10)

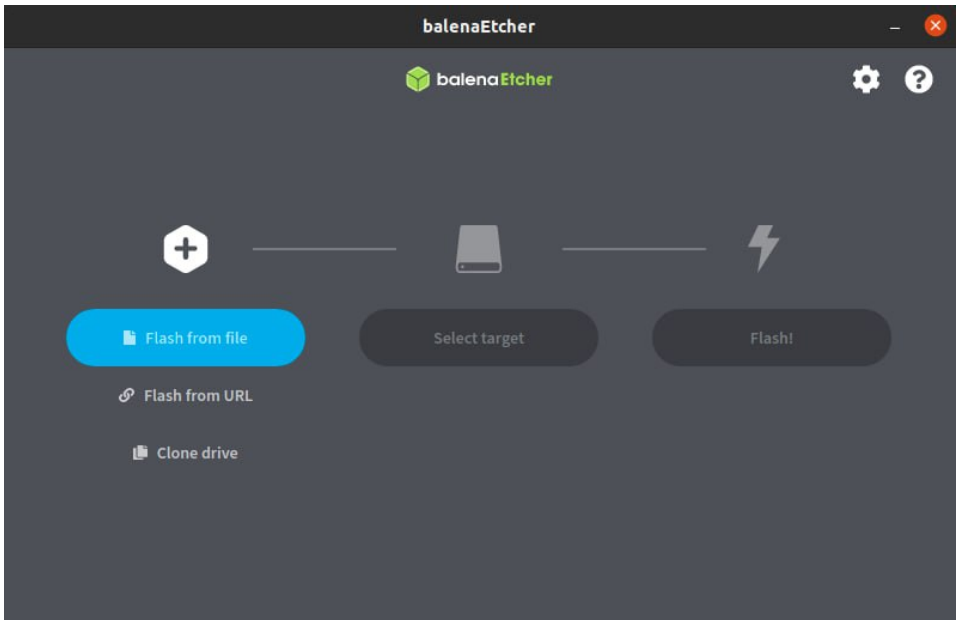


Рисунок 3.10 BalenaEtcher головний екран

Після натискання на кнопку з’явиться діалогове вікно, в якому необхідно вибрати завантажений файл. Наступним кроком необзхідно обрати диск, на який буде записаний образ та настиснути кнопку «Flash» (рис.3.11).

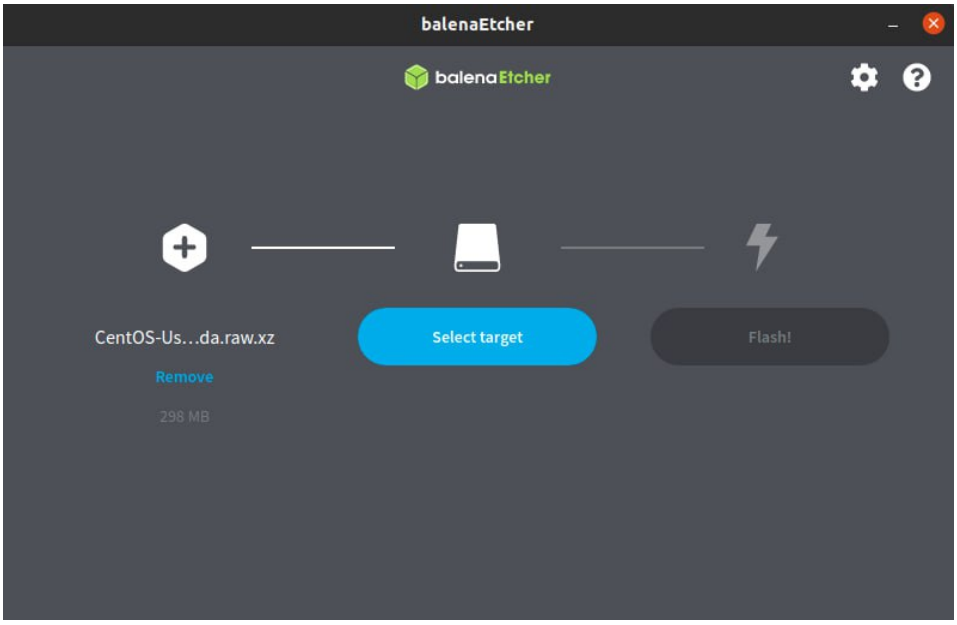


Рисунок 3.11 Запис на карту пам’яті

Після виконання вище зазначених кроків необхідно дістатати флеш карту та встановити її у Raspberry PI. Після виконання всіх кроків, які з’являться після встановлення флеш-карти у Raspberry PI система буде встановлена. Кроки не відрізняються від звичайного встановленн системи.

3.3. Gitlab Runner та його підготовка

GitLab Runner це програма, яка працює з GitLab CI/CD для запуску завдань у конвеєрі.

