ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «СИСТЕМА АВТОМАТИЗОВАНОГО КЕРУВАННЯ ІНФРАСТРУКТУРОЮ ДЛЯ АУТСОРСИНГОВИХ ДОДАТКІВ У ХМАРНОМУ СЕРЕДОВИЩІ»

на здобуття освітнього ступеня магістра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Максим ФЕДОРЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КСДМ-62
Максим ФЕДОРЕНКО
Ім'я, ПРІЗВИЩЕ
Керівник: Віталій ВОЛОХІН
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти Магістр

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Федоренко Максим Андрійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система автоматизованого керування інфраструктурою для аутсорсингових додатків у хмарному середовищі»
керівник кваліфікаційної роботи Віталій ВОЛОХІН, канд.техн.наук, доцент,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320,
2. Строк подання кваліфікаційної роботи «29» грудня 2024р.
3. Вихідні дані до кваліфікаційної роботи: Інфраструктура як код є інноваційним підходом до управління ресурсами в хмарних середовищах, що дозволяє автоматизувати процеси розгортання, налаштування та масштабування інфраструктури. Для компаній, які займаються аутсорсингом, такі рішення є важливими, оскільки вони сприяють зменшенню людського фактору, підвищенню ефективності роботи та забезпеченню високої доступності додатків. Серед доступних інструментів для впровадження IaC особливе місце займає Terraform, який дозволяє описувати, створювати та керувати інфраструктурою у хмарних середовищах, таких як Amazon Web Services (AWS). Це забезпечує не лише автоматизацію процесів, але й дозволяє компаніям швидко адаптувати інфраструктуру до змін у вимогах клієнтів. Поєднання Terraform з CI/CD інструментами, такими як GitHub Actions чи Jenkins, забезпечує автоматизацію повного життєвого циклу інфраструктури, від її розробки до тестування та розгортання.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи та сучасні підходи до проектування хмарної інфраструктури
2. Практична реалізація інфраструктури для підтримки аутсорсингових додатків
3. Техніко економічне обґрунтування
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «01» вересня 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	10.10-15.10.2024	Виконано
2	Аналіз існуючих методів віртуалізації	15.10-25.10.2024	Виконано
3	Розрахунок витрат апаратних ресурсів для організації віртуалізації	01.11-10.11.2024	Виконано
4	Обробка матеріалу	11.11-18.11.2024	Виконано
5	Висновки за результатами аналізу	19.11-24.11.2024	Виконано
6	Розробка теоретичної частини	25.11-01.12.2024	Виконано
7	Розробка презентації	02.12-07.12.2024	Виконано
8	Передзахист	09.12-11.12.2024	Виконано
9	Здача в деканат	20.12-29.12.2024	Виконано

Здобувач вищої освіти _____ Максим ФЕДОРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Віталій ВОЛОХІН
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Мета роботи – розробити та впровадити автоматизовану систему керування інфраструктурою для аутсорсингових додатків у хмарному середовищі, що використовує інструмент Terraform та сервіси AWS для забезпечення надійного, масштабованого та ефективного управління ресурсами.

Об'єкт дослідження – процес розробки та реалізації інфраструктури для аутсорсингових додатків у хмарному середовищі.

Предмет дослідження – методи та технології автоматизації інфраструктури на основі хмарних сервісів AWS із застосуванням Terraform для управління, розгортання та масштабування ресурсів.

Короткий зміст роботи: У роботі розглянуто процес розробки та впровадження автоматизованої системи керування інфраструктурою для аутсорсингових додатків у хмарному середовищі. Основною технологією є використання Infrastructure as Code (IaC) на базі Terraform, що дозволяє спростити розгортання та управління інфраструктурою.

Розроблена система забезпечує автоматизоване керування інфраструктурою, спрощує процес впровадження змін та забезпечує гнучке масштабування ресурсів відповідно до вимог додатків. Це дозволяє ефективно підтримувати роботу аутсорсингових додатків у різних умовах середовища та оптимізувати витрати на інфраструктуру.

КЛЮЧОВІ СЛОВА: СИСТЕМА АВТОМАТИЗОВАНОГО КЕРУВАННЯ, INFRASTRUCTURE AS CODE, AWS, TERRAFORM, ХМАРНА ІНФРАСТРУКТУРА, АУТСОРСИНГОВІ ДОДАТКИ, АВТОМАТИЗАЦІЯ, МАСШТАБУВАННЯ, МОНІТОРИНГ, CI/CD.

ABSTRACT

The purpose of the study is to develop and implement an automated infrastructure management system for outsourced applications in the cloud environment that uses the Terraform tool and AWS services to ensure reliable, scalable, and efficient resource management.

Object of research - the process of developing and implementing infrastructure for outsourcing applications in the cloud environment.

Subject of research - methods and technologies for infrastructure automation based on AWS cloud services using Terraform for managing, deploying and scaling resources.

Summary of work: The paper describes the process of developing and implementing an automated infrastructure management system for outsourced applications in the cloud environment. The main technology is the use of Infrastructure as Code (IaC) based on Terraform, which simplifies the deployment and management of infrastructure.

The developed system provides automated infrastructure management, simplifies the process of implementing changes, and ensures flexible scaling of resources in accordance with application requirements. This makes it possible to effectively support the operation of outsourced applications in different environmental conditions and optimize infrastructure costs.

KEYWORDS: AUTOMATED MANAGEMENT SYSTEM, INFRASTRUCTURE AS CODE, AWS, TERRAFORM, CLOUD INFRASTRUCTURE, OUTSOURCED APPLICATIONS, AUTOMATION, SCALING, MONITORING, CI/CD.

ЗМІСТ

ВСТУП

У сучасному світі швидкого розвитку інформаційних технологій та хмарних рішень, компанії, що займаються аутсорсингом програмних продуктів, стикаються з численними викликами щодо ефективного управління інфраструктурою. Висока конкуренція на ринку змушує компанії шукати рішення, що дозволять автоматизувати процеси, зменшити витрати та забезпечити безперервність надання послуг. Одним із таких рішень є підхід інфраструктури як коду (IaC), що дозволяє створювати, підтримувати та оновлювати інфраструктуру за допомогою програмного коду.

Інфраструктура як код кардинально змінює підхід до управління хмарними інфраструктурами, забезпечуючи можливість автоматизації рутинних процесів, забезпечення надійності, повторюваності операцій та простоти у масштабуванні. Це особливо актуально для компаній, що займаються аутсорсингом, оскільки потребують швидкої адаптації інфраструктури до змін у вимогах замовників, забезпечення високої доступності сервісів та ефективного управління ресурсами.

Одним із ключових інструментів для впровадження IaC є Terraform, що дозволяє автоматизувати створення інфраструктури в хмарних середовищах, таких як Amazon Web Services (AWS). Впровадження Terraform у хмарні рішення дозволяє компаніям не лише пришвидшити процес розгортання інфраструктури, але й забезпечити її стабільність та відповідність вимогам безпеки.

Для досягнення максимального ефекту автоматизації процесів розгортання інфраструктури в хмарі, широко застосовується CI/CD (Continuous Integration/Continuous Deployment) – методологія безперервної інтеграції та розгортання, яка дозволяє безперервно оновлювати додатки та інфраструктуру без простоїв. Поєднання Terraform з CI/CD інструментами, такими як GitLab CI чи Jenkins, дає змогу автоматизувати весь цикл управління інфраструктурою: від розробки до розгортання та моніторингу.

Метою цієї роботи є дослідження та впровадження підходу IaC для підтримки аутсорсингових додатків у хмарних середовищах, таких як AWS, з використанням Terraform та CI/CD. Основними завданнями є:

— аналіз теоретичних основ і практичного застосування IaC;
— розробка та тестування Terraform-коду для створення інфраструктури;
— інтеграція CI/CD для автоматизації процесів;
— оцінка результатів впровадження та аналіз продуктивності системи.

Таким чином, дана робота спрямована на демонстрацію можливостей сучасних технологій для побудови ефективної, надійної та масштабованої інфраструктури в умовах хмарних рішень, що є критично важливим для компаній, які надають послуги аутсорсингу.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ТА СУЧАСНІ ПІДХОДИ ДО ПРОЕКТУВАННЯ ХМАРНОЇ ІНФРАСТРУКТУРИ

1.1 Поняття інфраструктури як коду (IaC)

Концепція інфраструктури як коду (Infrastructure as Code, IaC) виникла як відповідь на зростаючу складність управління IT-інфраструктурою в сучасних компаніях. В 90-х і на початку 2000-х років системні адміністратори використовували ручні підходи для налаштування серверів та інфраструктурних елементів, що потребувало значних затрат часу та підвищувало ризик людських помилок. Впровадження віртуалізації в 2000-х роках дозволило автоматизувати частину цього процесу, але проблема швидкого масштабування і підтримки залишалася актуальною.

З розповсюдженням хмарних технологій у 2010-х роках потреба в автоматизованому управлінні інфраструктурою зросла ще більше. Великі компанії та провайдери хмарних послуг, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud, почали пропонувати інструменти для спрощення управління своїми хмарними ресурсами. Це стало поштовхом для появи інфраструктури як коду, що дозволяє описувати інфраструктуру у вигляді програмного коду, який може бути автоматично виконаний для створення, зміни або видалення необхідних ресурсів.

Перші інструменти для автоматизації управління інфраструктурою з’явилися разом з такими технологіями, як Puppet (2005), Chef (2009), а пізніше і Ansible (2012). Однак Terraform (2014) став першим популярним інструментом, що дозволив абстрагуватися від конкретних провайдерів і працювати з різними хмарними платформами одночасно.

Інфраструктура як код (IaC) — це підхід до управління IT-інфраструктурою через програмний код. Замість ручного налаштування

серверів, мереж та інших елементів інфраструктури, всі конфігурації зберігаються у вигляді файлів коду, що дозволяє автоматизувати процеси створення, оновлення та видалення інфраструктури. Основною ідеєю IaC є те, що інфраструктура стає частиною процесу розробки і керується за допомогою тих самих методів, що і звичайний програмний код, зокрема через систему контролю версій.

Рисунок 1.1 – Використання IaC в сучасному світі

IaC включає два основні підходи:

1. Декларативний підхід (наприклад, Terraform) фокусується на тому, щоб описати кінцевий стан інфраструктури, а інструмент самостійно визначає, як досягти цього стану. Це спрощує управління змінами та мінімізує ризик помилок.
2. Імперативний підхід (наприклад, Ansible) вимагає чітких інструкцій щодо того, як виконати кожен крок процесу розгортання або конфігурації інфраструктури.

IaC забезпечує низку ключових переваг:

1. Автоматизація: дозволяє автоматично розгортати інфраструктуру та швидко її змінювати.
2. Масштабованість: легко адаптується до великих середовищ, дозволяючи одночасно керувати тисячами ресурсів.

3. Відтворюваність: оскільки інфраструктура описана у вигляді коду, будь-які зміни можуть бути точно повторені.
4. Спрощення співпраці: IaC дозволяє кільком командам працювати над інфраструктурою одночасно, оскільки всі конфігурації зберігаються у системах контролю версій.

У сучасних IT-середовищах інфраструктура як код (IaC) відіграє ключову роль у забезпеченні швидкого та безпечного розгортання програмних продуктів. Зі впровадженням хмарних технологій та підходу DevOps, IaC став невід'ємною частиною процесів безперервної інтеграції та безперервного розгортання (CI/CD). Цей підхід значно спрощує управління інфраструктурою, особливо в умовах, де застосовується динамічне масштабування ресурсів або де інфраструктура постійно змінюється відповідно до вимог ринку.

IaC дозволяє компаніям автоматизувати рутинні завдання, що знижує операційні витрати, покращує надійність інфраструктури та мінімізує кількість ручних помилок. Завдяки цьому забезпечується гнучкість та швидка адаптація до змін у бізнес-процесах, а також мінімізуються ризики шляхом тестування змін у контрольованих середовищах перед впровадженням у продуктивну систему.

