Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Автоматизація інфраструктури за допомогою Terraform та Ansible

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-2
Микола МУКОВОЗ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Дар’я ТИМОШЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Муковозу Миколі Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Автоматизація інфраструктури за допомогою Terraform та Ansible

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи
Тема дослідження
Офіційна документація Terraform, Ansible та Kubernetes
Файли конфігурацій інструментів Terraform та Ansible

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз літератури та практик з тематики Infrastructure as Code.
Огляд інструментів Terraform і Ansible: архітектура, синтаксис, сценарії застосування
Налаштування інвентаря та виконання плейбуків Ansible для встановлення Kubernetes
Побудова структури Terraform-проєкту: модулі, змінні, outputs, шаблони
Автоматизоване розгортання EC2-інстансів, SSH-ключів, груп безпеки та мережевих

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технологій та інструментів	31.05.25-01.06.25	
3	Розробка основних плейбуків і модулів	02.06.25-06.06.25	
4	Тестування та відлагодження	07.06.25-09.06.25	
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Дар'я ТИМОШЕНКО.
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 56 с., 23 рис., 0 табл., 1 дод., 13 джерел.

AWS, INFRASTRUCTURE AS CODE, ANSIBLE, TERRAFORM, НОДА, POD, K8S, КОНТЕЙНЕР, DOCKER.

Метою кваліфікаційної роботи є розробка, впровадження та дослідження методики автоматизованого розгортання кластерної інфраструктури Kubernetes у середовищі хмарної платформи Amazon Web Services (AWS) з використанням технологій Infrastructure as Code, зокрема інструментів Terraform для управління інфраструктурою та Ansible для конфігурації програмного забезпечення, з урахуванням сучасних підходів до масштабування, безпеки та підтримки життєвого циклу хмарних сервісів.

У ході виконання кваліфікаційної роботи було розроблено комплексне рішення для автоматизованого розгортання кластерної інфраструктури Kubernetes у хмарному середовищі Amazon Web Services (AWS). Основою підходу стало використання принципів Infrastructure as Code (IaC), що дозволило описати інфраструктуру декларативним способом та забезпечити її повторюваність, масштабованість і керованість. Засобами інфраструктурної автоматизації виступили Terraform для створення віртуальних ресурсів AWS (EC2-інстанси, групи безпеки, ключі доступу) та Ansible — для встановлення та конфігурації кластерного середовища Kubernetes. Було реалізовано повний життєвий цикл розгортання: від ініціалізації мережевих ресурсів до запуску майстер- та робочих вузлів з попередньо визначеними параметрами. Проведене тестування підтвердило працездатність запропонованого рішення, його гнучкість, а також можливість адаптації до різних сценаріїв використання в промислових та наукових середовищах.

ABSTRACT

Bachelor’s thesis: 56 pages, 23 figures, 0 tables, 1 appendices, 13 sources.

AWS, INFRASTRUCTURE AS CODE, ANSIBLE, TERRAFORM, NODE, POD, K8S, CONTAINER, DOCKER.

The major goal of this thesis is to develop, implement, and analyze a methodology for the automated deployment of a Kubernetes cluster infrastructure within the Amazon Web Services (AWS) cloud environment. This process is based on Infrastructure as Code (IaC) principles, employing tools such as Terraform for infrastructure provisioning and Ansible for software configuration, with consideration of modern approaches to scalability, security, and lifecycle management of cloud-based services.

In the course of the thesis work, a comprehensive solution was designed and implemented to automate the deployment of Kubernetes cluster infrastructure on AWS. The approach was grounded in the principles of Infrastructure as Code, enabling a declarative description of the environment and ensuring its repeatability, scalability, and manageability. Terraform was used to provision AWS virtual resources (EC2 instances, security groups, EC2 key pairs), while Ansible facilitated the installation and configuration of the Kubernetes components. The full deployment lifecycle was realized — from initializing network resources to launching controller and worker nodes with predefined specifications. Experimental validation confirmed the functionality, flexibility, and adaptability of the proposed solution for use in both industrial and academic contexts.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application Programming Interface

AWS – Amazon Web Services

CI/CD – Continuous Integration / Continuous Delivery

DNS – Domain Name System

EC2 – Elastic Compute Cloud

GCP – Google Cloud Platform

HCL – HashiCorp Configuration Language

IAM – Identity and Access Management

IaC – Infrastructure as Code

IT – Information Technology

NoSQL – Not Only SQL

PaaS – Platform as a Service

RDS – Relational Database Service

SG – Security Group

TF – Terraform

VPC – Virtual Private Cloud

YAML – YAML Ain’t Markup Language

ВСТУП

Процеси створення та випуску програмного забезпечення характеризуються значною складністю, що обумовлюється постійним ускладненням архітектури застосунків, зростанням команд розробників та необхідністю підтримки різноманітної внутрішньої інфраструктури. Зі збільшенням масштабів і комплексності проєктів виникають додаткові виклики, які вимагають удосконалення підходів до автоматизації та управління процесами розробки. З цією метою були розроблені та широко застосовуються взаємопов’язані концепції — безперервна інтеграція (Continuous Integration, CI) та безперервна доставка (Continuous Delivery, CD).

Безперервна інтеграція передбачає регулярне та часте внесення змін у центральний репозиторій, що дозволяє виявляти потенційні проблеми на ранніх етапах і сприяє ефективній колаборації між розробниками. Зазвичай інтеграція проводиться кілька разів на день, забезпечуючи постійний контроль якості коду та швидку реакцію на помилки чи конфлікти.

Наступним етапом після інтеграції є безперервна доставка, яка передбачає автоматизацію всіх необхідних кроків для безпечного і послідовного розгортання програмного забезпечення. Її основною перевагою є готовність до випуску будь-якої частини коду в будь-який момент часу, завдяки чому команда розробників може швидше реагувати на потреби ринку і замовника.

Безперервна інтеграція та безперервна доставка є не лише набором технічних практик, але й філософією та культурою розробки програмного забезпечення, що забезпечує постійну еволюцію і стабільну якість продукту. Вони є невід’ємною складовою методології DevOps, яка спрямована на зменшення розриву між розробкою і експлуатацією програмного забезпечення. Завдяки автоматизації процедур розгортання розробники

можуть зосередитися на вирішенні завдань бізнесу, підвищенні якості та забезпеченні безпеки коду.

У технічному сенсі, безперервна інтеграція включає в себе автоматичне збирання, пакування та тестування програмних компонентів після кожної зміни коду. Такий підхід гарантує негайну перевірку сумісності та працездатності всіх частин додатку, покращує комунікацію всередині команди та підвищує якість продукту в цілому.

Безперервна доставка розширює цей процес, автоматизуючи етапи випуску програмного забезпечення у різні середовища, такі як тестові, стадійні чи продуктивні. В результаті забезпечується швидке і передбачуване розгортання, що значно скорочує час від розробки до безпосереднього використання продукту кінцевими користувачами.

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ INFRASTRUCTURE AS CODE

1.1 Поняття Хмарних Технологій та їх види

Хмарні технології - це технології розподілених цифрових обчислень, які роблять комп'ютерні ресурси доступними для користувача Інтернету як онлайн-сервіс. Запуск програм і відображення їх результатів відбувається у вікні веб-браузера на локальному комп'ютері. При цьому всі програми та їхні дані, необхідні для роботи, знаходяться на віддаленому веб-сервері і тимчасово зберігаються на стороні клієнта: на персональному комп'ютері тощо.

Перевага цієї технології полягає в тому, що користувач має доступ до власних даних, але не повинен турбуватися про інфраструктуру, операційну систему та програмне забезпечення, з яким він працює. Слово «хмара» - це метафора складної інфраструктури, яка приховує всі технічні деталі . Існують наступні категорії хмарних технологій.

Публічна хмара - одночасний доступ декількох користувачів до IT-інфраструктури. Однак користувачі не мають можливості керувати та підтримувати цю хмару, вся відповідальність покладається на власника хмари. Будь-яка компанія або приватна особа може підписатися на пропоновані послуги[2].

Приватна хмара - це IT-інфраструктура, яка контролюється та експлуатується окремим абонентом у власних інтересах. Інфраструктура для управління приватною хмарою може знаходитися на вашій території або на території зовнішнього оператора, або вона може бути частиною приватної хмари.

Гібридна хмара - це IT-інфраструктура, яка поєднує в собі найкращі риси публічної та приватної хмар. Така мережа складається з окремих

об'єктів, які з'єднані між собою за допомогою стандартних або пропрієтарних технологій, що дозволяють передавати дані або додатки між компонентами.

Існує три основні рівні хмарних обчислень.

Перший рівень - це інфраструктура як послуга (IaaS), де користувачі мають доступ до основних IT-ресурсів, таких як процесори та пристрої зберігання даних, і можуть розробляти власні операційні системи та додатки. Хоча вони не керують фізичною інфраструктурою, користувачі мають контроль над операційними системами, сховищами і додатками, що використовуються, а також обмежений контроль над мережевими компонентами.

Другий рівень - платформа як послуга (PaaS), де користувачі можуть встановлювати власні додатки на платформі провайдера без управління базовою інфраструктурою - серверами, мережами, операційними системами і системами зберігання - але мають контроль над розгорнутими додатками і деякими параметрами конфігурації середовища.

Найвищий рівень - це програмне забезпечення як послуга (SaaS), де дані і додатки зберігаються в хмарі і доступні користувачеві через веб-браузер.