GitLab Runner можна встановити в інфраструктурі, якою керується. Якщо це зробити, слід встановити GitLab Runner на машині, відмінної від тієї, на якій розміщений екземпляр GitLab, з міркувань безпеки та

продуктивності. Коли використовуються окремі машини, можуть бути різні операційні системи та інструменти, такі як Kubernetes або Docker на кожній з них. [20]

GitLab Runner має відкритий вихідний код та написаний на Go. Його можна запустити як один двійковий файл, ніяких вимог до мови не потрібно.

Для встановлення GitLab Runner необхідно підключити Raspberry PI, та виконати кроки для встановлення GitLab Runner.

Необхідно завантажити виконуваний файл виконавши команду (1):

$$curl - LJ0\text{https://gitlab-runner-downloads.s3.amazonaws.com/latest/rpm/gitlab-runner\_arn} \quad (1)$$

Після успішного ванатаження необхдно виконати команду (2):

$$rpm - igitlab - runner_{arm}.rpm \ (2)$$

Після закінчення процесу встановлення необхідно перевірити, що екзекютор був успішно встановлений (рис 3.12).

```
root@8612a6416992:/# gitlab-runner --help
NAME:
   gitlab-runner - a GitLab Runner

USAGE:
   gitlab-runner [global options] command [command options] [arguments...]

VERSION:
   13.11.0 (7f7a4bb0)

AUTHOR:
   GitLab Inc. <support@gitlab.com>

COMMANDS:
     exec                execute a build locally
     list                List all configured runners
     run                 run multi runner service
     register            register a new runner
     install             install service
     uninstall           uninstall service
     start               start service
     stop                stop service
```

Рисунок 3.12 Перевікра встановленого Gitlab Runner

Крім того, необідно встановити Terraform, Terragrunt, а також інструмент для перевірки та тестування коду tflint. А також перевірити, що вони встановлені та працюють правильно.

Наступним кроком необзідно зареєструвати Gitlab Runner до готового серверу Gitlab. Для цього необхідно отримати авторизаційний токен Gitlab. Це можна зробити за допомогою документації гітлаб, необхідно скопіювати токен з Gitlab. Зайти на у термінал Raspberry PI та виконати команду (3):

$$sudogitlab - runnerregister \quad (3)$$

Після виконання команди та введення всіх данних, екзекютор буде зареєстрований у Gitlab та його можна буде використовувати за призначення дипломної роботи (рис 3.13).

```
root@8612a6416992:/# gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=313358 revision=7f7a4bb0 version=13.11.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
xxxxxx
Enter a description for the runner:
[gitlab-runner]: Raspberry Pi Runner
Enter tags for the runner (comma-separated):
Raspberry
```

Рисунок 3.13 Реєстрація Gitlab Runner

3.4. Написання CI/CD пайплайну

Для написання коду CI/CD пайплайну необхідно для початку створити файл .gitlab-ci.yml у вже інуючому репозиторію (рис 3.14).

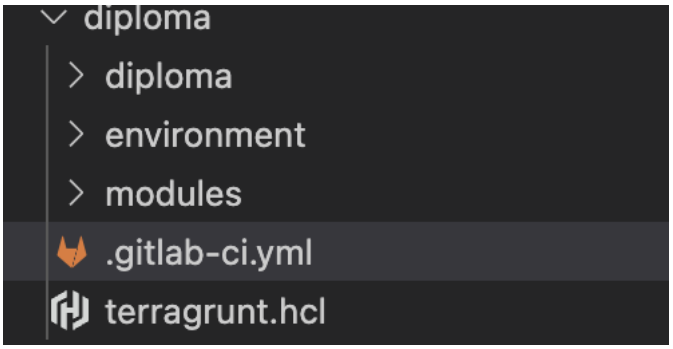


Рисунок 3.14 Файл gitlab-ci.yml

В цьому файлі буде описано увесь код пайплайну. Це основний файл, який запускаеться самим Gitlab та необхідний для виконання пайплайну.

Для правильного та безпечного виконання необхідно розділити задачі на декілька стадій (рис 3.15), а саме:

1. lint
2. validate
3. plan
4. apply

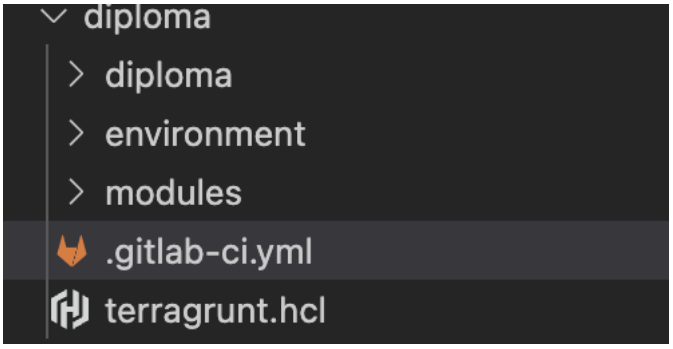


Рисунок 3.15 Стадії пайплайну

На стадії lint код буде проходити первинну перевірку на відстуність помилок у форматуванні коду.

На стадії validate код буде проходити другий етап перевірки на правильність написання ресурсів та видасть помилку у випадку, якщо не вистачає змінних, або ресурс не встановлено.