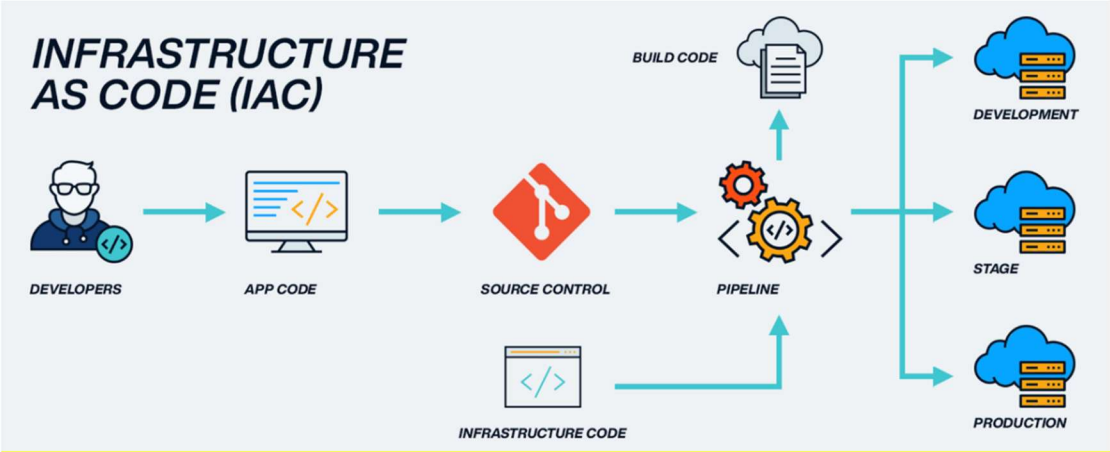


Рисунок 1.2 – Принцип роботи IaC

На практиці IaC є основою для швидкого розгортання інфраструктури в хмарних середовищах, таких як AWS, Google Cloud і Azure. Цей підхід стає незамінним для компаній, що використовують методологію DevOps, а також для команд розробників і системних адміністраторів, які прагнуть пришвидшити процеси та покращити стабільність своїх систем.[1]

1.2 Переваги IaC для компаній, що працюють в аутсорсингу

Інфраструктура як код (Infrastructure as Code, IaC) є однією з ключових технологій, яка відкриває нові можливості для компаній, що займаються аутсорсингом. У сфері аутсорсингу, де швидкість, гнучкість і точність виконання проектів мають вирішальне значення, IaC дозволяє значно покращити якість надання послуг, оптимізувати управління інфраструктурою та забезпечити швидку адаптацію до змін у вимогах замовників.

Автоматизація процесів є одним з основних переваг використання IaC для компаній, що займаються аутсорсингом. В умовах швидких змін і постійних вимог клієнтів, компанії стикаються з потребою часто розгортати нові середовища, тестові майданчики або продуктивні системи. Використання IaC дозволяє автоматизувати ці процеси, що значно скорочує час і витрати на їх виконання.[2]

Завдяки IaC всі ресурси інфраструктури можуть бути створені і налаштовані за кілька хвилин або годин, а не днів чи тижнів, як це було раніше при ручному налаштуванні серверів, мереж і інших елементів. Наприклад, створення нових віртуальних машин, налаштування мережевих з'єднань, баз даних та інших ресурсів може бути виконане автоматично, згідно з налаштуваннями, описаними в коді. Це особливо важливо для компаній-аутсорсерів, які часто працюють з різними клієнтами і повинні швидко адаптувати свою інфраструктуру під конкретні вимоги кожного проекту.

Автоматизація також дозволяє уникнути ручного введення конфігурацій, що є суттєвим джерелом помилок. Замість цього, всі дії з розгортання ресурсів контролюються за допомогою скриптів і конфігураційних файлів, що дозволяє зменшити кількість помилок і забезпечити стабільну роботу інфраструктури. Компанії можуть сфокусуватися на своїх основних завданнях, делегуючи рутинні та трудомісткі завдання автоматизованим системам.

Крім того, автоматизація дає можливість швидко відновлювати інфраструктуру в разі виникнення несправностей або технічних проблем. Оскільки конфігурація інфраструктури зберігається у вигляді коду, в разі збою чи втрати даних компанія може відновити всю інфраструктуру, використовуючи ті ж самі скрипти, які використовувалися під час її початкового розгортання.

Масштабованість є ще однією важливою перевагою IaC для аутсорсингових компаній. Аутсорсингові компанії часто обслуговують велику кількість клієнтів, і їхні потреби можуть швидко змінюватися. Наприклад, один клієнт може раптово вимагати масштабування системи через збільшення навантаження на додаток або необхідність швидко розгортати нові середовища для різних етапів проекту.

IaC дозволяє автоматично масштабувати інфраструктуру, додаючи нові сервери, ресурси або налаштовуючи параметри вже існуючих без значних ручних втручань. Це можливо завдяки використанню таких інструментів, як Terraform або AWS CloudFormation, які дозволяють описати інфраструктуру у вигляді декларативного коду. Таким чином, масштабування виконується простим оновленням конфігураційного файлу та запуском процесу розгортання.[3]

Для компаній-аутсорсерів це є критично важливим, оскільки вони можуть оперативно реагувати на вимоги клієнтів і масштабувати свої проекти залежно від необхідності. Наприклад, якщо клієнт потребує швидкого

розгортання додаткових серверів для підтримки високих навантажень у пікові періоди, IaC дозволяє це зробити без значних витрат часу і ресурсів.

Масштабованість також полегшує підтримку проектів, що потребують розподілу інфраструктури між кількома клієнтами. Аутсорсингові компанії можуть використовувати IaC для керування кількома інфраструктурами одночасно, при цьому кожна з них може бути розгорнута, змінена або масштабована незалежно від інших.

Однією з головних переваг IaC для аутсорсингових компаній є можливість забезпечення повторюваності процесів. Оскільки вся конфігурація інфраструктури описана у вигляді коду, компанії можуть повторювати процеси створення інфраструктури у будь-який час з абсолютною точністю.

Це особливо корисно для компаній, що займаються аутсорсингом, оскільки вони часто працюють з різними клієнтами і проектами, для яких потрібні схожі конфігурації інфраструктури. Наприклад, якщо компанія розробляє веб-додатки для різних клієнтів, вона може використовувати один і той самий шаблон інфраструктури для створення середовищ розробки, тестування і продуктивної системи. Завдяки повторюваності процесів компанія може швидко адаптувати рішення під різні потреби клієнтів, мінімізуючи час на налаштування нових проектів.

IaC дозволяє створювати шаблони конфігурацій, які можна повторно використовувати для різних проектів і клієнтів. Це забезпечує не лише економію часу, але й стабільність, оскільки кожен раз використовується перевірений і протестований код, що знижує ризик помилок. Крім того, повторюваність процесів дозволяє забезпечити єдність у підходах до управління інфраструктурою, що важливо для великих аутсорсингових компаній, які мають численні команди і проекти.

Ручне налаштування інфраструктури є схильним до помилок процесом, який часто призводить до нестабільної роботи систем або порушень у безпеці. Однак, використання IaC значно знижує ризик виникнення таких помилок. Оскільки всі процеси розгортання і конфігурації ресурсів автоматизовані і

виконуються на основі коду, людський фактор практично виключається з рівняння.

Компанії, що займаються аутсорсингом, можуть знизити кількість помилок, пов'язаних з неправильною конфігурацією серверів або іншими ресурсами, завдяки використанню IaC. Після того, як конфігураційні файли були створені і протестовані, їх можна безпечно використовувати для створення середовищ для різних клієнтів і проектів без необхідності ручного втручання в процес.

IaC також дозволяє компаніям відстежувати всі зміни в інфраструктурі за допомогою систем контролю версій, таких як Git. Це дозволяє відкотити зміни або відновити попередню версію конфігурації у випадку помилок або збоїв. Крім того, всі зміни проходять через процес перевірки та тестування перед тим, як бути впровадженими у продуктивне середовище, що додатково знижує ризик виникнення проблем.[4]

1.3 Огляд хмарних платформ та їх сервісів для підтримки додатків

Порівняння основних хмарних платформ: Серед найбільших постачальників хмарних сервісів виділяються Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. Ці платформи надають широкий спектр інструментів для управління та підтримки інфраструктури, що дозволяє автоматизувати процеси, підвищувати продуктивність та масштабованість, а також значно знижувати витрати на адміністрування ресурсів. Кожна з цих платформ має свої унікальні функції, проте AWS залишається лідером ринку завдяки великому досвіду роботи з інфраструктурою як коду (IaC) і підтримці широкого спектра сервісів, оптимальних для аутсорсингових компаній.

Amazon Web Services — найбільш зріла та широко використовувана хмарна платформа, яка пропонує понад 200 сервісів для різноманітних застосувань, включаючи обчислювальні, зберігання, бази даних, аналітику, машинне навчання та інші послуги. Вона є популярною серед великих і середніх аутсорсингових компаній завдяки своїм можливостям з високим рівнем захисту даних, гнучкості й надійності. Основні сервіси AWS, що забезпечують підтримку аутсорсингових додатків, включають:

— EC2 (Elastic Compute Cloud): забезпечує віртуальні сервери для виконання додатків з налаштовуваними характеристиками, що дозволяє легко масштабувати ресурси відповідно до потреб бізнесу.

— S3 (Simple Storage Service): надає високомасштабоване і надійне зберігання даних, придатне для зберігання резервних копій, даних додатків та аналітики. S3 також підтримує різні рівні зберігання для оптимізації вартості.

— RDS (Relational Database Service): управляє реляційними базами даних, включаючи MySQL, PostgreSQL, Oracle та інші. Завдяки автоматизації резервного копіювання і масштабуванню під навантаженнями, цей сервіс дозволяє аутсорсинговим компаніям зосередитись на розробці, а не на адмініструванні баз даних.

— IAM (Identity and Access Management): забезпечує контроль доступу до ресурсів AWS, що є важливим аспектом для аутсорсингових компаній, яким потрібно обмежити доступ розробників і забезпечити безпеку.

Google Cloud Platform розвивається швидкими темпами і часто обирається компаніями, що займаються аналітикою, обробкою великих даних та машинним навчанням. Основна перевага GCP — тісна інтеграція з інструментами Google для обробки даних і штучного інтелекту, такими як BigQuery та AI Platform. GCP також пропонує ефективні інструменти для

DevOps-процесів, але його сервіси для баз даних та віртуальних серверів менш універсальні в порівнянні з AWS.

Microsoft Azure займає друге місце за ринковою часткою після AWS і популярний завдяки своїй інтеграції з продуктами Microsoft, такими як Windows Server, Active Directory та інші. Azure підходить для компаній, які мають корпоративну інфраструктуру, побудовану на рішеннях Microsoft. Azure також має сильні позиції в області гібридних хмарних рішень, що дозволяє компаніям поєднувати власні ресурси з хмарними. Однак, на відміну від AWS, Azure має менш розвинуту підтримку API та частіше стикається з проблемами сумісності з інструментами DevOps.

Оскільки аутсорсингові компанії зазвичай працюють з клієнтами з різних індустрій, вони потребують високого рівня гнучкості і масштабованості для задоволення потреб кожного клієнта. AWS забезпечує аутсорсингові компанії гнучкою інфраструктурою, яка дозволяє швидко адаптувати середовище для різних додатків. Такі сервіси, як EC2 та RDS, підтримують легке масштабування та мінімальні налаштування, що знижує витрати на адміністрування і дозволяє компаніям прискорити час виходу на ринок для нових проєктів. Завдяки IAM компанії можуть управляти доступом до ресурсів, підтримуючи безпеку даних і зберігаючи контроль над конфіденційною інформацією, що є особливо важливим для проєктів в аутсорсингу.

Таким чином, AWS надає оптимальне рішення для аутсорсингових компаній завдяки своїм широким можливостям, перевіреній надійності і гнучкості. Серед доступних сервісів саме AWS забезпечує найбільшу підтримку інструментів для автоматизації процесів і управління інфраструктурою, що робить його ідеальним вибором для побудови IaC та підтримки аутсорсингових додатків.