1.2 Існуючі рішення хмарних платформ

Сьогодні існує багато платформ, які використовують технології хмарних обчислень і пропонують функціональність, необхідну для автоматизації процесів розгортання та забезпечення безперервної інтеграції клієнт-серверних систем.

Платформи хмарних обчислень - це готові програмні та апаратні засоби, які орендуються через Інтернет для розгортання, розробки та тестування додатків.

Ідея платформ хмарних обчислень виникла в Інтернеті, але перша платформа хмарних обчислень з'явилася лише у 2006 році під назвою

Amazon Web Services (хоча компанія почала пропонувати доступ до обчислювальних ресурсів через Інтернет ще у 2002 році). До 2006 року Amazon пропонувала понад 50 різних послуг у 14 географічних регіонах.

Наступною великою платформою стала Microsoft Azure, яка була запущена в 2010 році (зараз вона є другою після Amazon за кількістю користувачів та послуг). У 2011 році світ став свідком запуску третього великого гравця на ринку хмарних обчислень - Google Cloud Platform. Через рік кілька компаній почали активно створювати нові платформи та сервіси, що призвело до падіння цін на оренду платформ. Були створені такі сервіси, як i-Teco OpenStack Cloud, Azure Cloud OS, Jelastic для PHP і Java додатків та інші.

Сьогодні хмарні платформи популярні як ніколи. Основними перевагами використання хмарних платформ є швидка розробка нових додатків, гнучкість і масштабованість системи.

При використанні хмарних платформ клієнтам потрібно організувати лише одне робоче місце (ПК/ноутбук/планшет), незалежно від того, де вони знаходяться. Єдина вимога - підключення до Інтернету. Клієнт (користувач хмарної платформи) керує лише власними додатками та програмним забезпеченням.

За всі інші аспекти, такі як управління базами даних, віртуалізація, мережеве та серверне обладнання, управління операційними системами та обслуговування інфраструктури центру обробки даних, відповідає постачальник хмарної платформи, якого часто називають хмарним провайдером. Окрім платформи для виконання додатків, хмарна платформа також пропонує оренду хмарного сховища та підтримку таких технологій, як машинне навчання, штучний інтелект і великі дані.

Для забезпечення безпеки часто використовується окрема VPN і мережева адреса, виділений брандмауер і гнучкі налаштування DNS.

Ціна оренди хмарної платформи зазвичай складається з плати за обчислювальну потужність (включаючи технічне обслуговування

обладнання), витрат на ліцензії, програмне забезпечення/операційні системи/системи віртуалізації, що використовуються, та плати за послуги хмарного провайдера як постачальника хмарної платформи.

Хмарні платформи використовуються компаніями в багатьох сферах, наприклад, для інтелектуальних систем страхування, телеметрії в хмарі, юзабіліті-тестування власного програмного забезпечення або веб-сайтів, систем онлайн-освіти, бронювання готелів або оренди житла, систем ідентифікації в онлайн-банкінгу тощо.

1.2.1 Amazon Web Services

У 2006 році компанія Amazon Web Services (AWS) почала пропонувати підприємствам послуги IT-інфраструктури через свою систему веб-сервісів, які зараз зазвичай називають хмарними обчисленнями. Однією з ключових переваг хмарних обчислень є можливість замінити авансові капітальні витрати на інфраструктуру низькими змінними витратами, які адаптуються до потреб бізнесу. Завдяки хмарним обчисленням компаніям більше не потрібно планувати і купувати сервери та іншу IT-інфраструктуру за кілька тижнів чи місяців до початку роботи. Замість цього вони можуть негайно розгорнути сотні або тисячі серверів за лічені хвилини і швидше отримати результати.

Сьогодні Amazon Web Services пропонує високонадійну, масштабовану та економічно ефективну інфраструктурну платформу.

AWS надає масивну глобальну хмарну інфраструктуру, яка дозволяє швидко впроваджувати інновації, експериментувати та ітерації. Замість того, щоб тижнями чи місяцями чекати на апаратне забезпечення, ви можете миттєво розгортати нові додатки, масштабувати їх у міру зростання робочих навантажень і вимикати, коли це необхідно. Незалежно від того, чи потрібен вам один віртуальний додаток або тисячі, чи потрібен він вам лише на кілька годин або 24 години на добу, ви платите лише за те, що використовуєте.

Платформа AWS не залежить від мови та операційної системи. Ви можете вибрати платформу для розробки або модель програмування, яка найкраще відповідає вашим завданням. Ви можете вибрати одну або кілька послуг, якими хочете користуватися, а також спосіб їх використання. Така гнучкість дозволяє вам зосередитися на інноваціях, а не на інфраструктурі.

AWS — це безпечна та надійна технологічна платформа, сертифікована та перевірена в галузі. Послуги та центри обробки даних мають багато рівнів операційної та фізичної безпеки для забезпечення цілісності та безпеки даних.

Найважливішою інфраструктурною послугою є послуга оренди віртуальних серверів EC2. Абоненти мають у своєму розпорядженні віртуальні машини, що працюють на гіпервізорі Xen і 17 запатентованих варіантах KVM, вибір машин з різною обчислювальною потужністю, а також машини з доступом до спеціалізованого обладнання (графічні карти для GPGPU, програмовані комутаційні матриці). EC2 тісно інтегрований з іншими послугами хмарної інфраструктури, включаючи Elastic File System, який забезпечує файлову систему, підключену до віртуальних машин, Elastic Block Store (EBS), який підключає томи у вигляді блокових пристроїв до віртуальних машин, та S3, який забезпечує дисковий простір у великому масштабі.

Інші інфраструктурні послуги включають Route 53 (керований DNS у хмарі), VPC (спосіб створення ізольованої групи VPN-послуг у хмарі), Elastic Load Balancing (розподіл трафіку між віртуальними машинами), послугу Glacier для довгострокового зберігання даних («на холоді») та CloudFront, мережу розподілу контенту. Низка послуг дозволяє автоматизувати управління інфраструктурою, розміщеною в AWS, включаючи CloudFormation, OpsWorks і CloudWatch.

Хмара пропонує широкий спектр систем управління базами даних у хмарі різних категорій. Серед доступних систем NoSQL є Amazon SimpleDB, DynamoDB, резидентна система SGBD ElastiCache і графічна система SGBD

Neptune. В рамках послуги Amazon Relational Database Service (RDS) абоненти можуть впроваджувати бази даних у хмарі, що управляються популярними системами SGBD, такими як MySQL, Oracle Database, Microsoft SQL Server та PostgreSQL. Також доступна масштабована реляційна система SGBD Amazon Aurora, сумісна з MySQL та PostgreSQL. ParAccel, масово паралельна реляційна система SGBD, оптимізована для хмарної інфраструктури, доступна під брендом Amazon Redshift.

Служба Amazon Athena дозволяє аналізувати дані в Amazon S3 за допомогою стандартної мови SQL (з Presto) без необхідності мати виділену обчислювальну потужність (використовується безсерверна обчислювальна стратегія), а абоненти платять виключно за кількість мегабайтів, оброблених у виконаних запитах. Elastic MapReduce дозволяє абонентам створювати кластери Hadoop, оснащені відповідним екосистемою продуктів Big Data (включаючи Spark, Hive, HBase, Presto). QuickSight пропонує абонентам функції візуального аналізу даних, що зберігаються в сервісах AWS. Amazon Elasticsearch Service забезпечує доступ до хмари для стека пошукових двигунів Elasticsearch і Kibana. Служба Amazon Machine Learning надає абонентам доступ до інструментів машинного навчання.

Серед middleware-сервісів — Amazon Kinesis Message Broker (за функціональністю схожий на Apache Kafka), служба черг SQS та служба передачі повідомлень SNS.

Інструмент для розгортання додатків у моделі «Function as a Service» з використанням безсерверної стратегії: AWS Lambda; Elastic Kubernetes Service пропонує можливість розгортання додатків у контейнерній інфраструктурі, що управляється Kubernetes.

1.2.2 Google Cloud Platform

Google Cloud Platform, що пропонується компанією Google, — це набір хмарних сервісів, які працюють на тій самій інфраструктурі, що й продукти

Google для кінцевих користувачів, такі як Google Search, Gmail, Google Drive та YouTube. Окрім набору інструментів для управління, він пропонує широкий спектр модульних хмарних сервісів, включаючи обробку даних, зберігання даних, аналіз даних та машинне навчання.

Google Cloud Platform надає інфраструктуру як сервіс, платформу як сервіс та безсерверні обчислювальні середовища.

У квітні 2008 року Google анонсувала App Engine, платформу для створення та хостингу веб-додатків у керованих Google центрах обробки даних, яка стала першою хмарною службою компанії. Служба була відкрита для широкої публіки в листопаді 2011 року. З моменту анонсу App Engine Google додала до платформи кілька хмарних служб.

Google Cloud Platform є частиною Google Cloud, що включає інфраструктуру публічної хмари Google Cloud Platform, а також Google Workspace (G Suite), корпоративні версії операційних систем Android і Chrome, а також інтерфейси прикладного програмування (API) для комп'ютерів, навчальні послуги та картографічні послуги для підприємств.

Google Cloud Platform надає різноманітні сучасні послуги, що дозволяють компаніям та розробникам створювати, розгортати та підтримувати різноманітні типи додатків і сервісів. Серед них є такі платформи, як App Engine, що є платформою як послугою (PaaS), яка дозволяє легко розміщувати та керувати веб-додатками, забезпечуючи автоматичне масштабування та інтеграцію різних технологій розробки.