На стадії plan буде відбуватися планування виконання коду. Terragrunt змоделює те, як код буде виконуватись та спробує зпрогнозувати помилки, а також покаже, що модуль буде робити: що видалить, або створить.

На стадії apply буде виконано код. Без будь яких додаткових перевірок.

Необхідно почати з першої стадії lint. Для цього буде використано інструмент tflint (рис 3.16). Який був описаний у попередньому розділі.

У задачі CI/CD пайплайну використовуються настпуні ключові слова:

stage позначаю стадію на якій задача повинна виконуватись.

Rules позначає правила, за якими задача буде запусккатись автоматично, або за допомогою ручних маіпуляцій.

Script основний блок, в якому пишеться сценарій, який буде виконано протяго роботи задачі.

Для виконання lint необхідно зайти в директорію з модулем, який буде перевірятися та виконати воману.

```
lint_eks:
  stage: lint
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME =~ /(^master$)/'
      when: manual
  script:
    - cd modules/eks
    - tflint -v
    - tflint --disable-rule=terraform_required_providers
```

Рисунок 3.16 Виконання lint для модулю EKS

Друга стадія validate. Буди виконуватись таким самим чином та матиме ті самі ключові слова, що і попередній.

Validate буде виконуватись самим Terraform для перевіки форматування граматичної правильності коду (рис 3.17).

```
validate_eks:
  stage: validate
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME =~ /(^master$)/'
      when: manual
  script:
    - cd modules/eks
    - terraform init
    - terraform validate
    - terraform fmt -list=true -write=false -diff=true -check=true -recursive
```

Рисунок 3.17 Validate стадія

Наступний крок виконання plan. Як вже було зазначено, дипломна робота буде виконана на оточенні staging, тому для виконання плану, необхідно зайти у директорію staging та виконати команди.

```
plan:
  stage: plan
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME =~ /(^master$)/'
      when: manual
  script:
    - cd diploma/eu-west-1/staging
    - terragrunt run-all init
    - terragrunt run-all plan
```

Рисунок 3.18 Виконання Plan

Останній крок виконання коду, воно буде доступно тільки у тому випадку, якщо всі попередні стадії пройдуть вдало.

Для виконання останньої стадії необхідно виконати ті ж самі команди, що і у попередній стадії, але команда «plan» зміниться на команду «apply» (рис. 3.19).

```
apply:
  stage: apply
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME =~ /(^master$)/'
      when: manual
  script:
    - cd diploma/eu-west-1/staging
    - terragrunt run-all init
    - terragrunt run-all apply
```

Рисунок 3.20 Виконання «apply»

Повний код пайплайну наведено у додатку.

Висновки до розділу 3

Під час розробки розділу було проаналізовані одноплатні комп’ютери представлені на ринку, обрано одноплатний комп'ютер Raspberry PI.

Було проаналізовано існуючі дистрибутиви на базі Linux та обрано дистрибутив CentOS, який один із найстабільніших та з найменшими системними вимогами.

Було написано код CI/CD пайплайну, який буде використано протягом наступного розділу та протестовано.

РОЗДІЛ 4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

4.1. Локальне тестування написаного коду Terraform та Terragrunt

Після виконання всіх вище зазначених кроків, можна приступати до тествунання написаного коду. Для цього необхідно перейти у директорію «staging» та перш за все виконати команду «terragrun run-all init» (рис 4.1).

Рисунок 4.1 Команда «terragrun run-all init»

Команда рекурсивно виконає перегляд всіх ресурсів та встановить всі провайдери, які було описано у коді, а також перевірить версії встановлених провайдерів.

Нуступним кроком необхідно встановити змінні оточення на комп’ютер, на якому виконується (рис 4.2). Повинно бути 3 змінні оточення:

1. AWS_ACCESS_KEY_ID змінна оточення з ідентифікатором AWS аккаунту, який був згенерований на почтаку розділу
2. AWS_SECRET_ACCESS_KEY API ключ від AWS аккаунту, який був згенерований проятг цього розділу
3. AWS_DEFAULT_REGION регіон до якого треба звертатися за замовчуванням

```
> export AWS_ACCESS_KEY_ID=xxx
> export AWS_SECRET_ACCESS_KEY=xxx
> export AWS_DEFAULT_REGION=eu-west-1
```

Рисунок 4.2 Встановлення змінних оточення

Останнім кроком необхідно виконати команду «terragrunt run-all plan». Під час виконання цієї команди Terragrunt спробує завантажити модуль, створити файл стану, зробити перевірку конфігурації, а також покаже те, що він збирається зробити у хмарному провайдері, а саме видалити, створити, або модифікувати будь які ресурси.

Terragrunt створює план виконання, який описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та конфігурації.

У випадку якщо Terragrunt не виявить помилок та покаже ресурси (рис. 4.3), які планує створити завдання виконане правильно.

```
Plan: 10 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + arn          = (known after apply)
```

Рисунок 4.3 Виконання команди terragrunt run-all plan

4.2. Raspberry PI цільва платформа для виконання коду

Цільової платформую для виконання коду було обрано платформу мікрокомп’ютеру Raspberry PI. З наступними характеристиками:

1. RAM: 2GB
2. CPU: 4 ARM cores

3. Disk: 32GB
4. OS: Linux based OS

Raspberry Pi гарно підходить тим, що у період постіних відключень електроєнергії працює від джерела живлення 5V, що є дуже економним варіантом у порівнянні з 220V, яке використовує класичні сервери. Це значить, що Raspberry PI може живитися від Power Bank.