1.4 Огляд хмарних сервісів EC2, S3, RDS, IAM

Хмарні сервіси стали невід'ємною частиною сучасного IT-середовища, забезпечуючи гнучкість, масштабованість та зручність управління ресурсами. Для компаній, що надають аутсорсингові послуги, хмарні платформи, зокрема Amazon Web Services (AWS), пропонують широкий спектр сервісів для побудови, управління та підтримки додатків, що дозволяє їм оптимізувати свої процеси та знижувати витрати. У цьому огляді розглядаються ключові сервіси AWS — EC2, S3, RDS та IAM, — які є основою для підтримки додатків та інфраструктури аутсорсингових компаній.

Рисунок 1.3 – Найбільш популярні сервіси в Амазоні

Amazon EC2 — це один з основних сервісів AWS, який надає віртуальні сервери (інстанси) для виконання будь-яких додатків у хмарі. EC2 дозволяє масштабувати обчислювальні ресурси в залежності від потреб бізнесу, надаючи можливість швидко запускати нові сервери, змінювати їх характеристики або зупиняти роботу, коли вони не потрібні.

Основні можливості EC2:

— Масштабованість: EC2 надає можливість створювати інстанси різного типу, від малих віртуальних серверів до

високопродуктивних обчислювальних потужностей. Аутсорсингові компанії можуть масштабувати свої сервери залежно від вимог проєкту чи додатків.

— Різноманітність типів інстансів: AWS пропонує безліч типів EC2-інстансів, зокрема для загальних цілей, обчислювальних завдань, пам'яттєво-інтенсивних або графічних обчислень. Це дозволяє вибрати оптимальний тип віртуальних машин для виконання конкретних завдань або додатків.

— Автоматичне масштабування: за допомогою функції Auto Scaling можна автоматично збільшувати або зменшувати кількість запущених EC2-інстансів в залежності від навантаження на додаток. Це особливо важливо для аутсорсингових компаній, де попит на ресурси може змінюватися в залежності від часу чи кількості клієнтів.

— Оптимізація витрат: можливість використання різних варіантів платіжних моделей — від інстансів на вимогу (On-Demand) до зарезервованих (Reserved Instances) та спотових (Spot Instances) інстансів — дозволяє компаніям гнучко керувати витратами на інфраструктуру.

Для аутсорсингових компаній EC2 забезпечує критично важливу обчислювальну інфраструктуру, яка легко адаптується до потреб проєктів. Завдяки гнучкості EC2 компанії можуть швидко розгортати нові проєкти або додатки, тестувати їх, масштабувати інфраструктуру під робочі навантаження клієнтів, а також контролювати витрати, використовуючи спотові інстанси або інші оптимальні моделі розрахунку.

Amazon S3 є провідним сервісом для зберігання об'єктів у хмарі. Він дозволяє зберігати будь-які типи даних — від медіафайлів до резервних копій або журналів додатків. Простота у використанні та висока надійність роблять S3 важливим компонентом для аутсорсингових компаній, які займаються розробкою або підтримкою додатків.

Основні можливості S3:

— Масштабованість і гнучкість зберігання: S3 може зберігати необмежену кількість даних, при цьому не вимагаючи від компаній турбуватися про фізичну інфраструктуру для зберігання. Це дозволяє аутсорсинговим компаніям гнучко керувати своїми даними, зберігаючи все — від невеликих текстових файлів до великих відеоархівів.

— Безпека даних: S3 підтримує шифрування як під час передачі, так і при зберіганні даних, а також дозволяє визначати детальні політики доступу для користувачів або додатків. Це особливо важливо для компаній, які працюють з конфіденційними даними своїх клієнтів.

— Інтеграція з іншими сервісами AWS: S3 може використовуватися в поєднанні з іншими хмарними сервісами, такими як CloudFront для доставки контенту, Lambda для обробки подій або S3 Glacier для довгострокового зберігання архівних даних.

— Доступність і стійкість: S3 забезпечує надзвичайно високу доступність і стійкість даних, що робить його ідеальним рішенням для зберігання критично важливих даних додатків або клієнтських проектів.

S3 ідеально підходить для зберігання великих обсягів даних, таких як бекапи додатків, користувацькі файли, мультимедіа або журнали. Для аутсорсингових компаній це забезпечує безпеку даних клієнтів і легкий доступ до них для обробки, аналітики або резервного копіювання. Також S3 дозволяє гнучко керувати вартістю зберігання за рахунок вибору відповідного класу зберігання залежно від частоти доступу до даних.

Amazon RDS є керованим сервісом реляційних баз даних у хмарі. AWS надає можливість працювати з популярними СУБД, такими як MySQL, PostgreSQL, Oracle, SQL Server та інші, забезпечуючи автоматизацію завдань адміністрування, таких як резервне копіювання, масштабування і оновлення.

Основні можливості RDS:

— Підтримка різних СУБД: RDS підтримує кілька різних типів баз даних, дозволяючи обирати ту, яка найбільше підходить під специфіку проєкту. Це дає аутсорсинговим компаніям можливість надавати клієнтам рішення на основі будь-якої популярної бази даних.

— Автоматизація управління: RDS автоматизує багато завдань адміністрування, таких як регулярне резервне копіювання, патчинг бази даних і моніторинг продуктивності. Це знижує навантаження на інженерів і дозволяє більше зосередитись на розробці та підтримці додатків.

— Масштабованість: RDS дозволяє легко масштабувати обчислювальні ресурси або дисковий простір бази даних залежно від зростання вимог додатку або проєкту.

— Безпека та високий рівень доступності: RDS забезпечує шифрування даних під час передачі та зберігання, а також надає можливості для створення резервних копій і налаштування реплікації для високої доступності.

Для аутсорсингових компаній, що надають послуги з розробки та підтримки додатків, RDS дозволяє автоматизувати багато рутинних завдань адміністрування баз даних, що підвищує ефективність команди та скорочує час на обслуговування. Крім того, масштабованість RDS дозволяє без проблем адаптувати базу даних до змін у навантаженнях на додаток.

AWS IAM — це сервіс для управління доступом до ресурсів AWS. Він дозволяє створювати політики доступу для користувачів, груп, ролей і сервісних акаунтів, забезпечуючи належний контроль і безпеку доступу до хмарної інфраструктури.

Основні можливості IAM:

— Гранульовані права доступу: IAM дозволяє детально налаштувати права доступу до кожного ресурсу AWS, що забезпечує максимальний контроль над тим, хто і що може робити з хмарними ресурсами.
— Ролі та політики: компанії можуть створювати ролі, яким надаються певні права, і використовувати їх для контролю доступу користувачів або сервісів до інфраструктури.
— Аудит і моніторинг доступу: За допомогою IAM можна відстежувати і контролювати всі дії користувачів, створюючи детальний лог доступу, що важливо для безпеки аутсорсингових компаній, які працюють з чутливими даними клієнтів.

Для аутсорсингових компаній IAM забезпечує захист даних клієнтів і контроль над доступом до інфраструктури. Це дозволяє гарантувати, що тільки авторизовані користувачі мають доступ до критичних ресурсів або систем, а також мінімізує ризик людських помилок чи зловживань.

Кожен з цих сервісів — EC2, S3, RDS та IAM — забезпечує аутсорсинговим компаніям надійну і масштабовану інфраструктуру для підтримки додатків та управління хмарними ресурсами. AWS пропонує високу гнучкість, інтеграцію з іншими сервісами та автоматизацію процесів, що дозволяє компаніям надавати клієнтам стабільні та безпечні рішення.[9]

1.5 Архітектурні підходи до проєктування хмарної інфраструктури для додатків

Проєктування хмарної інфраструктури для додатків є ключовим завданням для будь-якої компанії, зокрема аутсорсингових фірм, які надають IT-послуги іншим підприємствам. Правильно побудована архітектура

забезпечує надійність, продуктивність, безпеку та доступність додатків, що є критично важливими для успішного функціонування бізнесу. Нижче розглянуто основні архітектурні підходи до проєктування хмарної інфраструктури з урахуванням цих ключових аспектів.

Безпека хмарної інфраструктури. Безпека є найважливішим аспектом при розробці хмарної інфраструктури, особливо для аутсорсингових компаній, які часто працюють з конфіденційними даними своїх клієнтів. Архітектурні рішення мають гарантувати захист даних на всіх етапах їхньої обробки — від зберігання до передачі.

Основні принципи безпеки хмарної інфраструктури:

1. Шифрування даних: Важливо впроваджувати шифрування як під час передачі (SSL/TLS), так і при зберіганні даних (AES-256 або інші стандарти). AWS пропонує вбудовані механізми для шифрування, такі як шифрування в S3, RDS і інших сервісах.
3. Мультифакторна автентифікація (MFA): для доступу до критичних систем і ресурсів рекомендується використовувати MFA, що значно ускладнює несанкціонований доступ до хмарної інфраструктури.
4. Управління доступом через IAM: AWS IAM дозволяє створювати ролі і політики доступу з гранульованими правами, що мінімізує ризики надмірних прав доступу. Принцип "найменшого привілею" означає, що користувачі повинні мати лише ті права, які потрібні для виконання своїх завдань.
5. Сегментація мережі та VPC: За допомогою Amazon VPC (Virtual Private Cloud) можна ізолювати ресурси додатків і налаштовувати підмережі, правила маршрутизації та брандмауери для захисту інфраструктури від зовнішніх загроз. Важливо також впроваджувати DMZ-зони для публічних ресурсів.
6. Журнали та моніторинг: Інструменти моніторингу, такі як Amazon CloudWatch і AWS CloudTrail, допомагають відстежувати

активність користувачів і додатків, дозволяючи швидко виявляти потенційні загрози або аномальні дії.

Продуктивність хмарної інфраструктури. Продуктивність хмарної інфраструктури залежить від того, як правильно налаштовані ресурси для виконання додатків і обробки даних. Архітектурні рішення повинні забезпечувати високу пропускну здатність, низьку затримку і ефективне використання обчислювальних ресурсів.

Основні принципи підвищення продуктивності:

1. Вибір оптимальних типів інстансів EC2: AWS пропонує широкий вибір інстансів для різних типів навантажень — від стандартних (T-серія) до обчислювально-інтенсивних (C-серія) або пам'яттєво-інтенсивних (R-серія). Для кожного додатку необхідно вибирати оптимальний тип інстансів з огляду на його вимоги до процесора, пам'яті та дискового простору.
2. Автоматичне масштабування: Використання Auto Scaling дозволяє автоматично збільшувати або зменшувати кількість EC2-інстансів залежно від навантаження на додаток. Це гарантує, що система працюватиме оптимально, навіть коли навантаження змінюється.
3. Оптимізація зберігання: Використання високопродуктивних дисків, таких як Amazon EBS (Elastic Block Store) або Amazon S3 для зберігання файлів і баз даних, дозволяє знизити затримки при доступі до даних. Використання EBS об'ємом під конкретні навантаження або використання S3 для великих обсягів даних знижує витрати на зберігання та покращує продуктивність додатку.
4. Кешування даних: Впровадження кешування через сервіси, такі як Amazon ElastiCache (Redis, Memcached) або CloudFront, дозволяє зменшити затримки доступу до часто запитуваних даних. Це критично важливо для додатків, що обробляють великий обсяг запитів від користувачів.

Надійність хмарної інфраструктури. Надійність є одним із ключових аспектів, який гарантує безперервність роботи додатків і мінімізацію простоїв. Архітектура повинна бути побудована таким чином, щоб додатки могли працювати навіть за умови збоїв в окремих компонентах або сервісах.