Для зберігання і роботи з великими обсягами даних на платформі доступна послуга BigQuery, що є потужною аналітичною системою, яка дозволяє швидко обробляти масштабні набори даних завдяки інфраструктурі, побудованій на основі хмарних технологій. Для роботи з неструктурованими та напівструктурованими даними існує сервіс BigTable — масштабована NoSQL-база даних, що дозволяє зберігати великі обсяги інформації з мінімальною затримкою та високою доступністю.

Для розробників, які мають обмежений досвід роботи з машинним

навчанням, Google пропонує Cloud AutoML. Це набір інструментів, що дає можливість створювати нейронні мережі та реалізовувати задачі штучного інтелекту без глибоких знань у цій сфері. Також для організації документів і простих структурованих даних у хмарі використовується Cloud Datastore, який являє собою високодоступну базу документів з підтримкою запитів та транзакцій.

Послуга Cloud Pub/Sub дозволяє додаткам взаємодіяти між собою за допомогою системи публікації та підписки, створюючи ефективні потоки даних без потреби прямого зв'язку між додатками. Це забезпечує стабільність та гнучкість в обміні інформацією між мікросервісами.

Інфраструктура Compute Engine надає віртуальні машини для розгортання додатків у хмарному середовищі, дозволяючи легко масштабувати обчислювальні ресурси відповідно до потреб. Kubernetes Engine надає повністю керовану систему для автоматизації розгортання, масштабування і управління контейнеризованими додатками, що істотно спрощує роботу з контейнерами та допомагає в реалізації стратегії DevOps.

Крім цього, Google Cloud Platform підтримує спеціалізовані послуги, такі як Google Genomics для аналізу геномних даних, Cloud Video Intelligence для аналізу та розпізнавання відео-контенту, Cloud Vision для роботи з візуальними даними та автоматичного розпізнавання зображень. Storage — ще одна важлива послуга, що забезпечує простий REST API-доступ до файлів і контенту в хмарних сховищах, підтримуючи високу швидкість передачі та надійність зберігання інформації.

1.2.3 Microsoft Azure

Раніше відома як Windows Azure, це загальнодоступна хмарна платформа від Microsoft. Вона пропонує низку хмарних послуг, зокрема IT-послуги, аналітичні послуги, послуги зберігання даних та мережеві послуги. Користувачі можуть обирати з-поміж багатьох послуг, щоб створювати та

розвивати нові додатки або запускати існуючі додатки в загальнодоступній хмарі.

Платформа Azure була розроблена, щоб допомогти компаніям вирішувати завдання та досягати організаційних цілей. Вона надає інструменти для всіх секторів, включаючи електронну комерцію, фінанси та різні підприємства зі списку Fortune 500, а також сумісна з технологіями з відкритим кодом. Це дає користувачам велику гнучкість у використанні обраних інструментів та технологій. Крім того, Azure пропонує чотири різні форми хмарної обробки: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) та Serverless Platform.

Microsoft Azure пропонує широкий спектр послуг, завдяки чому можливості його використання надзвичайно різноманітні. Одним з найпопулярніших застосувань Microsoft Azure є використання віртуальних машин або контейнерів у хмарі. Ці ресурси можуть розміщувати компоненти інфраструктури, такі як DNS-сервери (Domain Name System), служби Windows Server, такі як Internet Information Services (IIS), або додатки інших виробників. Microsoft також підтримує операційні системи інших виробників, такі як Linux.

Azure також часто використовується як платформа для хостингу баз даних у хмарі. Microsoft пропонує безсерверні реляційні бази даних, такі як Azure SQL, та нереляційні бази даних, такі як NoSQL.

Крім того, платформа часто використовується для створення резервних копій та відновлення даних після збою. Багато компаній використовують Azure Storage як архів для задоволення своїх потреб у довгостроковому зберіганні даних.

2 ЗАСОБИ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ

2.1 Docker Compose та Docker Swarm

У той час як Docker Standalone дозволяє запускати додатки або служби в окремих контейнерах, Docker пропонує два потужні інструменти, які оптимізують управління та координацію контейнерів.

Docker Compose, окремо встановлений плагін Docker, є інструментом для визначення та управління додатками з декількома контейнерами. Він використовує декларативний файл YAML для переліку служб та їх конфігурацій. За допомогою Docker Compose ви можете визначити взаємозв'язки та залежності між контейнерами та запустити весь стек додатків за допомогою однієї команди «docker-compose up».

Однією з основних функцій Docker Compose є визначення служб, кожна з яких виконується у власному контейнері, а образ, порти, змінні середовища та розділи файлової системи можна вказати окремо. Docker Compose автоматично створює стандартну мережу для зв'язку між контейнерами, а також дозволяє створювати власні мережі для ізоляції та контролю зв'язку. Інструмент пропонує контроль обсягу, що дозволяє створювати іменовані розділи або призначати локальні каталоги для зберігання даних. Docker Compose також підтримує налаштування середовища за допомогою змінних середовища, що спрощує налаштування контейнерів без зміни коду.

Docker Swarm — це рішення Docker для кластеризації та оркестрування. Воно дозволяє створювати та керувати кластером вузлів Docker і перетворювати їх на розподілений обчислювальний ресурс. За допомогою Swarm ви можете розгортати та масштабувати додатки на декількох машинах, забезпечуючи високу доступність та відмовостійкість.

Docker Swarm — це інтегрована в Docker технологія для кластеризації

та координації контейнерів, яка дозволяє об'єднати кілька хостів Docker в один кластер для централізованого управління та масштабування контейнеризованих додатків. Головною особливістю Swarm є режим Swarm, який перетворює групу серверів у кластер, що містить два типи вузлів: вузли управління, які відповідають за координацію, планування та управління кластером, та робочі вузли, на яких запускаються самі контейнери.

Swarm пропонує прості та ефективні механізми масштабування послуг. Користувач може вказати кількість реплік послуги, а Swarm автоматично розподіляє ці репліки між доступними вузлами, враховуючи ресурси та поточне навантаження. Це дозволяє гнучко реагувати на коливання навантаження та забезпечує високу доступність додатків[11].

Для балансування навантаження Swarm має вбудований розподільник запитів, який перенаправляє вхідний трафік до активних контейнерів послуг, рівномірно розподіляючи навантаження та покращуючи стабільність і продуктивність системи. Це забезпечує безперервність роботи додатків навіть у разі виходу з ладу окремих вузлів.

Swarm підтримує безперервне оновлення сервісів за допомогою механізму поетапного оновлення. Це означає, що контейнери оновлюються по одному, без повного переривання роботи сервісу. Це дозволяє уникнути простоїв і мінімізувати вплив на користувачів. Крім того, Swarm дозволяє налаштувати стратегії оновлення, щоб забезпечити плавний перехід на нові версії вашого додатка.

Важливою особливістю Docker Swarm є Swarm Config, механізм управління конфігураційними файлами, що використовуються службами в кластері. Ці файли надійно зберігаються в кластері і передаються тільки службам з відповідними правами доступу. Це підвищує безпеку і спрощує централізоване управління конфігурацією.

Таким чином, Docker Swarm поєднує просту конфігурацію, автоматичне масштабування, балансування навантаження і безперервні оновлення, щоб забезпечити потужний інструмент для розгортання і

управління контейнеризованими додатками в різних середовищах, від локальних серверів до

Docker Swarm спрощує управління та масштабування контейнерних додатків завдяки єдиному інтуїтивно зрозумілому інтерфейсу для управління кластерами вузлів Docker. Docker Compose та Docker Swarm допомагають оптимізувати весь процес розробки та розгортання, дозволяючи розробникам створювати складні багатоконтейнерні додатки та ефективно їх організовувати.

2.2 Підхід «Інфраструктура як код»

Інфраструктура як код (IaaC) — це підхід до управління інфраструктурою, в якому інфраструктурні ресурси (наприклад, сервери, мережі, бази даних) визначаються у вигляді коду з метою автоматизації їх надання, конфігурації та управління. Основними принципами підходу, заснованого на коді, є декларативність, яка полягає в описі бажаного стану системи, а не послідовності кроків, що ведуть до цього стану, так що фактичний стан автоматично приводиться у відповідність до опису. Версіювання та контроль версій гарантують зберігання коду інфраструктури в системах, подібних до Git, що дозволяє відстежувати зміни, повертатися до попередніх версій та ефективно працювати в команді.

Завдяки автоматизації інфраструктуру можна швидко та без помилок розгортати, налаштовувати та управляти нею, що підвищує її надійність та швидкість реагування. Відтворюваність гарантує, що код може бути використаний на різних платформах і в різних хмарних середовищах, що збільшує гнучкість у виборі середовища розгортання. Стабільність і надійність забезпечуються узгодженим кодом, який мінімізує людські помилки і підтримує узгоджене стан системи. Швидкість розгортання підвищується завдяки автоматизації створення і налаштування нових ресурсів.

Завдяки гнучкості інфраструктурного коду ресурси можна легко додавати або видаляти за потреби. Системи контролю версій забезпечують зберігання історії та журналів контролю, що дозволяє відстежувати всі зміни та гарантувати прозорість розгортання та конфігурації системи. «Інфраструктура як код» революціонізує сучасні процеси розробки та забезпечує швидше, надійніше та ефективніше впровадження та управління інфраструктурою. Таким чином, розробники мають більше часу, щоб зосередитися на бізнес-цілях та інноваціях, замість того, щоб витрачати час на ручне управління інфраструктурою.