Raspberry Pi (рис. 4.4) це недорогий комп'ютер розміром із кредитну картку, який підключається до комп'ютерного монітора або телевізора та використовує стандартну клавіатуру та мишу. [18]

Рисунок 4.4 Raspberry PI 4 без корпусу

Це функціональний маленький пристрій, який дозволяє людям різного віку вивчати обчислювальну техніку і вчитися програмувати такими мовами, як Scratch і Python. Він здатний робити все, що здатний робити звичайний настільний комп'ютер, від роботи в Інтернеті та відтворення відео високої чіткості до створення електронних таблиць,

обробки текстів та ігор. А також виконання будь якого коду, сумісного з архітектурою процесору мікрокомп’ютера.

Для виконання дипломної роботи було обрано систему CentOS. Як одну з найпопулярніших систем на базі Linux.

Для встановлення CentOS на Raspberry Pi 4 необхідно мати карту пам’яті мінімум на 16 GB. Але для дипломної роботи було обрана карта пам’яті на 32 GB. А також комп’ютер або ноутбук з адаптером для читання та записування данних на карту пам’яті.

Загальна схема роботи програмно-апаратного комплексу виглядає наступним чином (рис. 4.5).

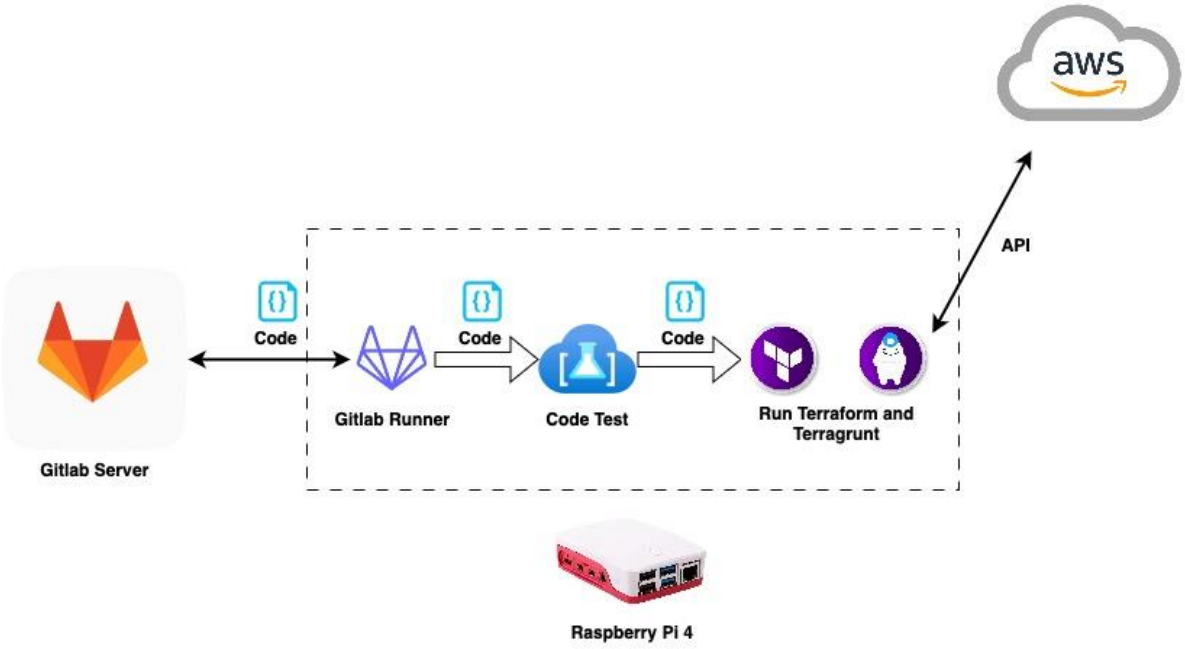


Рисунок 4.5 Загальна схема роботи програмно-апаратного комплексу

4.3. Тестування CI/CD пайплайну

Тестування написаного коду буде вібуватися за допомогою вже існуючого екземпляру серверу Gitlab.

А також налаштованого та підключеного до інтернет мережі Raspberry PI. Після виконання минулих розділів просто увімкнення одноплатного комп’ютера достатньо для того, щоб екзекютор завантажився та почав працювати.

Первши кроком буде виконаний пуск пайплайну, адже пайплайн був зпроєктований запускатися лишу вручну.

Необхідно перевірити правильність написання коду Terraform, правильність структури Terragrunt, а також правильність написання коду пайплайну (рис 4.5).

```
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
$ terraform validate
Success! The configuration is valid.
$ terraform fmt -list=true -write=false -diff=true -check=true -recursive
Cleaning up project directory and file based variables
Job succeeded
```

Рисунок 4.5 Виконання validate у gitlab

З рисунку 4.5 видно, що модуль eks був написаний правильно та функція validate пройшла успішно.