Основні підходи до забезпечення надійності:

1. Реплікація даних та резервні копії: Використання сервісів резервного копіювання, таких як AWS Backup або функції автоматичного резервного копіювання у RDS, дозволяє зберігати копії критичних даних і швидко їх відновлювати у разі збою. Реплікація баз даних між кількома регіонами забезпечує доступність навіть у разі серйозних збоїв.
2. Розподіл навантаження: Elastic Load Balancing (ELB) дозволяє автоматично розподіляти трафік між кількома EC2-інстансами, забезпечуючи балансування навантаження і підвищуючи стійкість додатку до перевантажень. Це дозволяє підтримувати додаток в робочому стані навіть у разі виходу з ладу одного або кількох інстансів.
3. Використання кількох зон доступності (Availability Zones): AWS надає можливість розміщувати ресурси в кількох зонах доступності, що знижує ймовірність простою додатку через збої в інфраструктурі. Використання мультизональних архітектур забезпечує стійкість додатку до збоїв у межах однієї зони доступності.
4. Моніторинг та сповіщення: Впровадження системи моніторингу за допомогою Amazon CloudWatch, AWS Health та налаштування сповіщень допомагає оперативно реагувати на збої або погіршення продуктивності додатків.

Доступність хмарної інфраструктури. Доступність додатків є критичною, особливо для бізнесів, які потребують безперервної роботи своїх

сервісів. Архітектурні рішення повинні гарантувати, що додатки будуть доступні для користувачів 24/7 без значних простоїв.

Основні підходи до забезпечення доступності:

1. Глобальний масштаб: AWS надає можливість розгортати додатки в різних географічних регіонах, використовуючи Amazon Route 53 для геолокаційного маршрутизації запитів. Це підвищує доступність додатку для користувачів по всьому світу.
2. Зони відмовостійкості (Fault Tolerance): Використання архітектур з підтримкою відмовостійкості, таких як резервування EC2-інстансів або RDS баз даних в декількох регіонах, забезпечує відновлення роботи додатку після збоїв з мінімальним часом простою.
3. CDN (Content Delivery Network): Використання Amazon CloudFront для кешування статичних ресурсів і розповсюдження їх через мережу серверів по всьому світу знижує затримки при доступі до додатку та підвищує доступність для користувачів з різних регіонів.
4. Автоматичне відновлення після збоїв: AWS підтримує функціонал для автоматичного перезавантаження інстансів або їх міграції у разі збою на рівні інфраструктури, що забезпечує мінімальний час простою додатку.

Проєктування хмарної інфраструктури для додатків вимагає врахування низки важливих факторів — безпеки, продуктивності, надійності та доступності. Правильно налаштована архітектура гарантує ефективну роботу додатків, їхню стійкість до збоїв та забезпечення високої якості обслуговування користувачів, що є особливо важливим для аутсорсингових компаній, які працюють з різноманітними клієнтами та проєктами. AWS надає потужні інструменти, що допомагають досягти всіх цих цілей, забезпечуючи масштабованість і надійність інфраструктури.[11]

1.6 Огляд найпопулярніших інструментів IaC

Інфраструктура як код (IaC) стала основою для сучасного управління IT-ресурсами в компаніях, які використовують хмарні технології, особливо таких, як AWS (Amazon Web Services). Для реалізації IaC існує кілька популярних інструментів, серед яких найбільш відомими є Terraform, Ansible і Chef. Кожен з цих інструментів має свої унікальні характеристики, підходи до автоматизації та можливості інтеграції з хмарними платформами. Вибір між цими рішеннями залежить від специфіки проєкту, вимог компанії і особливостей хмарної інфраструктури. Нижче представлено детальний огляд кожного інструмента, порівняння їхніх можливостей і аналіз того, який з них є найбільш оптимальним для роботи з AWS.[5]

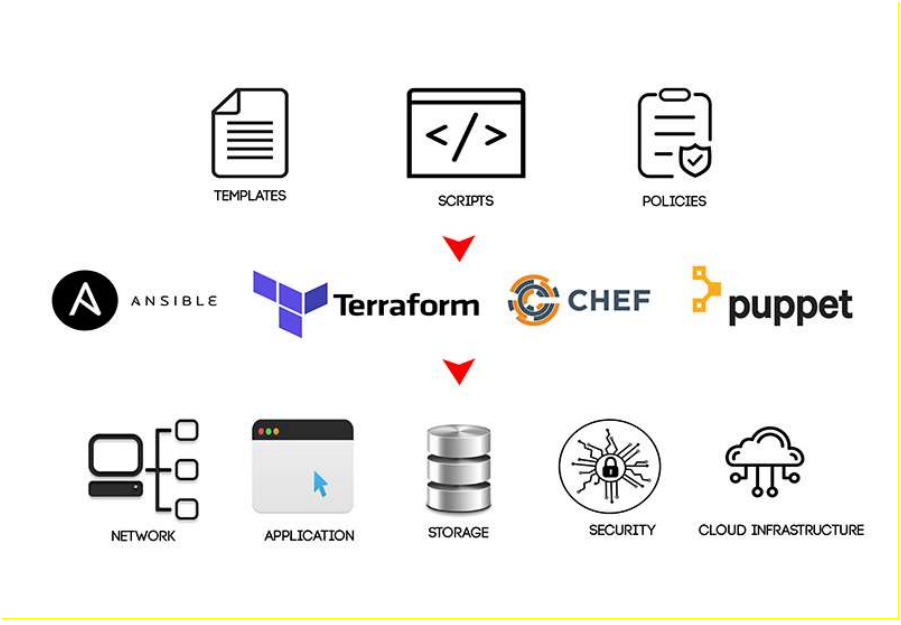


Рисунок 1.4 – Сучасні інструменти для використання IaC

Terraform. Terraform є одним з найпопулярніших інструментів для реалізації інфраструктури як коду, створений компанією HashiCorp у 2014 році. Він підтримує декларативний підхід до управління інфраструктурою і широко використовується для роботи з різними хмарними платформами,

зокрема AWS, Azure, Google Cloud та іншими. Terraform дозволяє описати інфраструктуру у вигляді файлів конфігурації, використовуючи свій власний формат — HCL (HashiCorp Configuration Language).

Terraform має низку важливих переваг, які роблять його ефективним інструментом для управління інфраструктурою. Однією з головних переваг є його хмарна незалежність, що дозволяє працювати з кількома хмарними провайдерами одночасно. Це спрощує управління інфраструктурою в різних середовищах, без необхідності освоєння окремих інструментів для кожного провайдера.

Інструмент використовує декларативний підхід, де користувачі описують бажаний стан інфраструктури, а Terraform автоматично визначає, як досягти цього стану. Це значно полегшує управління змінами та забезпечує чіткий контроль над конфігураціями.

Ще однією перевагою є масштабованість. Завдяки Terraform компанії можуть легко додавати нові сервери чи налаштовувати інші ресурси автоматично. Паралельне створення ресурсів дозволяє значно пришвидшити процес розгортання інфраструктури. Крім того, інструмент відстежує стан середовища через файл стану (state file), що дає актуальне уявлення про інфраструктуру та дозволяє синхронізувати зміни.

Однак Terraform має і недоліки. Файл стану, попри свою корисність, може стати вразливим місцем, особливо для великих інфраструктур. Неправильне управління ним може призвести до серйозних проблем із синхронізацією. Крім цього, Terraform має обмежені можливості для управління внутрішньою конфігурацією серверів. Для цих завдань краще використовувати інші інструменти, наприклад, Ansible, які спеціалізуються на налаштуванні операційних систем і програмного забезпечення.[6]

Ansible. Створений у 2012 році, є інструментом для автоматизації управління конфігурацією та розгортанням інфраструктури, розробленим компанією Red Hat. Ansible використовує імперативний підхід і дозволяє

описувати процеси налаштування серверів через YAML-файли, відомі як Playbooks.

Ansible є одним із найпростіших інструментів для налаштування та використання у сфері інфраструктури як коду. Він не потребує встановлення спеціального програмного забезпечення на хостах, з якими працює, а весь процес керується через SSH-з'єднання.

Інструмент використовує імперативний підхід, що дозволяє користувачам чітко контролювати кожен етап налаштування інфраструктури. Це забезпечує можливість точно вказувати, що і коли має бути виконано.

Однією з головних переваг Ansible є його здатність не лише створювати ресурси, а й керувати конфігурацією програмного забезпечення, налаштуванням серверів і забезпеченням безпеки. Завдяки цьому Ansible особливо корисний для великих середовищ, де необхідно детально контролювати всі етапи процесу.

Попри свої переваги, Ansible має певні обмеження. Він менш ефективний для створення хмарних ресурсів, таких як S3-бакети або бази даних у AWS, порівняно з інструментами, спеціалізованими на таких завданнях, наприклад, Terraform. Ansible більше орієнтований на управління конфігурацією, ніж на створення хмарної інфраструктури.

Імперативний підхід, хоч і забезпечує високий рівень контролю, може стати недоліком у великих проектах. Оскільки користувачі повинні вручну вказувати кожен крок, це може ускладнити підтримку інфраструктури та збільшити час, необхідний для виконання завдань.

Chef. Це ще один популярний інструмент для управління інфраструктурою як кодом, створений у 2009 році. Він фокусується на автоматизації управління конфігураціями серверів і систем. Chef використовує імперативний підхід і дозволяє описувати налаштування серверів за допомогою мови програмування Ruby.

Chef є одним із найпотужніших інструментів для управління конфігурацією серверів, дозволяючи налаштовувати всі аспекти серверів —

від встановлення програмного забезпечення до складних сценаріїв розгортання.

Цей інструмент ідеально підходить для масштабних інфраструктур, оскільки здатен ефективно керувати тисячами серверів одночасно. Гнучкість Chef забезпечується використанням мови програмування Ruby, що дозволяє створювати складні та індивідуальні сценарії конфігурації.

Попри свою потужність, Chef має кілька недоліків. Він є відносно складним для налаштування та використання, особливо для новачків. Використання Ruby як основної мови конфігурації може стати проблемою для тих, хто не володіє цією мовою.

Ще одним недоліком є повільніше виконання завдань порівняно з іншими інструментами, такими як Ansible, що пов’язано з більш складними процесами конфігурації. Chef може вимагати більше часу на розгортання, що може вплинути на швидкість виконання завдань у динамічних середовищах.

Для управління інфраструктурою на AWS, кожен з цих інструментів має свої переваги і недоліки, але важливо розуміти їхню взаємодію з конкретною хмарною платформою.

— Terraform є найоптимальнішим вибором для створення хмарних ресурсів в AWS. Він інтегрується безпосередньо з сервісами AWS і дозволяє автоматизувати створення та налаштування таких ресурсів, як EC2, S3, RDS, VPC та інші. Terraform також забезпечує можливість автоматичного управління змінами в інфраструктурі і може працювати з іншими хмарними провайдерами.
— Ansible буде кращим вибором для автоматизації управління конфігурацією вже створених серверів та додатків у AWS. Він дозволяє автоматично розгортати програми, керувати налаштуванням серверів і виконувати завдання з адміністрування.
— Chef підходить для компаній, яким потрібно гнучко керувати складними сценаріями конфігурації серверів і програмного

забезпечення, але він є менш зручним для управління хмарними ресурсами, як-от бази даних або мережеві служби в AWS.

Для роботи з AWS Terraform є найкращим рішенням з огляду на його здатність створювати та керувати широким спектром хмарних ресурсів. Це інструмент, який пропонує декларативний підхід до управління інфраструктурою, добре підтримується в екосистемі AWS і дозволяє легко інтегруватися з іншими хмарними рішеннями.[6]

Ansible може бути використаний у поєднанні з Terraform для управління налаштуванням серверів і додатків, що створює потужний тандем для аутсорсингових компаній, які обслуговують складні хмарні середовища. Chef, через свою складність і фокус на управлінні конфігурацією, може бути корисним лише у специфічних випадках, де потрібно високий рівень гнучкості і контроль за процесом налаштування.

1.7 Як працює Terraform

Отже, детальніше розглянемог цей інструмент. Terraform використовує декларативний синтаксис для опису інфраструктури. Це означає, що замість опису покрокових інструкцій, як це роблять у традиційних сценаріях конфігурації, ви описуєте бажаний кінцевий стан інфраструктури. Наприклад, ви визначаєте, що хочете мати певну кількість серверів EC2, певну базу даних у RDS і кілька об'єктів у сховищі S3. Terraform самостійно розраховує, які операції необхідно виконати, щоб досягти цього стану.