Загалом, інструменти Infrastructure-as-Code можна класифікувати за такими категоріями:

- Інструменти управління конфігурацією: вони часто використовують імперативно-декларативний формат, а не декларативний, і зазвичай застосовуються для налаштування створених ресурсів або для управління ними після створення (Ansible);
- інструменти управління інфраструктурою – вони використовуються для безпосереднього створення, модифікації та видалення інфраструктури як платформи для різних видів діяльності (Terraform).

Для ефективної роботи з інфраструктурою, де можливі модульність, ідемпотентність та передбачуваність, важливо правильно розподілити обов'язки між різними інструментами.

2.2.1 HashiCorp Terraform

HashiCorp Terraform — це інструмент «інфраструктура як код», який дозволяє описувати ресурси в хмарі та на місці за допомогою коду в зрозумілих для людини файлах конфігурації, які можна змінювати, повторно використовувати та застосовувати для розгортання. Terraform забезпечує узгоджений робочий процес для представлення та управління всією інфраструктурою протягом усього її життєвого циклу. Terraform працює як з

компонентами високого, так і низького рівня.

Terraform може створювати ресурси в хмарних середовищах та інших службах і керувати ними за допомогою API (рисунок 2.1) .

Рисунок 2.1 – Взаємодія між Terraform та API

За словами постачальника, Terraform може працювати практично з усіма платформами або сервісами, що мають доступний API-інтерфейс.

Завдяки співпраці між HashiCorp і спільнотою Terraform з'явилася велика кількість постачальників, що управляють різними типами ресурсів і сервісів:

Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog і багато інших.

Основна робота Terraform складається з трьох етапів (рисунок 2.2). Спочатку розробник готує код і визначає ресурси, необхідні для різних хмарних середовищ. Потім відбувається планування, під час якого Terraform аналізує існуючу інфраструктуру та конфігурацію, щоб скласти план виконання, який вказує, які ресурси потрібно створити, оновити або видалити. На останньому етапі, після затвердження плану, Terraform вносить зміни в потрібному порядку, враховуючи взаємозалежності між ресурсами.

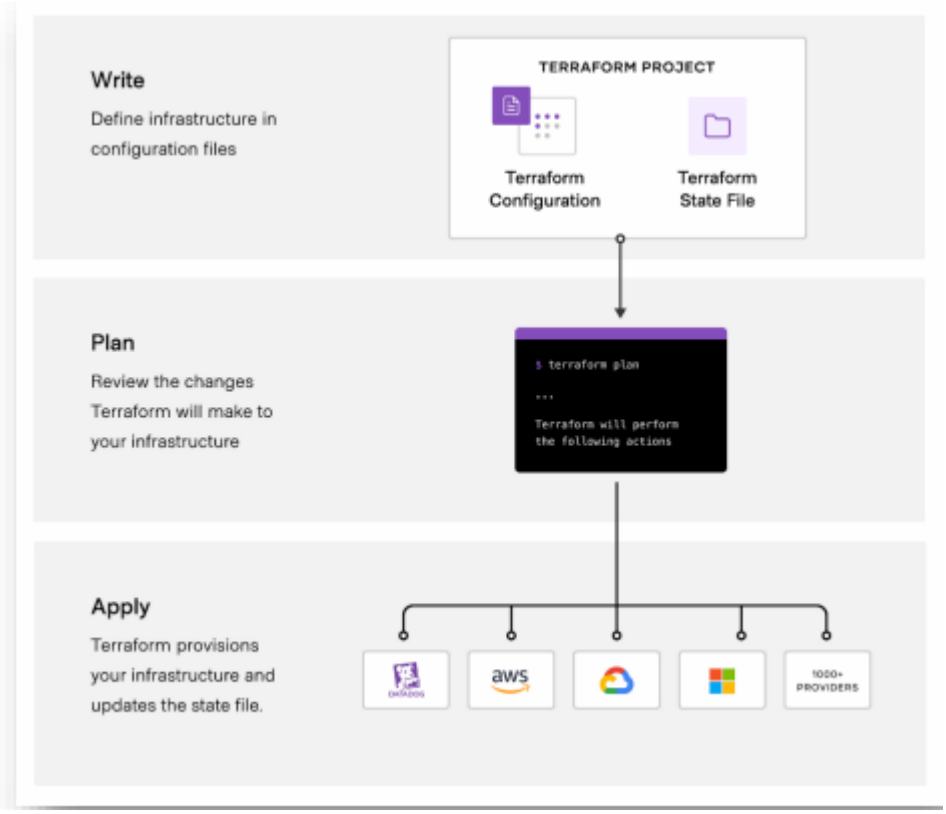


Рисунок 2.2 – Основні етапи роботи Terraform

Terraform State (TF State) — це файл або набір файлів, в яких зберігається інформація про стан ресурсів, що управляються Terraform. Цей стан є важливим елементом для Terraform і використовується для визначення поточного стану інфраструктури та виявлення змін, необхідних для досягнення бажаного стану.

Неправильно визначений стан Terraform призводить до проблем, що виникають внаслідок розбіжності між реальними ресурсами та їх «представленням» Terraform, що називається «відхиленням стану». Для вирішення цієї проблеми необхідні додаткові маніпуляції, такі як «terraform import».

Основні характеристики та функції стану Terraform такі: Стан Terraform — це репозиторій, що містить інформацію про всі керовані ресурси, їхній стан та конфігурацію. Ця інформація включає ідентифікатори ресурсів, значення їхніх атрибутів та взаємозв'язки між ресурсами у вигляді графа залежностей. Таким чином Terraform може відстежувати доступні

ресурси, їхню конфігурацію та взаємозв'язки.

Terraform State відповідає за моніторинг залежностей між ресурсами, що дозволяє точно визначити правильний порядок створення, оновлення або видалення ресурсів. Ця інформація є важливою для ефективного управління інфраструктурою, оскільки зміни повинні вноситися в правильному порядку з урахуванням залежностей.

На основі даних State Store Terraform планує зміни. Він порівнює поточний стан інфраструктури з цільовим станом, описаним у конфігураційному коді, і визначає зміни, які необхідно внести для досягнення цього цільового стану. Це дозволяє створювати детальні плани, які вказують, які ресурси будуть створені, змінені або видалені.

Безпека та конфіденційність є важливими елементами Terraform State. Оскільки репозиторій стану може містити конфіденційні дані, такі як паролі, ключі доступу та інші секрети, для захисту цієї інформації використовується шифрування або зовнішні системи управління секретами. Це дозволяє захистити конфіденційну інформацію від несанкціонованого доступу.

Terraform State можна зберігати локально на комп'ютері розробника або в віддалених сховищах, таких як Terraform Cloud, Amazon S3, Consul та інших. Завдяки використанню віддалених рішень кілька розробників можуть одночасно працювати над інфраструктурою, гарантуючи безпеку, цілісність та актуальність даних.

HashiCorp Configuration Language (HCL) — це мова конфігурації, яка використовується для опису інфраструктури в Terraform. HCL — це спрощена, відкрита та декларативна мова, яка дозволяє розробникам чітко та лаконічно визначати структуру, параметри та взаємозв'язки між ресурсами інфраструктури.

Основні характеристики мови конфігурації HashiCorp (HCL) такі:

HCL має просту та зручну синтаксис, що полегшує навчання та зменшує кількість помилок під час написання файлів конфігурації. Мова підтримує декларативний підхід, що дозволяє однозначно описувати

бажаний стан інфраструктурних ресурсів, а не описувати кроки, що ведуть до цього стану. Це значно спрощує створення та управління інфраструктурою.

HCL підтримує модульність, тобто можливість створення стандартних, багаторазових блоків конфігурації. Завдяки цьому окремі елементи інфраструктури можна швидко впровадити та підтримувати в належному стані.

Мова дозволяє використовувати вирази, змінні та функції, завдяки чому конфігурації є більш гнучкими та динамічними, а параметри можна адаптувати до різних умов та сценаріїв впровадження.

HCL можна інтегрувати з різними постачальниками хмарних послуг, такими як AWS, Azure та Google Cloud Platform, завдяки чому їхні ресурси та функції можна використовувати безпосередньо в конфігураційних файлах.

HCL також підтримує коментарі, що полегшує документування коду та покращує співпрацю між програмістами та адміністраторами інфраструктури.

2.2.2 Ansible

Ansible — це інструмент автоматизації, що дозволяє налаштовувати та керувати системами. Ansible використовує декларативний підхід і базується на мові YAML для визначення файлів конфігурації. Ansible пропонує функції розгортання та управління конфігурацією через Secure Shell (SSH) з'єднання.

Ansible є одним з найпопулярніших інструментів автоматизації завдяки кільком важливим перевагам. По-перше, він простий у використанні: має просту та інтуїтивно зрозумілу синтаксис, що дозволяє швидко навчитися автоматизувати конфігурацію та управління системами за допомогою декларативного підходу. По-друге, Ansible має безагентну архітектуру: він не вимагає встановлення додаткового програмного забезпечення на керованих комп'ютерах, оскільки спілкується з ними через SSH, що значно спрощує розгортання та управління.