Також необхідно перевірити lint модулю (рис 4.6).

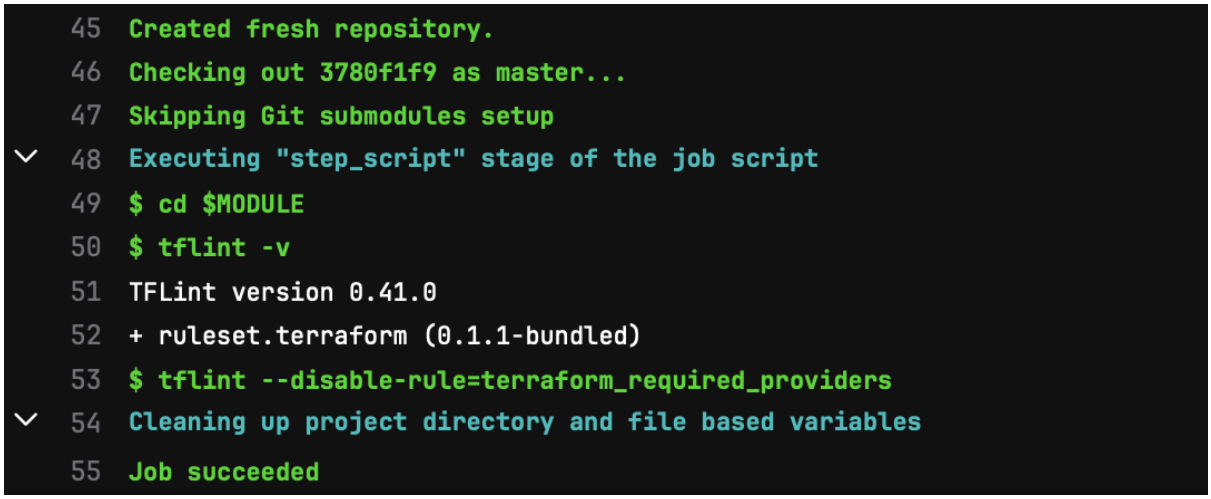


Рисунок 4.6 Виконання lint у gitlab

Отже, виходячи з отриманої інформації, можна зробити висноко, що конфігурація записана у Terraform modules, структура каталогів Terragrunt, код пайплайну та налаштуванн одноплатного комп’ютера вірні. А апаратно-програмна реалізяція проєкту готова до використання. Останнім тестом необхідно виконати план майбутньої інфраструктури (рис. 4.7).

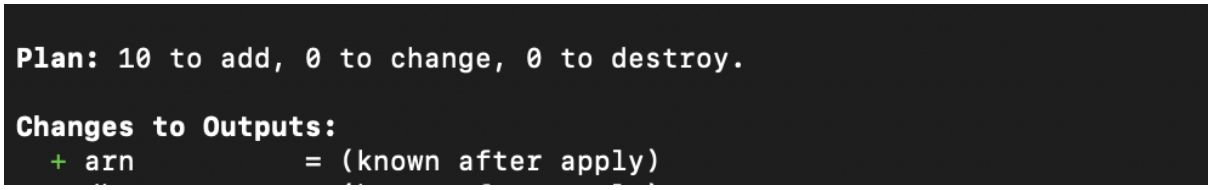


Рисунок 4.7 Виконання Plan

4.4. Керівництво користувача

Програмного-апаратний модуль може використовуватись у багатьох варіаціях, для базового використання програмно-апаратного модулю необхідно:

1. Мати обліковий запис хмарного провайдеру
2. Мати API ключі для облікого запису

3. Джерело живлення для одноплатного комп'ютеру та доступ до інтернет.

Необхідно завантажити та встановити змінні оточення для хмарного провайдеру та програмно-апаратний модуль готовий до використання.

У випадку, коли програмно-апаратний модуль необхідно підключити до існуючої CI/CD системи, треба встановити додаткові інструменти накшталт Gitlab runner.

Висновки до розділу 4

Під час розробки розділу було виконане кінцеве налаштування програмно-апаратного модулю, проведене тестування, а також розроблене керівництво користувача.

Проаналізувавши отримані результати, можна зробити висновок, що програмно-апаратний модуль працює правильно та може бути використаний для розгортання, оновлення, повного або часткового перенесення IT-інфраструктури, а також, модулб енергоефективний у порівнянні з класичними серверами.

ВИСНОВКИ

Виходячи з мети дипломної роботи та сформульованих завдань, отримані наступні результати:

Досліджено інструменти Infrastructure as code та проаналізовано найпопулярніші рішення представлені на ринку.

Проаналізовано екзекютори коду на основі Linux Based OS. Для виконання роботи було обрано Gitlab Runner написаний на крос платформеній мові програмування Go.

Розроблено програмний код Terraform та структуру каталогів Terragrunt для виконання коду.