Основні компоненти Terraform:

1. Конфігураційні файли: Вся інфраструктура описується у файлах з розширенням `.tf`. Ці файли містять декларації ресурсів (наприклад, інстансів EC2, кластерів Kubernetes або ресурсів Google Cloud).

2. Планування (Plan): Після створення або зміни конфігураційних файлів, команда `terraform plan` дозволяє побачити, які зміни буде внесено в існуючу інфраструктуру. Цей етап дає змогу розробникам перевірити наслідки змін перед їх застосуванням.
3. Застосування (Apply): Команда `terraform apply` застосовує зміни до інфраструктури, створюючи або модифікуючи ресурси відповідно до конфігураційних файлів.
4. Стан (State): Terraform зберігає поточний стан інфраструктури у файлі стану. Це дає можливість відслідковувати зміни у ресурсах і синхронізувати конфігурації з реальним станом інфраструктури.

Переваги декларативного підходу Terraform:

— ви точно знаєте, як виглядатиме інфраструктура після застосування конфігурацій. Це особливо важливо для великих проєктів, де будь-яка помилка може призвести до великих втрат часу та ресурсів;

— Terraform виконує операції тільки тоді, коли вони необхідні. Якщо інфраструктура вже перебуває в бажаному стані, Terraform не робитиме зайвих дій;

— Terraform автоматично обробляє залежності між ресурсами. Наприклад, якщо ви створюєте базу даних і сервер, який має до неї підключатися, Terraform спочатку створить базу, а потім сервер.

Автоматизація інфраструктури — один із ключових аспектів ефективного управління IT-ресурсами. Ручне налаштування серверів, мережевих елементів, баз даних та інших ресурсів є не лише часозатратним процесом, але й піддає інфраструктуру ризикам людських помилок. Terraform допомагає вирішити ці проблеми, надаючи засоби для автоматизації всіх етапів життєвого циклу інфраструктури — від створення до масштабування та оновлення.

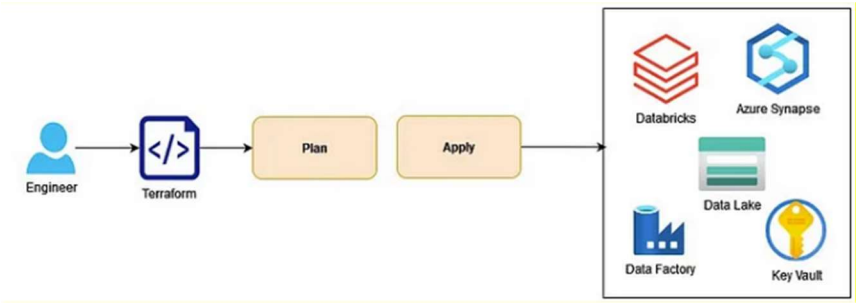


Рисунок 1.6 – Схема роботи Terraform

Основні ролі Terraform в автоматизації інфраструктури. Завдяки Terraform, команди можуть створювати цілі комплекси хмарної інфраструктури за лічені хвилини або години, а не дні або тижні. Вся необхідна інфраструктура описується у вигляді коду, що дає змогу створювати нові середовища одним клацанням або за допомогою команд у CI/CD пайплайні.

Оскільки вся інфраструктура описана у вигляді коду, вона може бути збережена в системах контролю версій (наприклад, Git). Це дозволяє зберігати історію змін, створювати гілки для експериментів, а також працювати з інфраструктурою так само, як із будь-яким іншим програмним забезпеченням. Це спрощує співпрацю між різними командами, зокрема розробниками та адміністраторами.

Коли інфраструктура вже розгорнута, Terraform дає змогу легко вносити зміни та оновлювати ресурси. Наприклад, якщо зростає навантаження на систему, ви можете змінити конфігураційні файли, збільшивши кількість інстансів EC2, і Terraform автоматично внесе ці зміни. Крім того, масштабування відбувається прогнозовано і без ризику помилок.

Terraform часто використовується в поєднанні з системами безперервної інтеграції та доставки (CI/CD), такими як Jenkins, GitLab CI, або CircleCI. Це дозволяє автоматизувати процеси розгортання інфраструктури разом з кодом додатків. Кожен коміт у репозиторії може автоматично запускати оновлення інфраструктури або створювати нові середовища для тестування чи продакшну.

Однією з унікальних особливостей Terraform є його мультихмарність. Він підтримує численні провайдери хмарних послуг (AWS, Microsoft Azure, Google Cloud Platform та інші), а також локальні рішення. Це дозволяє створювати інфраструктури, які можуть одночасно використовувати ресурси різних хмарних провайдерів, а також переносити ресурси між ними.

Автоматизація з використанням Terraform значно знижує ризик помилок, що виникають при ручному управлінні інфраструктурою. Коли всі кроки розгортання та управління описані у вигляді коду, можна легко уникнути невідповідностей між середовищами розробки, тестування і продакшну.

Отже, Terraform є важливим інструментом для автоматизації сучасної хмарної інфраструктури. Завдяки декларативному підходу до управління ресурсами, мультихмарній підтримці та можливостям інтеграції з CI/CD процесами, він дозволяє компаніям зосередитися на вирішенні бізнес-задач, залишаючи управління інфраструктурою під контролем автоматизованих інструментів. Це особливо важливо для аутсорсингових компаній, які часто мають справу з багатьма проєктами, різними клієнтами та складною інфраструктурою. Terraform робить процес розгортання, оновлення та масштабування інфраструктури надійним, швидким і ефективним.[7]

1.8 Основна концепція контейнерів та її значення в сучасній IT-інфраструктурі

Контейнери — це технологія, яка надає змогу створювати ізольоване середовище для запуску додатків з усіма необхідними компонентами, включаючи залежності, налаштування і конфігурації. Завдяки цьому підходу забезпечується незалежність виконання додатка від операційної системи, на

який він запущений, що значно полегшує та прискорює розробку, тестування і розгортання нових версій програмного забезпечення.

Основна ідея контейнеризації полягає у створенні так званого контейнера — самодостатнього блоку, в якому зібрані всі файли і бібліотеки, необхідні для роботи додатка. Це дозволяє уникнути ситуацій, коли додаток працює на одному комп'ютері, але не запускається або працює неправильно на іншому через відмінності у версіях операційної системи, відсутність певних бібліотек чи інші конфігураційні особливості.

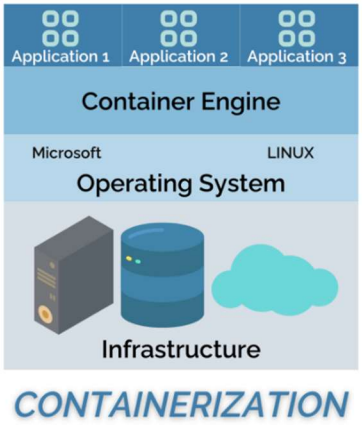


Рисунок 1.7 – Контейнеризація

У традиційному підході для таких цілей зазвичай використовувалися віртуальні машини (VM), які дозволяють запускати декілька ізольованих операційних систем на одному фізичному сервері. Проте віртуальні машини мають досить великі вимоги до ресурсів, оскільки вони включають в себе повноцінну операційну систему та вимагають значного обсягу пам'яті і обчислювальних потужностей для запуску кожного віртуального середовища. Віртуальні машини також створюють додаткове навантаження на гіпервізор (компонент, що керує віртуалізацією), що ще більше збільшує споживання ресурсів.[15]

Контейнери, на відміну від віртуальних машин, не включають в себе повноцінну операційну систему; натомість, вони використовують ядро базової ОС. Це знижує вимоги до обсягу пам’яті, прискорює процес запуску

контейнерів та оптимізує загальне використання ресурсів. Легкість і швидкість контейнерів робить їх чудовим вибором для мікросервісної архітектури, де кожен мікросервіс може бути ізольованим в окремому контейнері. Це значно полегшує масштабування, моніторинг, тестування і випуск оновлень для конкретних компонентів без втручання в роботу інших частин системи.[13]

Переваги використання контейнерів:

1. Завдяки контейнеризації розробники можуть бути впевненими, що додаток буде працювати однаково незалежно від середовища — будь то комп'ютер розробника, сервер у локальному дата-центрі чи хмарний сервер.
2. Запуск контейнера відбувається значно швидше, ніж запуск віртуальної машини, що дозволяє швидко розгортати нові версії та тестувати зміни без простоїв.
3. Контейнери споживають менше пам'яті і обчислювальних ресурсів у порівнянні з віртуальними машинами, оскільки вони не потребують повноцінної ОС.
4. Легкість контейнерів дозволяє швидко масштабувати додатки шляхом додавання або видалення контейнерів в залежності від навантаження.

1.9 Порівняння Docker з іншими контейнеризаційними інструментами

Контейнери — це технологія, що дозволяє упаковувати додаток з усіма необхідними залежностями, конфігураціями і середовищем, необхідними для його роботи, в ізольований від системи блок. Контейнери дозволяють

запускати додатки незалежно від операційної системи, забезпечуючи гнучкість, масштабованість і легкість в обслуговуванні інфраструктури.

Основна концепція контейнеризації полягає у створенні середовища, яке дозволяє запускати додаток так само, як і на розробницькому комп'ютері, і на продуктивному сервері, незалежно від різниці в налаштуваннях або операційній системі. Контейнери створюються легкими та швидкими для завантаження, на відміну від традиційних віртуальних машин, і, зазвичай, не включають в себе цілу операційну систему, що зменшує вимоги до ресурсів та часу розгортання.

Рисунок 1.8 – Популярні інструменти для контейнеризації

Серед популярних інструментів для створення контейнерів можна виділити Docker, Podman, LXC та OpenVZ, а для управління контейнерами на кластерному рівні найпоширенішим інструментом є Kubernetes. Давайте розглянемо переваги, недоліки та особливості кожного з цих інструментів.

Docker — це найпоширеніша платформа контейнеризації, розроблена в 2013 році. Основна мета Docker — спростити процес розгортання та обслуговування додатків, ізолювавши їх від базової операційної системи за допомогою контейнерів.

Переваги Docker:

— має інтуїтивний інтерфейс і велику кількість інструментів, що полегшують створення та управління контейнерами;
— дозволяє запускати контейнери на різних платформах та операційних системах без потреби змінювати код додатка;
— контейнери завантажуються набагато швидше, ніж традиційні віртуальні машини;
— Docker Hub — публічний репозиторій для образів, що дозволяє розробникам обмінюватись контейнерами.

Однак Docker має певні недоліки, зокрема залежність від сервісу Docker Daemon, який потребує підвищених прав, що може створювати ризики з точки зору безпеки. Це обмеження надало поштовх до розвитку альтернативних рішень.

Podman — це альтернатива Docker, що також використовується для створення та управління контейнерами, але з іншою архітектурою. Podman розроблений компанією Red Hat і відомий своєю здатністю працювати без демон-сервісу.

Переваги Podman:

— працює без фонової служби, яка постійно виконується з підвищеними правами (root), що робить його більш безпечним;
— використовує той самий формат контейнерів, що й Docker, тож Docker-образи можна запускати через Podman;
— підтримує rootless-контейнери, що дозволяє запускати контейнери без адміністративних прав, знижуючи ризики для безпеки.

Недоліки Podman пов'язані з менш розвиненою екосистемою, оскільки він не має такого широкого вибору інструментів і репозиторіїв, як Docker. Крім того, деякі інструменти та сервіси, що базуються на Docker, можуть потребувати додаткової адаптації для роботи з Podman.

LXC (Linux Containers) — це контейнерна технологія, яка забезпечує рівень віртуалізації, ближчий до традиційних віртуальних машин. LXC дозволяє запускати кілька ізольованих Linux-систем на одному хості, і він є попередником сучасних контейнерних рішень.