Ще однією важливою особливістю є ідемпотентність: Ansible виконує тільки ті кроки, які необхідні для переведення системи в бажаний стан, уникаючи зайвих дій. Інструмент підтримує багато операційних систем і хмарних платформ, таких як Linux, Windows, AWS, Azure і GCP, що робить його універсальним для багатьох завдань.

Ansible дозволяє реалізувати принцип «інфраструктура як код», який дозволяє зберігати конфігурації в системах контролю версій, забезпечуючи повторюваність і автоматизацію розгортання інфраструктури. Крім того, велика та активна спільнота користувачів і розробників сприяє швидкому вирішенню проблем, розробці модулів і підтримці, що робить Ansible надійним і постійно вдосконалюваним інструментом.

Ключові поняття Ansible:

- Контрольний вузол хост, на якому виконуються скрипти Ansible;
- Керувані вузли цільові хости, з якими взаємодіє Ansible;
- Інвентар: конфігураційний файл, що описує цільові хости
- Plays контекст виконання Ansible;
- Playbooks – основна одиниця виконання скрипту Ansible, яка є файлом, що містить оголошені зв'язки між компонентами Ansible;
- Ролі – одиниці, що використовуються для імпорту та використання;
- Завдання – визначення дій, які повинні бути виконані на керованому вузлі.
- Контрольний вузо кожен вузол, на якому встановлено Ansible.
- Керуваний вузол – це мережевий пристрій (та/або сервер), який керується Ansible. Керуваний вузол іноді називають «хостом». Ansible не встановлюється на керованому вузлі. Зазвичай керуваний вузол повинен бути доступним з вузла керування через SSH і мати встановлений Python.
- Інвентар – це список керованих вузлів. Файл інвентаризації іноді називають «файлом хоста».

Файл інвентаризації може містити таку інформацію, як IP-адреса кожного керованого вузла, конфігурація SSH-з'єднань тощо. Файл

інвентаризації також може організовувати керовані вузли в групи для полегшення управління. Файли інвентаризації можуть мати різні формати, зокрема YAML та INI.

Виконання Ansible включає виконання команди «ansible-playbook», якій передається аргумент: ім'я файлу, в якому визначена послідовність завдань. Для виконання завдань необхідно правильно налаштувати інвентаризацію та мати доступ до цільових хостів через протокол SSH.

Для спрощення можна використовувати структуру каталогів Ansible-galaxy, яка дозволяє чітко організувати ролі Ansible.

Завдяки такому підходу автоматизацію Ansible можна розглядати як модулі коду, де ролі створюються та пов'язуються між собою в різних сценаріях для використання в різних середовищах.

Технології Ansible і Terraform не є взаємозамінними і можуть використовуватися разом. У багатьох випадках інфраструктура поділяється на області відповідальності і відповідно розподіляється між Ansible і Terraform (рисунок 2.3) : Terraform використовується для створення основи інфраструктури для розгортання, а Ansible — для автоматизації конфігурації та встановлення програмного забезпечення.

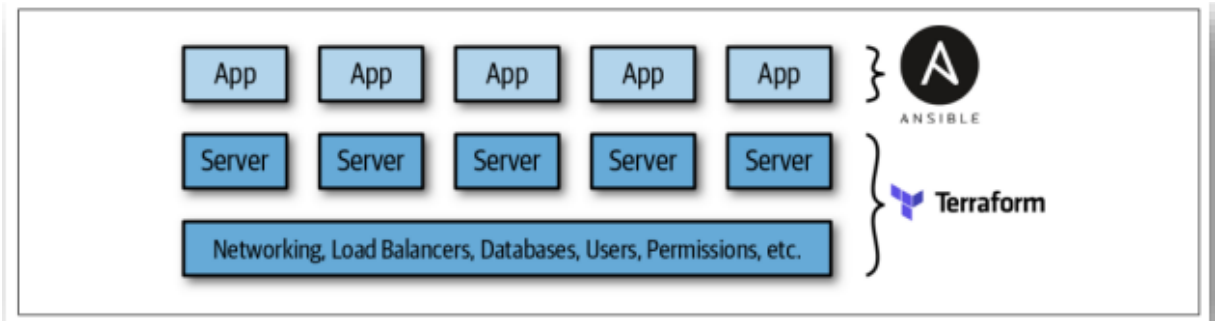


Рисунок 2.3 - Розподіл "зон відповідальності" Ansible та Terraform

Щоб забезпечити оптимальну взаємодію між двома технологіями, ви можете створювати динамічні інвентари Ansible на основі вихідних даних Terraform, що дозволяє автоматизувати послідовність завдань, що виконуються обома технологіями, та об'єднати їх у ланцюжок додатків.

2.3 Gitlab, Gitlab CI/CD

GitLab — це онлайн-платформа DevOps, яка пропонує повний набір інструментів для управління життєвим циклом програмного забезпечення. Заснована на популярній системі контролю версій Git, вона пропонує безліч функцій, зокрема управління вихідним кодом, відстеження проблем, безперервну інтеграцію та розгортання (CI/CD) та багато іншого.

Одним з найважливіших елементів GitLab є вбудовані функції CI/CD, відомі як GitLab CI/CD. CI/CD означає безперервну інтеграцію та безперервну доставку/розгортання, що є важливими практиками в сучасному розробленні програмного забезпечення.

Завдяки GitLab CI/CD розробники можуть автоматизувати процес створення та доставки своїх додатків у вигляді артефактів. GitLab CI/CD дозволяє розробникам регулярно інтегрувати зміни, внесені в код, у спільний репозиторій. За допомогою конвеєрів розробники можуть визначати завдання та робочі процеси, які автоматично створюють код після затвердження змін у репозиторії. Це дозволяє швидко виявляти проблеми інтеграції та сприяє співпраці між членами команди.

GitLab CI/CD виходить за межі безперервної інтеграції та пропонує інструменти для автоматичного розгортання додатків у різних середовищах. Завдяки налаштуванню конвеєра GitLab розробники можуть визначати етапи розгортання та вказувати умови передачі змін коду в різні середовища. Це дозволяє командам швидко та надійно публікувати оновлення програмного забезпечення.

GitLab CI/CD можна інтегрувати з інфраструктурними інструментами, такими як Terraform і Kubernetes, що дозволяє командам керувати інфраструктурою як кодом. Визначаючи конфігурації інфраструктури у файлах, що контролюються версіями, розробники можуть легко створювати або видаляти інфраструктурні ресурси в рамках своїх конвеєрів CI/CD. Це сприяє узгодженій та відтворюваній розгортанню додатків.

GitLab CI/CD пропонує надійну систему управління середовищем, яка дозволяє командам визначати та керувати різними середовищами для своїх додатків. Розробники можуть вказувати змінні, секретні дані та конфігурації, специфічні для кожного середовища, забезпечуючи тим самим узгоджене впровадження додатків на різних етапах.

GitLab CI/CD пропонує вбудовані механізми моніторингу та зворотного зв'язку, які дозволяють відстежувати стан і прогрес CI/CD-пайплайнів. Розробники можуть переглядати детальні журнали, звіти про тестування та показники продуктивності, щоб виявляти проблеми та виправляти помилки. Це дозволяє командам гарантувати якість і надійність випущених версій програмного забезпечення.

GitLab CI/CD легко інтегрується з іншими інструментами та сервісами DevOps, такими як Docker, Terraform, AWS, Kubernetes та багатьма іншими. Він пропонує API, що дозволяє командам розширювати його функціональність та інтегрувати його в існуючі робочі процеси та інструменти.

Для виконання завдань у конвеєрі GitLab CI/CD використовує одиниці, які називаються Gitlab Runners (рисунок 2.4). Це агенти GitLab, які відповідають за виконання завдань, що їм призначені, тобто вони діють як середовища виконання. Gitlab дозволяє використовувати загальнодоступний агент або створити власного агента з особистими ресурсами для виконання завдань. Існує кілька типів агентів, але найпоширенішими є:

- - агент «shell», який використовує середовище хоста, на якому він зареєстрований;

- - агент «Docker», який використовує середовище конкретного образу Docker для кожного етапу конвеєра.

За замовчуванням для опису конвеєрів використовується файл з назвою «.gitlab-ci.yml». Файли конвеєра використовують синтаксис YAML і дозволяють описувати виконання послідовних завдань за умови, що доступний агент GitLab. Якщо файл «.gitlab-ci.yml» знаходиться в репозиторії

Git, GitLab зчитує інструкції, що містяться в ньому, і запускає процес планування виконання на доступному агенті. GitLab може використовувати кілька агентів одночасно. Щоб вказати виконання конвеєра на конкретному агенті, використовується спеціальний тег (тег у конфігурації агента і тег у конвеєрі), який дозволяє керувати плануванням завдань для конкретних агентів. Синтаксис «.gitlab-ci.yml» відрізняється для різних агентів.