Проведено аналіз інсуючих одноплатних комп'ютерів та обрано оптимальний варіант. Також обрано операційну систему для одноплатного комп'ютеру.

Було проведено тести написаного коду на локальному ком’ютері, а також налаштовано одноплатний комп’ютер Raspberry PI для виконання коду.

Протягом виконання дипломної роботи було приєднано одноплатний комп’ютер до CI/CD системи та автоматизовано процес розгортання та оновлення кластерної системи. А також

Розроблено розділ “Охорона праці”. Проаналізовані основні правила охорони праці та безпеки життєдіяльності під час робочого процесу.

Отже, мета дипломної роботи була досягнена. Протягом дипломної роботи було розроблено програмно-апаратний модуль на базі Raspberry PI, який готовий до використання та полегшує створення, відновлення та переміщення інфраструктури, написаний код та структуру каталогів для інструменту IaC Terragrunt, налаштовано автоматичний процес розгортання та оновлення кластерної системи. А також збережена можливість перевикористовувати написаний код для зміненя, або

переміщення конфігурації кластерної системи за допомогою одноплатного комп’ютеру.

Крім того, програмно-апаратний муль може працювати з різними CI/CD системами за рахунок того, що у його основі лежить Linux based дистрибутив CentOS. Розроблений модуль має низьке енергоспоживання, що є особливо актуальним у даний час, адже Raspberry PI живиться від 5V та 3A кабулю роз’ємом Type-C, що означає щл екзекютор може використовувати живлення від Power Bank.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кушнір С. Ю., Пузирьов С. В. Кластерна серверна система на базі Raspberry PI та Terragrunt. Інформаційні технології та інженерія : тези доп. Всукр. наук. - практ. Конф. Миколаїв, 7-10 лютого 2023. Миколаїв : Чорном. нац. Ун-т. ім. Петра Могили, 2023. С. 78
2. Опис IaC RedHat [сайт]. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (Дата звернення 29.01.2023)
3. Опис Terragrunt GruntWork [сайт]. URL: https://terragrunt.gruntwork.io/#:~:text=Terragrunt%20is%20a%20thin%20wrapper,Get%20Started (Дата звернення 29.01.2023)
4. Akond Rahman, Rezvan Mahdavi-Hezaveh, Laurie Williams. A systematic mapping study of infrastructure as code research: A review. Publ. 2019. URL: https://www.mendeley.com/catalogue/732ee5a8-ea92-3fbb-9722-c63c0e154247/ (Last access: 29.01.2023)
5. Опис Chef Edureka [сайт]. URL: https://www.edureka.co/blog/what-is-chef/
6. Порівняння Chef, Ansible, Terraform IBM [сайт]. URL: https://www.ibm.com/cloud/blog/chef-ansible-puppet-terraform (Дата звернення 29.01.2023)
7. Огляд Puppet Puppet [сайт]. URL: https://www.puppet.com/docs/puppet/6/puppet_overview.html (Дата звернення 29.01.2023)
8. Ansible for IaC IBM [сайт]. URL: https://developer.ibm.com/articles/ansible-iac-for-faster-deployment-on-aix/ (Дата звернення 30.01.2023)
9. Як працює Ansible Ansible [сайт]. URL: https://www.ansible.com/overview/how-ansible-works (Дата звернення 30.01.2023)

10. Визначення SaltStack [сайт]. URL: https://www.techtarget.com/searchitoperations/definition/SaltStack (Дата звернення 31.01.2023)
11. Terraform HashiCorp [сайт]. URL: https://developer.hashicorp.com/terraform/intro (Дата звернення 01.02.2023)
12. Керівництво користувача AWS CloudFormation AWS [сайт]. URL: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html (Дата звернення 02.02.2023)
13. Опис IDE AWS [сайт]. URL: https://aws.amazon.com/what-is/ide/ (Дата звернення 02.02.2023)
14. Visual Studio Code переваги VisualStudio [сайт]. URL: https://code.visualstudio.com/docs/editor/whyvscode (Дата звернення 03.02.2023)
15. Розширення Visual Studio Code VisualStudio [сайт]. URL: https://code.visualstudio.com/docs/editor/extension-marketplace (Дата звернення 03.02.2023)
16. Встановлення Terraform HashiCorp [сайт]. URL: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli
17. Cloud Provider AWS AWS [сайт]. URL: https://aws.amazon.com/what-is-aws/ (Дата звернення 03.02.2023)
18. Описання Gitlab Simple Learn [сайт]. URL: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab#what_is_gitlab (Дата звернення 03.02.2023)
19. Опис Raspberry PI Raspberry PI [сайт]. URL: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ (Дата звернення 04.02.2023)