Переваги LXC:

— забезпечує більшу ізоляцію між контейнерами та базовою операційною системою;

— дозволяє тонке налаштування ресурсів, що особливо корисно для великих і критичних додатків;

— має високу продуктивність і менші накладні витрати, оскільки працює на рівні ядра системи.

Проте, LXC може бути більш складним у налаштуванні та управлінні порівняно з Docker чи Podman. Він вимагає більшої технічної компетентності і не має такої ж розвиненої екосистеми.

OpenVZ — це ще одна технологія контейнеризації, що дозволяє запускати ізольовані Linux-контейнери з доступом до спільного ядра. Вона забезпечує повну віртуалізацію ОС і дозволяє ефективно використовувати ресурси.

Переваги OpenVZ:

— висока ефективність використання ресурсів: OpenVZ дозволяє запускати десятки контейнерів на одному сервері з мінімальними накладними витратами;

— віртуалізація ОС: OpenVZ працює на рівні ОС, що дозволяє створювати багато контейнерів з однією спільною копією ядра.

Основним недоліком OpenVZ є його підтримка лише на Linux, а також відсутність можливості запустити контейнери на інших операційних системах. Це обмеження може бути перешкодою для використання в мультиопераційних середовищах.

Kubernetes — це платформа для оркестрації контейнерів, яка автоматизує розгортання, масштабування і управління контейнеризованими додатками. Kubernetes був розроблений Google і тепер підтримується фондом Cloud Native Computing Foundation (CNCF).

Переваги Kubernetes:

— Kubernetes може автоматично масштабувати контейнери на основі навантаження, забезпечуючи оптимальне використання ресурсів;
— при збоях Kubernetes автоматично перезапускає контейнери та перерозподіляє навантаження;
— підтримує роботу з тисячами контейнерів у великих середовищах, що дозволяє компаніям безперервно масштабувати свої додатки;
— Kubernetes підтримує інтеграцію з системами управління конфігураціями, що дозволяє ефективно керувати додатками в різних середовищах.

Недоліки Kubernetes пов'язані з його складністю в налаштуванні та керуванні, що може вимагати спеціальних знань і ресурсів.[14]

Кожен інструмент має свої переваги та недоліки, і вибір залежить від конкретних потреб компанії. Docker є чудовим варіантом для швидкого розгортання та широкої сумісності, тоді як Podman надає альтернативу з підвищеним рівнем безпеки. LXC та OpenVZ підходять для сценаріїв з високою ізоляцією і продуктивністю, але мають більшу складність. Kubernetes, в свою чергу, є незамінним інструментом для оркестрації контейнерів, забезпечуючи високу надійність та масштабованість для великих інфраструктур.

1.10 Концепція CI/CD, цілі та порівняння основних інструментів

CI/CD (Continuous Integration/Continuous Deployment) — це набір практик та інструментів, які забезпечують безперервну інтеграцію та розгортання програмного забезпечення. Мета CI/CD — автоматизувати та прискорити процес розробки, тестування та впровадження програмного забезпечення, що особливо важливо у середовищах з частими оновленнями та розподіленими командами.

Continuous Integration (CI) — це практика регулярного інтегрування коду в основну гілку репозиторію декількома розробниками. При кожному оновленні коду запускаються автоматичні тести, що дозволяє оперативно виявляти і усувати помилки. CI дозволяє скоротити час між внесенням змін і їх перевіркою, що, в свою чергу, підвищує якість коду та забезпечує швидкий зворотний зв'язок.

Continuous Deployment (CD) — це практика автоматичного розгортання нових версій програмного забезпечення на продуктивні сервери одразу після успішного проходження тестів. Це дозволяє швидко доставляти нові функції та виправлення користувачам, скорочуючи час виведення продукту на ринок. CD також включає етапи контролю якості, щоб забезпечити стабільність та надійність нових релізів.[17]

Ці практики дають змогу уникнути довгих циклів розгортання і ручних перевірок, замінюючи їх на короткі автоматизовані етапи, що забезпечує постійне оновлення і підтримку актуальності коду. Основні переваги CI/CD включають:

- Швидкість розгортання: нові версії продукту можна доставляти на ринок швидше.
- Підвищення якості: автоматизовані тести дозволяють швидко ідентифікувати помилки, що сприяє покращенню якості коду.

— Зниження ризиків: часте розгортання і тестування дозволяють виявляти і усувати помилки на ранніх стадіях.

— Ефективність команди: розробники витрачають менше часу на інтеграцію та тестування, зосереджуючись на написанні нового функціоналу.[18]

Таблиця 1.1 – Порівння інструментів

Інструмент	**Переваги**	**Недоліки**	**Найкращий вибір для**
GitLab CI/CD	Інтеграція з GitLab, проста конфігурація	Обмежена підтримка інших VCS	Проекти на GitLab
Jenkins	Велика кількість плагінів, висока налаштовуваність	Складність у налаштуванні, потребує ресурсів	Великі команди з унікальними вимогами
GitHub Actions	Простота використання, інтеграція з GitHub	Залежність від GitHub, обмежений функціонал	Проекти на GitHub
CircleCI	Швидка інтеграція з Docker, проста хмарна конфігурація	Високі витрати на преміум, обмеження для локальних рішень	Компанії з акцентом на Docker

Кожен з перелічених інструментів має свої переваги та підходить для різних типів проєктів. GitLab CI/CD та GitHub Actions є чудовими варіантами для команд, які вже користуються відповідними платформами для контролю версій. Jenkins пропонує високу кастомізацію та сумісність з широким спектром інструментів, що робить його вибором номер один для великих команд, які потребують унікальних налаштувань. CircleCI ідеально підходить для команд, що активно використовують Docker і потребують швидкого розгортання в хмарі.[19]

РОЗДІЛ 2 ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФРАСТРУКТУРИ ДЛЯ ПІДТРИМКИ АУТСОРСИНГОВИХ ДОДАТКІВ

2.1 Практична реалізація інфраструктури додатку в AWS за допомогою Terraform

Мета практичної частини — створення інфраструктури для підтримки додатку аутсорсингової компанії. Основна ціль цієї інфраструктури — забезпечити надійну, автоматизовану та масштабовану платформу, яка легко адаптується до мінливих вимог бізнесу і здатна обслуговувати значну кількість користувачів. Щоб досягти цих вимог, інфраструктура має бути орієнтована на високий рівень відмовостійкості та безпеки, мінімізуючи людські помилки завдяки автоматизації.

Використання AWS як основної хмарної платформи дає значні переваги, оскільки вона надає широкий спектр керованих сервісів, що дозволяє сфокусуватися на основних завданнях додатку, замість управління фізичними серверами та їх конфігурацією. Натомість, Terraform як інструмент управління інфраструктурою як кодом (IaC) допомагає зберігати всі конфігурації в зрозумілому та повторюваному вигляді, який можна з легкістю оновлювати та масштабувати відповідно до потреб проєкту.

Основні компоненти інфраструктури

1. EC2 (Elastic Compute Cloud) — основний сервіс для запуску віртуальних серверів, на яких виконується бізнес-логіка додатку. Завдяки EC2 легко змінювати обчислювальну потужність, додаючи нові інстанси або змінюючи конфігурації за необхідності. Використання EC2 в Auto Scaling групах дозволяє автоматично додавати або видаляти ресурси в залежності від навантаження;

2. S3 (Simple Storage Service) — служба для зберігання статичного контенту, як-от файли користувачів, резервні копії або статичні ресурси додатку. S3 відомий своєю високою доступністю і масштабованістю, а також вбудованими функціями для управління правами доступу та інтеграції з іншими AWS сервісами;
3. Amazon RDS (Relational Database Service) — керована служба для роботи з реляційними базами даних, що дозволяє обслуговувати критичні дані додатку без необхідності адміністрування сервера бази даних. RDS забезпечує автоматичне резервне копіювання, моніторинг продуктивності і відновлення після збоїв, що особливо важливо для додатків, де критичні дані мають бути доступними в будь-який час;
4. Route 53 — система управління DNS-запитами, яка дозволяє направляти користувачів до оптимального ресурсу залежно від їх місцезнаходження та інших критеріїв. Route 53 забезпечує надійне та швидке підключення користувачів до додатку, а також інтеграцію з іншими сервісами AWS для забезпечення безперебійного обслуговування;
5. CloudFront — система доставки контенту (CDN), яка зменшує затримку при доставці контенту користувачам шляхом кешування даних у різних регіонах світу. Це особливо важливо для великих додатків, де швидкість відгуку має вирішальне значення для досвіду користувача;
6. IAM (Identity and Access Management) — інструмент для управління доступом до ресурсів AWS, що дозволяє створювати детальні політики прав доступу для користувачів і сервісів. IAM забезпечує надійний захист даних і ресурсів, дозволяючи обмежити доступ лише для авторизованих користувачів та служб;

7. CloudWatch — сервіс моніторингу та логування, що дозволяє відстежувати продуктивність інфраструктури та реагувати на можливі збої. За допомогою CloudWatch можна отримувати сповіщення, встановлювати порогові значення для ресурсів та автоматично реагувати на події, що знижує час на виявлення та усунення проблем.

2.2 Важливість використання AWS та Terraform

AWS є однією з найпопулярніших хмарних платформ, яка забезпечує широкий спектр послуг для розгортання та підтримки додатків у хмарі. Завдяки таким інструментам, як EC2, RDS, і S3, AWS надає рішення, які дозволяють зосередитися на функціональності додатку, а не на управлінні інфраструктурою. Важливою перевагою AWS є її здатність до горизонтального та вертикального масштабування, що дозволяє автоматично адаптувати інфраструктуру до поточного навантаження, уникаючи перевитрат ресурсів.

Terraform, як інструмент для управління інфраструктурою як кодом, є незамінним для проєктів, де потрібна висока автоматизація та швидкість розгортання. Використання Terraform має наступні переваги:

1. Автоматизація процесу розгортання: вручну налаштовувати інфраструктуру — це трудомісткий та схильний до помилок процес, особливо при складних конфігураціях. Завдяки Terraform інфраструктура може бути розгорнута автоматично за допомогою коду.
2. Повторюваність та консистентність: інфраструктура як код дозволяє створювати відтворювані середовища, що важливо для тестування, розробки та продуктивного використання.

3. Масштабованість: Terraform дозволяє легко змінювати конфігурації, додаючи або видаляючи ресурси в залежності від потреб додатку, що дає змогу масштабувати інфраструктуру відповідно до поточного навантаження.
4. Простота версійного контролю: Terraform-файли можуть зберігатися в системах контролю версій, що полегшує відстеження змін, усунення помилок та забезпечення стабільності інфраструктури. Це особливо важливо для командної роботи, де всі члени можуть бачити історію змін та коментарі до них.

У результаті, використання AWS та Terraform забезпечує надійну, масштабовану, і економічно вигідну інфраструктуру, яка відповідає потребам сучасних аутсорсингових компаній. Це поєднання дозволяє створити автоматизовану систему, яка легко адаптується до змінних бізнес-вимог та мінімізує час на налаштування і обслуговування.

2.3 Ідентифікація необхідних ресурсів для інфраструктури та обґрунтування їх вибору

Для побудови надійної та безпечної мережевої інфраструктури для підтримки додатків аутсорсингової компанії в AWS, були ідентифіковані та обрані наступні ресурси. Головні пукнти це:

— мережева інфраструктура;
— обчислювальні ресурси;
— база даних;
— сховище об’єктів;
— безпека та контроль доступу.

Мережева інфраструктура. VPC (Virtual Private Cloud) є основою для побудови ізольованого віртуального мережевого середовища, де можна створювати різні ресурси та керувати трафіком між ними.

- використання власного VPC дозволяє ізолювати ресурси від загальнодоступної мережі AWS і контролювати вхідний та вихідний трафік;
- VPC має визначений блок CIDR, що забезпечує унікальну IP-адресацію для всіх ресурсів;
- ідентифікатор та теги були налаштовані для легкого управління та моніторингу.