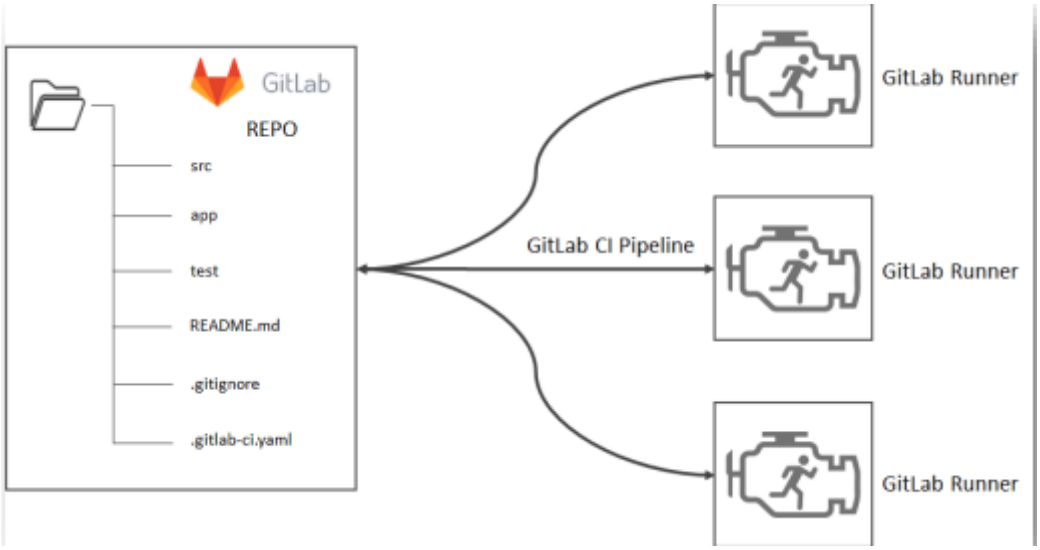


Рисунок 2.4 – Візуалізація розподілу задач між декількома агентами GitLab

3 РОЗГОРТАННЯ КЛАСТЕРА KUBERNETES

Ця частина роботи демонструє повноцінну реалізацію концепції Infrastructure as Code. Було реалізовано автоматизоване розгортання Kubernetes-кластера на хмарній платформі AWS із використанням двох ключових інструментів — Terraform та Ansible. Усі етапи реалізовані з урахуванням безперервної інтеграції, масштабованості та повторюваності.

Кроки для налаштування кластера Kubernetes на AWS за допомогою двох популярних інструментів DevOps: Terraform та Ansible.

3.1 Основа проекту

У цьому проєкті у нас буде дві основні папки (рисунок 3.1):

- Папка Terraform відповідатиме за налаштування інфраструктури з правильною конфігурацією.

- Папка Ansible міститиме плейбуки для встановлення попередніх компонентів та запуску команд kubeadm.

Рисунок 3.1 – Структура проєкту

3.2 Terraform

3.2.1Налаштування ролей AWS IAM

На цьому етапі відбувається створення ролей IAM, які EC2-інстанси

використовуватимуть для взаємодії з AWS-сервісами. Це важливо для безпечного доступу до API без використання статичних ключів. Через main.tf описується роль, політика та профіль, що прикріплюється до EC2.

3.2.2 Налаштування Terraform-проєкту

Файл providers.tf відповідає за підключення до відповідного хмарного провайдера. У цьому випадку використовується AWS-провайдер, офіційно підтримуваний компанією HashiCorp.

Перший блок описує, що для роботи необхідно завантажити провайдер aws із репозиторію hashicorp. Це дозволяє Terraform використовувати ресурси AWS (EC2, VPC, IAM тощо).

Наступний блок (рисунок 3.2) — це безпосереднє визначення конфігурації провайдера:

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
  }
}

provider "aws" {
    profile = var.profile
    region = var.region
}
```

Рисунок 3.2 – Вміст providers.tf

3.2.3 Security Group

Забезпечення безпеки є однією з ключових вимог до хмарної інфраструктури, особливо у випадку відкритого підключення до керованих сервісів, як-от API Kubernetes. У цьому контексті, Security Group (SG)

виконує роль віртуального брандмауера, який контролює вхідний (ingress) і вихідний (egress) мережевий трафік EC2-інстансів.

Модуль security_group (рисунок 3.3) в рамках даного проєкту дозволяє чітко задати правила доступу до вузлів кластера та гарантує, що лише дозволені протоколи та порти будуть відкриті, тим самим зменшуючи площу атаки.

```
resource "aws_security_group" "allow_tls_http_ssh" {
  name        = "k8s-sg-blog"
  description = "Allow all inbound traffic"
  vpc_id      = data.aws_vpc.default_vpc.id

  ingress = [
    {
      description      = "TLS"
      from_port        = 443
      to_port          = 443
      protocol         = "tcp"
      cidr_blocks      = var.allowed_cidr
      ipv6_cidr_blocks = [ "::/0" ]
      self = false
      security_groups = []
      prefix_list_ids = []
    },
    {
      description      = "TLS"
      from_port        = 6443
      to_port          = 6443
      protocol         = "tcp"
      cidr_blocks      = var.allowed_cidr
      ipv6_cidr_blocks = [ "::/0" ]
      self = false
      security_groups = []
      prefix_list_ids = []
    },
    {
      description      = "HTTP"
      from_port        = 80
      to_port          = 80
      protocol         = "tcp"
      cidr_blocks      = var.allowed_cidr
      ipv6_cidr_blocks = [ "::/0" ]
      self = false
      security_groups = []
      prefix_list_ids = []
    },
    {
      description      = "SSH"
      from_port        = 22
      to_port          = 22
      protocol         = "tcp"
      cidr_blocks      = var.allowed_cidr
      ipv6_cidr_blocks = [ "::/0" ]
      self = false
      security_groups = []
      prefix_list_ids = []
    }
  ]
```

Рисунок 3.3 – Вміст main.tf модуля Security Group

```
> cat modules/security_group/outputs.tf
output "sg_id" {
    value = aws_security_group.allow_tls_http_ssh.id
}%
```

Рисунок 3.4 – Вміст outputs.tf модуля Security Group

```
variable "allowed_cidr" {
  type = list(string)
  default = [ "0.0.0.0/0" ]
}

data "aws_vpc" "default_vpc"{
  default = true
}%
```

Рисунок 3.5 – variables.tf модуля Security Group

3.2.4 Пара ключів SSH

Однією з базових вимог для безпечного адміністрування EC2-інстансів у кластері Kubernetes є наявність механізму віддаленого доступу через протокол SSH. Щоб реалізувати це на практиці, AWS використовує пару ключів: приватний ключ зберігається у користувача, а публічний ключ імпортується до облікового запису AWS і автоматично розгортається на інстансах.

Модуль ssh_key (рисунок 3.6) автоматизує створення об’єкта AWS Key Pair та підключає публічний ключ до створених EC2-інстансів. Це дає змогу Terraform і Ansible отримати доступ до віртуальних машин у кластері без використання паролів або менш безпечних методів аутентифікації.

```
resource "aws_key_pair" "k8s_cluster_key" {
    key_name = var.key_pair_name
    public_key = "${file(var.public_key_path)}"
    tags = {
        Name        = "k8s-key"
        Environment = "test"
        Terraform   = "true"
    }
}%
```

Рисунок 3.6 – Вміст main.tf модуля SSH Key Pair

```
variable "key_pair_name" {
    type = string
}
variable "public_key_path" {
    type = string
}%
```

Рисунок 3.7 – Вміст variables.tf модуля SSH Key Pair

3.2.5 Контролер K8s

Контролер є центральним компонентом кластера Kubernetes, який відповідає за управління його станом. Саме на цьому вузлі запускаються компоненти: kube-apiserver, etcd, controller-manager, scheduler, які формують основу управління всією кластерною інфраструктурою.

У межах даного проєкту модуль controller створює окремий EC2-інстанс, який буде використано як master-вузол у кластері Kubernetes.

```
resource "aws_instance" "controller" {
  ami           = var.default_ami_id
  instance_type = var.instance_type
  key_name = var.key_pair_name
  security_groups = [ var.security_group_id ]
  subnet_id = data.aws_subnets.default_subnets.ids[0]

  associate_public_ip_address = true

  tags = {
    terraform = "true"
    project = "kube-auto"
  }

  root_block_device {
    delete_on_termination = true
    volume_size = 8
  }
}%
```

Рисунок 3.8 – Вміст main.tf модуля Controller

```
data "aws_vpc" "default_vpc"{
  default = true
}

data "aws_subnets" "default_subnets" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default_vpc.id]
  }
}
variable default_ami_id {
  type = string
}

variable "security_group_id" {
  type = string
}

variable "key_pair_name" {
  type = string
}

variable "instance_type" {
    default = "t2.medium"
}%
```

Рисунок 3.9 – Вміст variables.tf модуля Controller

У variables.tf (рисунок 3.9) ми вказуємо наші змінні, дані та вхідні дані

Потім ми передаємо створеному публічному IP-адресу контролера в outputs.tf (рисунок 3.10)

```
output "controller_public_ip" {
    value = aws_instance.controller.public_ip
}%
```

Рисунок 3.10 – Вміст outputs.tf модуля Controller

3.2.6 Працівники K8s

Модуль worker є важливою складовою інфраструктурного шаблону, оскільки відповідає за автоматизоване створення EC2-інстансів, які виступатимуть worker-нодами у кластері Kubernetes. На відміну від контролера, worker-ноди не керують кластером, а виконують навантаження: обробляють поди, виконують завдання та масштабуються відповідно до потреб. Саме цей модуль (рисунок 3.11) реалізує горизонтальне масштабування на рівні інфраструктури.