20. Архітектура ARM64 JumpCloud [сайт]. URL: https://jumpcloud.com/blog/why-should-you-use-arm64 (Дата звернення 04.02.2023)
21. Gitlab екзекютор Gitlab [сайт]. URL: https://docs.gitlab.com/runner/ (Дата звернення 04.02.2023)
22. CentOS for Raspberry PI Pimylife [сайт]. URL: https://pimylifeup.com/raspberry-pi-centos/ (Дата звернення 05.02.2023)
23. Опис OS TechTarget [сайт]. URL: https://www.techtarget.com/whatis/definition/operating-system-OS (Дата звернення 06.02.2023)
24. Linux Kernel Graphiz [сайт]. URL: https://graphviz.org/Gallery/directed/Linux_kernel_diagram.html (Дата звернення 07.02.2023)
25. HackBoard CrowdSupply [сайт]. URL: https://www.crowdsupply.com/hackboard/hb2 (Дата звернення 07.02.2023)
26. CentOS Hostinger [сайт]. URL: https://www.hostinger.com/tutorials/what-is-centos (Дата звернення 07.02.2023)

ДОДАТОК А
Код модулів Terraform

Модуль EKS cluster:

```
terraform {
  required_version = "> 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.72.0"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "2.10.0"
    }
    http = {
      source  = "terraform-aws-modules/http"
      version = "2.4.1"
    }

    tls = {
      source  = "hashicorp/tls"
      version = "3.1.0"
    }
  }
}
```

```
provider "tls" {
}

provider "http" {
}

data "aws_region" "current" {}

data "aws_partition" "current" {}

data "aws_caller_identity" "current" {}

module "eks" {
  source                          = "terraform-aws-modules/eks/aws"
  version                         = "18.20.0"
  cluster_name                    = var.eks_cluster_name
  cluster_version                 = var.cluster_version
  subnet_ids                      = var.private_subnets
  cluster_endpoint_private_access = true
  cluster_endpoint_public_access  = true
  enable_irsa                     = true

  cluster_addons = {
    vpc-cni = {
      resolve_conflicts = "OVERWRITE"
    }
  }
```

```
  tags = {
    "Name"                 = var.eks_cluster_name
    "Environment"          = var.environment
    "karpenter.sh/discovery" = var.eks_cluster_name
  }

  vpc_id = var.vpc_id

  manage_aws_auth_configmap = true

  aws_auth_roles = [
    {
      rolearn                                                                 =
"arn:${data.aws_partition.current.partition}:iam::${data.aws_caller_identity.cur
rent.account_id}:role/${aws_iam_role.nodes_iam_role.name}"
      username = "system:node:{{EC2PrivateDNSName}}"
      groups = [
        "system:bootstrappers",
        "system:nodes"
      ]
    },
    {
      rolearn                                                                 =
"arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/eks_admin_r
ole_${var.eks_cluster_name}"
      username = "eks-admin"
      groups = [
        "system:masters"
```

```
      ]
    },
    {
      rolearn                                                    =
"arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/eks_develop
er_role_${var.eks_cluster_name}"
      username = "eks-developer"
      groups = [
        local.developer_cluster_role
      ]
    }
  ]
}

resource "time_sleep" "wait_1_minute" {
  create_duration = "1m"

  depends_on = [
    module.eks
  ]
}

locals {
  developer_cluster_role = "developer-access-role"
  kubeconfig = replace(yamlencode({
    apiVersion = "v1"
    clusters = [{
      cluster = {
```

```
    "certificate-authority-data" = module.eks.cluster_certificate_authority_data
    server                       = module.eks.cluster_endpoint
   }
   name = var.eks_cluster_name
  }]
  contexts = [{
   context = {
    cluster = var.eks_cluster_name
    user    = var.eks_cluster_name
   }
   name = var.eks_cluster_name
  }]
  current-context = var.eks_cluster_name
  kind            = "Config"
  preferences     = {}
  users = [{
   name = var.eks_cluster_name
   user = {
    exec = {
     apiVersion = "client.authentication.k8s.io/v1beta1"
     args = [
      "token",
      "-i",
      var.eks_cluster_name,
      "-r",

"arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/eks_USER_r
ole_${var.eks_cluster_name}"
```

```
        ]
        command           = "aws-iam-authenticator"
        env               = "null"
        provideClusterInfo = "false"
       }
     }
     }
   ]
 }), "\"", "")
}

resource "local_file" "kubeconfig" {
  for_each = toset(["admin", "developer"])

  content              = replace(local.kubeconfig, "USER", each.value)
  file_permission      = "0644"
  filename                                                                    =
"${var.kubeconfig_path}/${each.value}.${var.eks_cluster_name}.kubeconfig.y
ml"
  directory_permission = "0755"

  depends_on = [
    module.eks
  ]
}

provider "kubernetes" {
  host                   = module.eks.cluster_endpoint
```

```
  cluster_ca_certificate                                          =
base64decode(module.eks.cluster_certificate_authority_data)

  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name", var.eks_cluster_name]
    command     = "aws"
  }
}
```