Публічні сабнети використовуються для ресурсів, які мають бути доступні з Інтернету, тоді як приватні сабнети захищені від зовнішнього доступу та використовуються для зберігання даних і запуску обчислювальних ресурсів.

- розділення на публічні і приватні сабнети забезпечує багаторівневу архітектуру та підвищує рівень безпеки;
- всі ресурси, що потребують підключення до Інтернету (наприклад, веб-сервери), розміщені в публічних сабнетах, тоді як бази даних та інші чутливі ресурси розміщуються у приватних сабнетах;
- публічні сабнети мають CIDR-блоки, прив'язані до інтернет-шлюзу для доступу до Інтернету;
- приватні сабнети зв’язані з NAT-шлюзом для виходу в Інтернет, але залишаються недоступними ззовні.

Інтернет-шлюз (Internet Gateway) забезпечує публічний доступ до Інтернету для ресурсів, розміщених у публічних сабнетах.

- дозволяє ресурсам, таким як веб-сервери або інші сервіси, бути доступними для користувачів через Інтернет;
- надає можливість виходу ресурсів на зовнішні API або служби, необхідні для функціонування додатка;

— інтернет-шлюз прикріплений до VPC та асоційований з публічними таблицями маршрутизації, що дозволяє підключення до Інтернету.

NAT-шлюз (Network Address Translation Gateway) забезпечує доступ до Інтернету для ресурсів, розміщених у приватних сабнетах, без відкриття їх для вхідного трафіку.

— підвищує безпеку, дозволяючи приватним ресурсам здійснювати вихідні підключення (наприклад, для отримання оновлень або доступу до API), не будучи видимими з Інтернету;
— у кожному публічному сабнеті налаштовані окремі NAT-шлюзи, щоб забезпечити високу доступність та надійність;
— NAT-шлюз прив'язаний до EIP (Elastic IP), що забезпечує стабільну зовнішню IP-адресу;
— таблиці маршрутизації приватних сабнетів налаштовані для направлення трафіку до NAT-шлюзу при доступі до зовнішніх мереж.

Таблиці маршрутизації (Route Tables) визначають правила маршрутизації трафіку для кожного сабнету в VPC.

Публічні таблиці маршрутизації налаштовані для підключення до Інтернет-шлюзу, що дозволяє публічним сабнетам отримувати вхідний та вихідний трафік.

Приватні таблиці маршрутизації спрямовують вихідний трафік до NAT-шлюзу, забезпечуючи безпечний вихід до Інтернету для ресурсів в приватних сабнетах.

Окремі таблиці маршрутизації створені для кожного типу сабнетів (публічних та приватних), що дозволяє ефективно контролювати маршрутизацію для різних частин інфраструктури.

У публічних таблицях маршрутизації налаштовані правила для підключення до Інтернет-шлюзу, а в приватних — для NAT-шлюзу.

Ця мережева інфраструктура забезпечує розділення ресурсів на публічні та приватні сегменти з відповідними рівнями доступу, що підвищує загальну безпеку архітектури. Використання VPC у поєднанні з сабнетами, шлюзами, таблицями маршрутизації та правилами безпеки дозволяє створити гнучке, безпечне та масштабоване середовище для розміщення додатків.

Обчислювальні ресурси. Для забезпечення продуктивності та надійності додатка були ідентифіковані наступні обчислювальні ресурси в AWS:

EC2 (Elastic Compute Cloud) або Fargate сервара надають віртуальні сервери для запуску додатків із можливістю вибору різних типів інстансів для оптимізації продуктивності. Однак, для багатьох сучасних додатків Fargate стає більш привабливим завдяки своїй архітектурі без серверів (serverless), що дозволяє запускати контейнери без необхідності управління основною інфраструктурою.

— EC2 підтримує автоматичне масштабування, адміністратори повинні вручну налаштовувати типи інстансів, конфігурувати AMI (Amazon Machine Images), і підтримувати операційну систему;
— Fargate знімає необхідність управління інстансами. Автоматично масштабує контейнери відповідно до потреб робочих навантажень, без додаткових налаштувань чи турбот про безпеку операційної системи.

EC2 може бути економічно вигідним для постійних та передбачуваних навантажень. Проте, Fargate дозволяє оплачувати лише за ресурси, які дійсно використовуються контейнерами (плата за секунди використання CPU та пам'яті), що робить його значно вигіднішим для динамічних або змінних навантажень.

— EC2: потребує детальної конфігурації, такої як вибір типів інстансів, налаштування масштабування та управління мережею;

— Fargate: мінімізує потребу в налаштуваннях. Розробники зосереджуються на створенні контейнерів та завантаженні їх у Fargate, тоді як AWS автоматично обробляє всю інфраструктуру.

Fargate пропонує вищий рівень безпеки, оскільки контейнери ізольовані на рівні ядра, без спільного доступу до інфраструктури між клієнтами. На відміну від EC2, Fargate забезпечує автоматичне оновлення та виправлення операційної системи, що зменшує ризик вразливостей.

Fargate є ідеальним вибором для компаній, які прагнуть мінімізувати витрати на управління інфраструктурою, збільшити автоматизацію, і використовувати переваги контейнерних додатків без ускладнень, пов'язаних із EC2. Хоча EC2 залишається важливим для гнучких обчислювальних середовищ, Fargate дозволяє значно спростити роботу, підвищуючи ефективність та безпеку.

ECS (Elastic Container Service) використовується для керування контейнерами Docker, що дозволяє ефективно розгортати і масштабувати мікросервісну архітектуру.

Підтримка контейнеризації дозволяє розгортати мікросервіси у вигляді Docker-контейнерів, що підвищує ізоляцію та портативність додатків.

Сервіс забезпечує автоматичне масштабування контейнерів і полегшує керування ресурсами, знижуючи витрати та оптимізуючи використання обчислювальних потужностей.

Для ECS був обраний режим Fargate, що дозволяє не турбуватися про управління серверами та забезпечує гнучке масштабування контейнерів в залежності від навантаження. ECS кластери розгорнуті в приватних сабнетах для забезпечення безпеки та захисту додатків від зовнішнього доступу, з доступом до Інтернету через NAT-шлюз.

Auto Scaling Group (ASG) використовується для динамічного масштабування EC2 інстансів на основі завантаження додатку.

Дозволяє автоматично додавати або видаляти EC2 інстанси в залежності від змін у навантаженні, забезпечуючи стійкість до пікових навантажень і оптимізацію витрат.

Забезпечує високу доступність системи, автоматично перезапускаючи інстанси у випадку збоїв. ASG налаштована з параметрами мінімальної, максимальної та бажаної кількості інстансів для підтримки оптимального рівня продуктивності.

Політики масштабування налаштовані на основі ключових метрик, таких як завантаженість процесора (CPU utilization), для забезпечення своєчасної реакції на зміну навантаження.

Elastic Load Balancer (ELB) розподіляє вхідний трафік між EC2 інстансами або контейнерами, забезпечуючи рівномірне навантаження та балансування трафіку.

Підвищує доступність і продуктивність, автоматично направляючи запити на здорові інстанси та обходячи збої. Підтримує як внутрішній, так і зовнішній баланс трафіку, що дозволяє розподіляти запити між обчислювальними ресурсами як зсередини мережі, так і з Інтернету.

Налаштований Application Load Balancer (ALB) для розподілу HTTP та HTTPS трафіку, забезпечуючи балансування навантаження для веб-додатків. ELB інтегрований з Auto Scaling Group, що дозволяє автоматично додавати нові інстанси в балансування трафіку.

Ці обчислювальні ресурси в сукупності забезпечують гнучку, продуктивну та надійну інфраструктуру для розгортання додатків в AWS. Використання EC2 для обчислювальних потреб, ECS для контейнерів, Auto Scaling Group для динамічного масштабування, та ELB для розподілу навантаження забезпечує ефективність, гнучкість і високу доступність, що критично важливо для підтримки масштабованих аутсорсингових додатків.

База даних. Для зберігання даних додатку та забезпечення їхньої доступності та надійності, було обрано наступні ресурси для роботи з базою даних:

Amazon RDS (Relational Database Service) використовується для розгортання і управління реляційною базою даних PostgreSQL, яка служить основним сховищем даних додатку.

Amazon RDS автоматизує багато задач адміністрування, таких як резервне копіювання, оновлення програмного забезпечення та управління зберіганням. Підтримка реплікації та автоматичне відновлення забезпечують високу доступність та стійкість до збоїв.

Надає можливість налаштовувати продуктивність бази даних, вибираючи відповідний клас інстансів і обсяг пам’яті в залежності від навантаження додатку.

Вибраний інстанс db.t3.micro для тестових середовищ, що забезпечує економічне використання ресурсів при невеликих навантаженнях. Для більш масштабних середовищ можуть використовуватись інстанси з більшою потужністю, наприклад db.m5.large. Простір зберігання налаштований з параметром `allocated_storage = 20`, що дозволяє забезпечити достатній об'єм для початкового розгортання.

Включено автоматичне резервне копіювання та точку відновлення, що дозволяє швидко відновити дані у випадку збоїв. База даних розміщується в приватних сабнетах для забезпечення безпеки, і доступ до неї обмежений за допомогою правил безпеки (Security Group), які дозволяють доступ тільки з обчислювальних ресурсів, що знаходяться у тій же VPC.

DB Subnet Group визначає сабнети в різних зонах доступності (Availability Zones), в яких буде розгорнута база даних RDS.

Розміщення бази даних у приватних сабнетах забезпечує додатковий рівень безпеки, оскільки база даних не має прямого доступу з Інтернету. Включення кількох зон доступності підвищує доступність та відмовостійкість бази даних.

Сабнет-група складається з приватних сабнетів у різних зонах доступності (`eu-central-1a` і `eu-central-1b`), що дозволяє зменшити ризик

збоїв та підвищити надійність за рахунок автоматичного перемикання на іншу зону у випадку збоїв.

Security Group (група безпеки) контролює мережевий доступ до бази даних, дозволяючи лише специфічні з'єднання.

Забезпечує рівень контролю доступу, дозволяючи лише трафік із заданих CIDR-блоків або інших Security Groups, що підвищує безпеку. Дозволяє встановлювати правила для дозволу або блокування трафіку, зменшуючи можливість несанкціонованого доступу.

Налаштовано доступ на порт 5432 (стандартний порт PostgreSQL) для з'єднань із приватних сабнетів, що забезпечує безпеку внутрішньої комунікації між EC2 або ECS інстансами та базою даних. Вихідний трафік дозволений на всі порти та IP-адреси для забезпечення необхідного з'єднання з іншими службами AWS.

Використання Amazon RDS для управління реляційною базою даних PostgreSQL забезпечує надійність, безпеку та масштабованість бази даних. DB Subnet Group і Security Group надають додатковий рівень безпеки, розміщуючи базу даних у приватних сабнетах і обмежуючи доступ. Така конфігурація дозволяє забезпечити ефективне зберігання даних з мінімальними вимогами до обслуговування, що є критично важливим для аутсорсингових додатків, які повинні бути завжди доступними та захищеними.

Сховище об’єктів. Для зберігання статичних ресурсів додатку, таких як файли зображень, відео, або інші медіафайли, які потрібно швидко доставляти користувачам, було обрано об'єктне сховище Amazon S3. Нижче наведені ресурси, налаштовані для реалізації сховища об’єктів.

S3 bucket використовується для зберігання статичного контенту додатку та може бути налаштований для використання як статичний веб-сайт.

Низька вартість зберігання та висока надійність, що робить S3 оптимальним рішенням для зберігання великих обсягів статичних файлів. Інтеграція з іншими сервісами AWS, такими як CloudFront для покращення доставки контенту та IAM для контролю доступу.

Створено S3 bucket з назвою `frontshop-bucket`, де зберігається основний статичний контент. Додано теги для визначення середовища (`Environment = "Dev"`) та зручності в управлінні ресурсами. Для забезпечення доступності ресурсів з браузера користувача налаштовано bucket як статичний веб-сайт, з налаштуванням `index.html` як початкової сторінки та сторінки для обробки помилок.