```
resource "aws_instance" "workers" {
  ami             = var.default_ami_id
  count           = var.worker_count
  instance_type   = var.instance_type
  key_name        = var.key_pair_name
  security_groups = [ var.security_group_id ]
  subnet_id       = data.aws_subnets.default_subnets.ids[0]

  associate_public_ip_address = true

  tags = {
    Name = "worker_${count.index}"
    terraform = "true"
    project = "kube-auto"
  }

  root_block_device {
    delete_on_termination = true
    volume_size = 8
  }
}%
```

Рисунок 3.11 – Вміст main.tf модуля workers

Блок (рисунок 3.12) дозволяє отримати список публічних IP-адрес worker-нод після їх створення:

Це критично важливо для генерації inventory.ini в Ansible, а також для ручного моніторингу або подальшої інтеграції з CI/CD.

```
output "workers_public_ips" {
    value = aws_instance.workers.*.public_ip
}%
```

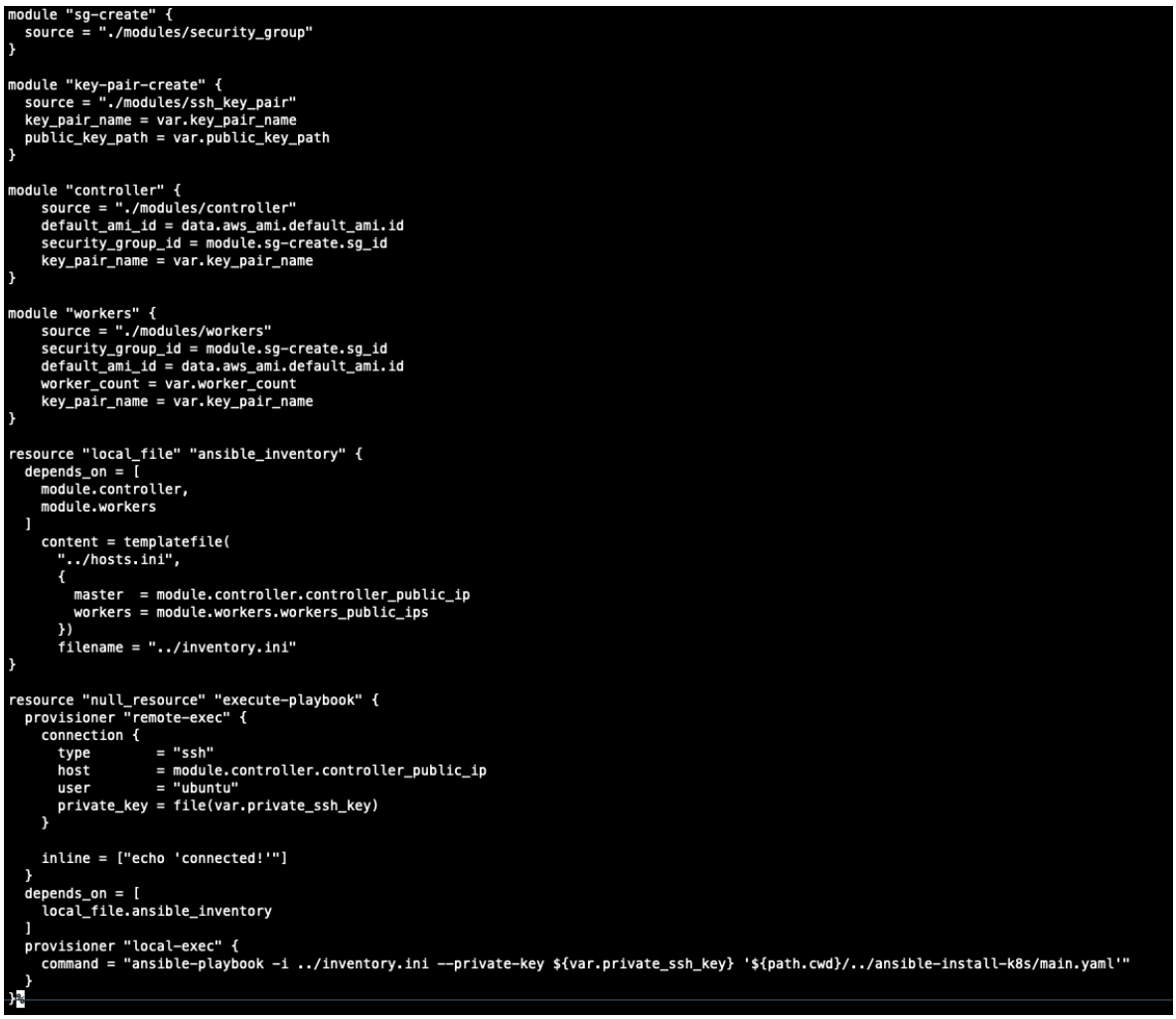
Рисунок 3.12 – Вміст outputs.tf модуля Security Group

```
data "aws_vpc" "default_vpc"{
  default = true
}

data "aws_subnets" "default_subnets" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default_vpc.id]
  }
}

variable "default_ami_id" {
  type = string
}

variable "worker_count" {
  type = string
}

variable "security_group_id" {
  type = string
}

variable "key_pair_name" {
  type = string
}

variable "instance_type" {
    default = "t2.medium"
}
```

Рисунок 3.13 – variables.tf модуля workers

3.2.7 Основна конфігурація

Ключовим елементом інфраструктурної автоматизації є інтеграція окремих функціональних блоків у єдиний сценарій розгортання. У цьому проєкті файл main.tf координує всі попередньо створені модулі (security_group, ssh_key, controller, workers), а також додає два ресурси: генерацію Ansible-інвентаря та запуск Ansible playbook після завершення побудови інфраструктури.



```
module "sg-create" {
  source = "./modules/security_group"
}

module "key-pair-create" {
  source = "./modules/ssh_key_pair"
  key_pair_name = var.key_pair_name
  public_key_path = var.public_key_path
}

module "controller" {
    source = "./modules/controller"
    default_ami_id = data.aws_ami.default_ami.id
    security_group_id = module.sg-create.sg_id
    key_pair_name = var.key_pair_name
}

module "workers" {
    source = "./modules/workers"
    security_group_id = module.sg-create.sg_id
    default_ami_id = data.aws_ami.default_ami.id
    worker_count = var.worker_count
    key_pair_name = var.key_pair_name
}

resource "local_file" "ansible_inventory" {
  depends_on = [
    module.controller,
    module.workers
  ]
    content = templatefile(
      "../hosts.ini",
      {
        master  = module.controller.controller_public_ip
        workers = module.workers.workers_public_ips
      })
      filename = "../inventory.ini"
}

resource "null_resource" "execute-playbook" {
  provisioner "remote-exec" {
    connection {
      type        = "ssh"
      host        = module.controller.controller_public_ip
      user        = "ubuntu"
      private_key = file(var.private_ssh_key)
    }

    inline = ["echo 'connected!'"]
  }
  depends_on = [
    local_file.ansible_inventory
  ]
  provisioner "local-exec" {
    command = "ansible-playbook -i ../inventory.ini --private-key ${var.private_ssh_key} '${path.cwd}/../ansible-install-k8s/main.yaml'"
  }
}
```

Рисунок 3.14 – Вміст main.tf

Цей блок (рисунок 3.14) об’єднує всі модулі — IAM, мережу, EC2, ключі, групи безпеки в єдину логіку через файл main.tf на верхньому рівні. Визначаються змінні (рисунок 3.15),передаються значення між модулями, а вихідні параметри зберігаються для подальшого використання (наприклад, IP-адреси в Ansible).

```
data "aws_vpc" "default_vpc"{
  default = true
}

data "aws_subnets" "default_subnets" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default_vpc.id]
  }
}

variable "default_ami_id" {
  type = string
}

variable "worker_count" {
  type = string
}

variable "security_group_id" {
  type = string
}

variable "key_pair_name" {
  type = string
}

variable "instance_type" {
    default = "t2.medium"
}
```

Рисунок 3.15 – Вміст variables.tf

3.3 Ansible

Після створення інфраструктури, наступним кроком є конфігурація ОС, налаштування Kubernetes і запуск кластеру через Ansible. Для цього створені роли, що поділяють завдання на логічні блоки. Вони зберігаються в ansible-install-k8s/roles.

3.3.1 Role kube-prerequisites

Ця роль (рисунок 3.16) реалізує початкову підготовку системи перед встановленням Kubernetes і контейнерного рушія. Вона виконується як на master-, так і на worker-вузлах. Роль орієнтована на забезпечення сумісності ОС з вимогами Kubernetes.