Модуль node_groups:

```
terraform {
  required_version = "> 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.72.0"
    }
  }
}

data "aws_iam_role" "nodes_iam_role" {
  name = var.nodes_iam_role_name
}
resource "aws_eks_node_group" "k8s_system" {
  count           = var.nodes.k8s_system.count
  cluster_name    = var.eks_cluster_name
```

```
node_group_name = "${var.project_name}-k8s-system"
node_role_arn   = data.aws_iam_role.nodes_iam_role.arn
subnet_ids      = var.private_subnets

ami_type      = "BOTTLEROCKET_ARM_64"
capacity_type = "ON_DEMAND"
disk_size     = "20"

instance_types = [
  "m6g.medium"
]

tags = {
  "nodetype"      = "system"
  provider_name   = "aws"
}

labels = {
  "k8s-app"  = "k8s-dns"
  "nodetype" = "system"
}

scaling_config {
  desired_size = 3
  max_size     = 3
  min_size     = 2
}
```

```
  update_config {
    max_unavailable = 1
  }

}
resource "aws_eks_node_group" "k8s_worker" {
  count          = var.nodes.k8s_worker.count
  cluster_name    = var.eks_cluster_name
  node_group_name = "${var.project_name}-k8s-worker"
  node_role_arn   = data.aws_iam_role.nodes_iam_role.arn
  subnet_ids      = var.private_subnets

  ami_type      = "BOTTLEROCKET_ARM_64"
  capacity_type = "ON_DEMAND"
  disk_size     = "20"

  instance_types = var.nodes.k8s_worker.instance_types

  lifecycle {
    ignore_changes = [scaling_config[0].desired_size]
  }

  tags = {
    "nodetype"      = "worker"
    provider_name   = "aws"

  }
```

```
  labels = {
    "k8s-app"  = "k8s-worker"
    "nodetype" = "worker"
  }

  taint {
    key    = "nodetype"
    value  = "worker"
    effect = "NO_EXECUTE"
  }

  scaling_config {
    desired_size = var.nodes.k8s_worker.instance_desired_size
    max_size     = 20
    min_size     = var.nodes.k8s_worker.instance_desired_size
  }

  update_config {
    max_unavailable = 1
  }
```

Код CI/CD пайплайну

```
stages:
  - lint
  - validate
  - plan
  - apply

lint_eks:
```

```
  stage: lint
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
    - cd modules/eks
    - tflint -v
    - tflint --disable-rule=terraform_required_providers

lint_node_groups:
  stage: lint
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
    - cd modules/node_groups
    - tflint -v
    - tflint --disable-rule=terraform_required_providers

validate_eks:
  stage: validate
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
```

```
    - cd modules/eks
    - terraform init
    - terraform validate
    - terraform fmt -list=true -write=false -diff=true -check=true -recursive

validate_node_groups:
  stage: validate
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
    - cd modules/node_groups
    - terraform init
    - terraform validate
    - terraform fmt -list=true -write=false -diff=true -check=true -recursive

plan:
  stage: plan
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
    - cd diploma/eu-west-1/staging
    - terragrunt run-all init
    - terragrunt run-all plan
```

```
apply:
  stage: apply
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web" && $CI_COMMIT_REF_NAME
=~ /(^master$)/'
      when: manual
  script:
    - cd diploma/eu-west-1/staging
    - terragrunt run-all init
    - terragrunt run-all apply
```

ДОДАТОК Б

Апробація

КЛАСТЕРНА СЕРВЕРНА СИСТЕМА НА БАЗІ RASPBERRY PI ТА TERRAGRUNT

Невід’ємна частина будь якої великої сучасної IT-інфраструктури – це інструменти IaC. Infrastructure as code. Інфраструктура як код (IaC) — це управління і розгортання інфраструктури за допомогою коду, а не за допомогою ручних процесів. За допомогою IaC створюються файли конфігурації, які містять специфікації інфраструктури, що полегшує редагування та розповсюдження конфігурацій. Це також гарантує, що кожного разу розгортається одне й те саме середовище. [1]

Як інструмент для побудови інфраструктури за допомогою коду було обрано Terragrunt. Terragrunt — це тонка обгортка, яка надає додаткові інструменти для збереження конфігурацій у без повторення коду, роботи з кількома модулями Terraform і керування віддаленим станом. [2]

IaC є важливою частиною впровадження практик DevOps і безперервної інтеграції/безперервної доставки (CI/CD). IaC забирає більшість роботи з ініціалізації у розробників, які можуть виконати сценарій, щоб підготувати свою інфраструктуру до роботи.

Таким чином, розгортання програм не затримується в очікуванні інфраструктури, а системні адміністратори не виконують трудомісткиі ручні процеси.

Цільова платформа для виконання коду – Raspberry PI. Ресурсні вимоги до платформи:

1) RAM: 2GB
2) CPU: 4 ARM cores
3) Disk: 32GB
4) OS: Linux based OS

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Опис IaC RedHat [сайт]. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (Дата звернення 29.01.2023)
2. Опис Terragrunt GruntWork [сайт]. URL: https://terragrunt.gruntwork.io/#:~:text=Terragrunt%20is%20a%20thin%20wrapper,Get%20Started (Дата звернення 29.01.2023)
3. Akond Rahman, Rezvan Mahdavi-Hezaveh, Laurie Williams. A systematic mapping study of infrastructure as code research: A review. Publ. 2019. URL: https://www.mendeley.com/catalogue/732ee5a8-ea92-3fbb-9722-c63c0e154247/ (Last access: 29.01.2023)