```
---
############## Install and configure prerequisites  ##############
- name: Create the file for Forwarding IPv4 and iptables modules
  become: true
  file:
    path: /etc/modules-load.d/k8s.conf
    state: touch

- name: configure Forwarding IPv4 and letting iptables see bridged traffic
  become: true
  blockinfile:
    path: /etc/modules-load.d/k8s.conf
    block: |
      overlay
      br_netfilter

- name: modprobe overlay
  become: true
  community.general.modprobe:
      name: overlay
      state: present

- name: Verify that the overlay modules are loaded
  shell: lsmod | grep overlay
  register: overlay
- debug: msg={{overlay.stdout_lines}}

- name: modprobe br_netfilter
  become: true
  community.general.modprobe:
      name: br_netfilter
      state: present

- name: Verify that the br_netfilter modules are loaded
  shell: lsmod | grep br_netfilter
  register: br_netfilter
- debug: msg={{br_netfilter.stdout_lines}}

- name:  sysctl params required by setup, params persist across reboots
  become: true
  sysctl:
    name: "{{ item }}"
    value: '1'
    state: present
    reload: true
  loop:
    - net.ipv4.ip_forward
    - net.bridge.bridge-nf-call-ip6tables
    - net.bridge.bridge-nf-call-iptables

- name: Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, net.ipv4.ip_forward system variables are set to 1
  shell: |
    sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
  register: bridge_net
- debug: msg={{bridge_net.stdout_lines}}

- name: install pip3
  become: true
  apt:
    name: python3-pip
    state: present

- name: install pre-requisites for python
  pip:
    name:
      - openshift
      - pyyaml
      - kubernetes
```

Рисунок 3.16 – Завдання ролі kube-prerequisites

3.3.2 Role containerd

```
- name: Update repositories cache and install "foo" package
  apt:
    pkg:
    - ca-certificates
    - curl
    - gnupg
    - lsb-release
    - apt-transport-https
    state: latest
    update_cache: yes

- name: create keyrings directory
  file:
    path: /etc/apt/keyrings
    state: directory
    mode: 0755

- name: Add Docker GPG key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg

- name: Add Docker APT repository
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/{{ ansible_system | lower }}/{{ ansible_distribution | lower }} {{ ansible_distribution_release }} stable

- name: install containerd
  apt:
    name: containerd.io
    state: present
    update_cache: yes

- name: Apply the default configuration for the containerd
  shell: |
    sudo containerd config default | sudo tee /etc/containerd/config.toml

############## Configuring the systemd cgroup driver ##############
- name: replace line
  lineinfile:
    path: /etc/containerd/config.toml
    regexp: 'SystemdCgroup = false'
    line: '            SystemdCgroup = true'
    backrefs: yes

- name: Restaring Containerd
  systemd:
    state: restarted
    name: containerd
    enabled: yes
    daemon-reload: yes
```

Рисунок 3.17 – Завдання ролі containerd

Ця роль (рисунок 3.17) відповідає за встановлення та налаштування контейнерного рушія containerd, який є рекомендованим за замовчуванням для Kubernetes починаючи з версії 1.20+. Контейнерний рушій виконує роль середовища виконання контейнерів (Container Runtime Interface — CRI), яке забезпечує роботу подів у кластері.

3.3.3 Role kubelet, kubeadm and kubectl

Ця роль (рисунок 3.18) встановлює три основні компоненти Kubernetes на всіх вузлах: kubelet, kubeadm, kubectl. Вона також відповідає за блокування оновлень пакетів, щоб уникнути неконтрольованого переходу на нові версії.

```
- name: add Kubernetes apt-key
  apt_key:
    url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
    state: present

- name: add Kubernetes' APT repository
  apt_repository:
    repo: deb https://apt.kubernetes.io/ kubernetes-xenial main
    state: present
    filename: 'kubernetes'

- name: install Kubelet
  apt:
    name: kubelet=1.28.*
    state: present
    update_cache: true

- name: install Kubeadm
  apt:
    name: kubeadm=1.28.*
    state: present

- name: install kubectl
  apt:
    name: kubectl=1.28.*
    state: present

- name: hold kubeadm kubeadm kubectl
  dpkg_selections:
    name: "{{ item }}"
    selection: hold
  loop:
    - kubelet
    - kubeadm
    - kubectl
```

Рисунок 3.18 – Завдання ролі kubelet, kubeadm and kubectl

3.3.4 Role init-cluster

Ця роль (рисунок 3.19) виконується виключно на master-вузлі і є відповідальною за ініціалізацію кластера за допомогою kubeadm. Вона запускає команду kubeadm init, створює адміністративний конфігураційний файл admin.conf, а також забезпечує подальше підключення kubectl для керування кластером.

```
> cat roles/init-cluster/tasks/main.yml
---
# tasks file for init-cluster
    ############## Create cluster with kubeadm - DONE ##############
    - name: reset kubernetes component
      shell: kubeadm reset --force
      register: reset_cluster

    - name: Create cluster with kubeadm
      # TODO - change the IP dynamically
      shell: sudo kubeadm init --kubernetes-version=1.28.0 --pod-network-cidr=192.168.0.0/16
      register: init_cluster

    - name: create .kube directory
      file:
        path: "$HOME/.kube/"
        state: directory
        mode: 0755
      # TODO - Change this dynamically
    - name: Copy admin.conf to Home directory
      copy:
          src: "/etc/kubernetes/admin.conf"
          dest: "$HOME/.kube/config"
          owner: "{{ ansible_user | default(ansible_user_id) }}"
          group: "{{ ansible_user | default(ansible_user_id) }}"
          mode: 0755
          remote_src: true

    ############## download and apply the calico network add-on ###############
    - name: Download calico manifest to the cluster.
      ansible.builtin.get_url:
        url: https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
        dest: ~/calico.yaml
        mode: '0664'

    - name: Apply calico manifest to the cluster.
      kubernetes.core.k8s:
        state: present
        src: ~/calico.yaml

    ############## Create the workers join ###############
    - name: create join token
      shell: sudo kubeadm token create --print-join-command
      register: join_token

    - name: save join token
      shell: " echo {{ join_token.stdout }} > join.sh"

    - name: save to local
      run_once: true
      fetch: src=join.sh dest="{{ playbook_dir }}/buffer/" flat=yes

    - name: create .kube repo
      file:
        path: .kube
        state: directory

    - name: Copy admin.conf to Home directory
      copy:
        src: "/etc/kubernetes/admin.conf"
        dest: .kube/config
        remote_src: true
```

Рисунок 3.19 – Завдання ролі init-cluster

ВИСНОВКИ

В ході виконаня кваліфікаційної роботи було досліджено процес автоматизації інфраструктури за допомогою Terraform та Ansible. Дослідження підтвердило, що впровадження інструментів автоматизації значно підвищує ефективність управління IT-інфраструктурою. Terraform дозволяє швидко та надійно створювати ресурси в хмарі або віртуальні ресурси за допомогою декларативних описів, що значно зменшує ризик помилок і гарантує повторюваність процесів. Ansible, у свою чергу, пропонує простий і масштабований підхід до конфігурації системи, автоматизації рутинних завдань і забезпечення узгодженості конфігурації.

Поєднання цих двох інструментів дозволило повністю автоматизувати розгортання: від створення компонентів інфраструктури до їх конфігурації та введення в експлуатацію. Такий підхід скорочує час розгортання, підвищує надійність і контроль змін, а також забезпечує легку масштабованість рішень. Досвід, отриманий під час реалізації проекту, підтвердив корисність Terraform і Ansible в сучасних практиках DevOps і продемонстрував потенціал цих інструментів у подальшому розвитку інфраструктурних рішень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cloud Computing Platform Software - OpenStack. Openstack.org. URL: https://www.openstack.org/software/project-navigator/openstackcomponents (дата звернення: 10.05.2025).

2. Кононюк А. Е. Фундаментальна теорія хмарних технологій. Київ : Освіта України, 2018. 620 с.

3. Муковоз М. С., Сітніков В. І. Лідарні технології у системах автоматизованого керування. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : Тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.1: секція 5, Баку: ІСУ АР, Харків: ХНУРЕ, Харків: НТУ «ХПІ», Харків: НАУ «ХАІ». Харків : Жиліна: УмЖ, 2025. С. 142. URL: https://doi.org/10.32620/ICT.25.t1.

4. OpenStack Cloud Computing Cookbook – Fourth Edition. Packt, 2018. 416 с. ISBN 9781788398763.

5. Identity, Authentication, and Access Management in OpenStack. O’Reilly Media, Inc., 2015. 306 с. ISBN 9781491941201.

6. University of Cambridge. Computing service: OpenStack Nova. Research Computing Services HPC Documentation. URL: https://docs.hpc.cam.ac.uk/cloud/userguide/03-nova.html (дата звернення: 10.05.2025).

7. University of Cambridge. Networking service: OpenStack Neutron. RCC Documentation. Research Computing Services HPC Documentation. URL: https://docs.hpc.cam.ac.uk/cloud/userguide/02-neutron.html (дата звернення: 10.05.2025).

8. University of Cambridge. Storage services: OpenStack Cinder and Swift. Research Computing Services HPC Documentation. URL: https://docs.hpc.cam.ac.uk/cloud/userguide/06-cinder_swift.html (дата звернення: 10.05.2025).

9. Brikman Y. Terraform: Up and Running: Writing Infrastructure as Code. 3rd ed. O'Reilly Media, 2017. 206 с. ISBN 978-1491977088.

10. Smith K. How to Use Terraform and Ansible to Automate Infrastructures and Deployments. Edge Delta blog, 05.02.2025. URL: https://edgedelta.com/company/blog/how-to-use-terraform-and-ansible-to-automate-infrastures (дата звернення: 15.05.2025).

11. Geerling J. Ansible for Kubernetes by Example: Automate Your Kubernetes Infrastructure. Leanpub, 2023. 150 с. ISBN 978-1484292846.

12. Nicora L. Kubernetes from scratch to AWS with Terraform and Ansible (Part 1). OpenCredo blog, 26.08.2016. URL: https://opencredo.com/blog/kubernetes-aws-terraform-ansible-1 (дата звернення: 15.05.2025).

13. Konala P. R. R., Kumar V., Bainbridge D., Haseeb J. A Framework for Measuring the Quality of Infrastructure-as-Code Scripts. arXiv, 05.02.2025. URL: https://arxiv.org/abs/2502.03127 (дата звернення: 5.05.2025).