Bucket Policy використовується для керування доступом до об’єктів у сховищі.

Дає можливість точно налаштувати доступ до bucket, що підвищує безпеку та гнучкість. Політика дозволяє відкритий доступ до об’єктів у S3 bucket, необхідний для загальнодоступного доступу до статичних ресурсів додатку.

Налаштовано bucket policy, який дозволяє анонімний доступ (з правом `s3:*` для всіх об’єктів у bucket), що необхідно для публічного статичного веб-контенту. Політика налаштована для доступу на основі ARN bucket, що дозволяє точково контролювати доступ до ресурсів.

Bucket Ownership Controls визначення налаштувань володіння об’єктами у S3 bucket, щоб забезпечити контроль власника bucket над всіма об’єктами.

Дозволяє bucket-власнику мати повний контроль над всіма об’єктами, незалежно від того, хто їх завантажує. Це особливо важливо для проектів, де об'єкти можуть бути завантажені різними обліковими записами.

Встановлено режим `BucketOwnerPreferred`, щоб усі об'єкти, завантажені в bucket, автоматично ставали власністю bucket-власника.

Bucket ACL (Access Control List) дозволяє налаштовувати рівень доступу до bucket.

Надає базовий рівень доступу, дозволяючи зробити bucket загальнодоступним для зчитування, що необхідно для публічного статичного контенту.

Встановлено ACL типу `public-read`, що дозволяє публічний доступ для зчитування об’єктів у bucket. Для захисту bucket від несанкціонованих змін встановлено додаткові налаштування у public access block.

Bucket Public Access Block цей ресурс дозволяє обмежити або дозволити публічний доступ до bucket.

Дозволяє контролювати доступ до bucket, дозволяючи публічний доступ лише для зчитування. Додатковий захисний шар, щоб обмежити випадковий доступ, але з дозволом на публічний статичний контент.

Налаштовано public access block, який дозволяє публічний доступ для зчитування (`block_public_acls = false`, `block_public_policy = false`).

AWS KMS Key використовується для забезпечення шифрування об’єктів у S3 bucket, що підвищує захист даних.

Дозволяє шифрувати об’єкти в S3 для додаткової безпеки. AWS KMS інтегрується з іншими сервісами для легкого керування ключами та шифрування.

Налаштовано ключ KMS з описом та вікном видалення (10 днів), що дозволяє автоматичне шифрування даних у bucket при збереженні об'єктів.

Використання Amazon S3 для об’єктного сховища є оптимальним рішенням для зберігання статичного контенту додатку. Налаштування bucket як статичного веб-сайту забезпечує швидкий та легкий доступ до файлів користувачами. Додаткові налаштування bucket policy, ownership controls, та KMS шифрування підвищують безпеку та керованість об'єктного сховища.

Безпека та контроль доступу. Для забезпечення безпеки та контролю доступу в архітектурі використовуються ресурси та сервіси AWS, які захищають обчислювальні, мережеві й зберігальні компоненти інфраструктури. Наведені нижче ресурси дозволяють контролювати доступ до критично важливих сервісів та гарантують, що лише авторизовані користувачі або сервіси можуть отримати до них доступ.

Security Groups використовуються для контролю вхідного та вихідного трафіку до обчислювальних ресурсів і баз даних, дозволяючи визначити, які порти можуть бути доступними.

Створено два security groups:

— db_security_group: обмежує доступ до бази даних через порт 5432 лише з приватних підмереж, захищаючи її від доступу ззовні;

— ecs_security_group: дозволяє доступ до обчислювальних ресурсів через порти 22 (SSH), 80 (HTTP), 443 (HTTPS), та 5050 (додатковий порт для специфічних сервісів), забезпечуючи доступ до ECS-контейнерів.

IAM (Identity and Access Management) використовується для управління правами доступу та ідентифікацією користувачів і сервісів, що дозволяє визначити, хто і які дії може виконувати з ресурсами AWS.

Надійний контроль доступу до ресурсів на основі ролей, що дозволяє використовувати принцип мінімальних прав доступу. Можливість призначати окремі права для різних користувачів і сервісів.

Визначено IAM ролі для ECS, EC2 та інших сервісів, що забезпечують доступ до необхідних ресурсів з мінімально необхідними правами. Додано політики для дозволу певних дій, таких як доступ до S3 bucket для зберігання, обмежуючи права на основі принципу "least privilege" (мінімально необхідних прав).

AWS KMS (Key Management Service) використовується для шифрування даних, збережених у різних сервісах, таких як S3 і RDS, забезпечуючи додатковий рівень безпеки.

Захищає конфіденційні дані за допомогою шифрування на стороні сервера. Інтеграція з іншими сервісами AWS для легкого керування ключами та контролю доступу до них.

Налаштовано KMS-ключ для шифрування об'єктів у S3 bucket, що забезпечує захист даних при їх зберіганні. Ключ доступний лише для

авторизованих IAM користувачів та сервісів, що дозволяє мінімізувати ризик витоку даних.

Використання S3 Bucket Policies та ACL (Access Control List) дозволяє встановити права доступу до S3 bucket на рівні об’єктів та bucket.

Гнучке налаштування доступу до окремих об’єктів та bucket, що дозволяє визначити правила на рівні ресурсу. Bucket policy використовується для контролю доступу до об’єктів, які повинні бути загальнодоступними (наприклад, статичний веб-контент), забезпечуючи при цьому захист від несанкціонованих дій.

Додано bucket policy для забезпечення публічного доступу лише для зчитування (необхідного для статичного веб-контенту), а також використання ACL з дозволом `public-read` для S3 bucket. Заблоковано модифікацію bucket для анонімних користувачів, залишаючи можливість лише зчитування.

Public Access Block на S3 Bucket дозволяє контролювати публічний доступ до bucket, обмежуючи доступ тільки до тих об’єктів, які явно мають дозволи на публічний доступ.

Знижує ризик випадкового відкриття доступу до bucket, обмежуючи права доступу та забезпечуючи захист приватних даних.

Увімкнено `block_public_acls` та `block_public_policy` для обмеження випадкового надання публічного доступу до конфіденційних об’єктів у S3 bucket.

Забезпечення безпеки та контролю доступу є критично важливим аспектом інфраструктури. Security Groups, IAM ролі та політики, шифрування за допомогою KMS, а також bucket policies для S3 забезпечують захист даних, контроль над ресурсами та управління правами доступу. Такий підхід до безпеки гарантує, що лише авторизовані користувачі та сервіси можуть отримати доступ до критично важливих даних та ресурсів інфраструктури.[8]

2.4 Аналіз взаємодії між компонентами архітектури

Аналіз взаємодії між компонентами архітектури є критично важливим для забезпечення сумісності, надійності та стабільності всієї системи. У даній архітектурі, побудованій на основі AWS та Terraform, всі компоненти спроектовані так, щоб вони ефективно працювали разом, доповнюючи один одного. Нижче наведено детальний опис взаємодії ключових компонентів і їх сумісності.

VPC забезпечує ізольоване середовище для обчислювальних ресурсів, зокрема EC2 інстансів і контейнерів ECS. Публічні підмережі надають EC2 інстансам доступ до Інтернету через інтернет-шлюз, а приватні підмережі захищають критичні компоненти, такі як бази даних, шляхом обмеження їхнього доступу до зовнішньої мережі.

Нат-шлюз забезпечує вихідний Інтернет-трафік для приватних ресурсів без необхідності відкриття публічних доступів, що сприяє підтримці стабільності та безпеки мережевої інфраструктури.

Fargate інстанси та ECS-контейнери можуть отримувати доступ до бази даних, що розміщена у приватних підмережах, через конфігуровані security groups. Це забезпечує захищений доступ до бази даних, обмежуючи її видимість лише для необхідних компонентів.

ECS та Fargate інстанси взаємодіють через мережу, а також використовують IAM ролі для доступу до ресурсів, таких як S3 bucket, забезпечуючи безперебійну роботу сервісів. Ця сумісність забезпечує, що всі компоненти архітектури можуть ефективно комунікувати один з одним, використовуючи лише необхідні дозволи.

База даних (наприклад, RDS) розміщується у приватних підмережах для додаткового захисту, а обчислювальні ресурси мають доступ до неї через security group, що дозволяє підключення тільки з обмежених IP-адрес (публічних або приватних підмереж, де розміщено EC2 та ECS).

Така взаємодія дозволяє мінімізувати ризик несанкціонованого доступу до бази даних, забезпечуючи стабільність та безпеку роботи додатка.

Взаємодія з обчислювальними ресурсами та мережею, S3 bucket використовується для зберігання статичного контенту (наприклад, зображень або статичних файлів для веб-інтерфейсу) та налаштований як статичний веб-сайт. Обчислювальні ресурси мають доступ до S3 через IAM ролі, що дозволяє зчитування та запис контенту безпосередньо з S3.

Використання політики bucket, а також шифрування через KMS дозволяють забезпечити сумісність та захист доступу до об’єктів у S3 bucket. Це важливо для підтримання стабільної роботи, оскільки дані захищені, а доступ можливий лише для дозволених користувачів та сервісів.

Security Groups обмежують доступ до EC2, RDS та інших компонентів, дозволяючи тільки визначені IP-адреси та порти. IAM ролі та політики використовуються для надання EC2 та ECS необхідних дозволів на доступ до S3, KMS та інших сервісів.

Правильне налаштування Security Groups та IAM ролей дозволяє забезпечити надійну взаємодію між компонентами, запобігаючи конфліктам та забезпечуючи стабільну роботу інфраструктури. Сервіси взаємодіють між собою через захищені канали доступу, що знижує ризик безпекових загроз.

KMS забезпечує шифрування для S3 bucket та інших ресурсів. Обчислювальні ресурси можуть дешифрувати дані лише за наявності відповідних IAM дозволів.

Використання KMS забезпечує безпеку чутливих даних, а сумісність із S3 та іншими сервісами дозволяє зберігати дані в зашифрованому вигляді, не порушуючи доступність і цілісність.

Завдяки тісній інтеграції та ретельно продуманій конфігурації мережевих, обчислювальних, зберігальних і безпекових компонентів архітектура забезпечує сумісність і стабільну роботу. AWS сервіси, зокрема VPC, ECS, S3 та IAM, працюють разом як єдина система, забезпечуючи надійну, масштабовану та безпечну інфраструктуру. Це сприяє ефективній

роботі додатку, знижує ризики безпеки та гарантує стабільну роботу всіх компонентів у межах архітектури.[10]

2.5 Огляд готового коду в GitHub та основних функцій кожного модуля

Модуль Cloud Front. Цей модуль створює інфраструктуру для CloudFront із підтримкою сертифікатів SSL/TLS для доменів, що забезпечує захищене підключення для статичних ресурсів, збережених у S3.

Ось короткий опис основних функцій коду:

Головний main.tf у Terraform об'єднує всі модулі в єдину архітектуру, яка автоматизує створення та налаштування інфраструктури для додатку. Він виконує наступні функції:

1. Організація інфраструктури через модулі:
   - створює мережу (VPC, підмережі, групи безпеки);
   - розгортає базу даних (PostgreSQL через RDS);
   - налаштовує зберігання файлів (S3, KMS);
   - запускає кластер контейнерів (ECS);
   - забезпечує балансування навантаження (ALB);
   - організовує глобальну доставку контенту (CloudFront);
   - реєструє DNS-зону для домену frontshop.tech.
2. Інтеграція компонентів:
   - модулі взаємодіють через передавання ресурсів між ними (наприклад, підмережі з VPC використовуються для RDS та ECS);
   - використовує залежності між модулями (depends_on), щоб ресурси створювалися в правильному порядку.
3. Як працює вся система:
   - користувачі звертаються до домену через Route 53;

— CloudFront забезпечує кешування та швидку доставку статичного контенту з S3;
— динамічні запити перенаправляються на ECS через ALB;
— ECS запускає контейнери додатку, які можуть взаємодіяти з RDS для роботи з базою даних;
— логи та моніторинг виконуються через CloudWatch.

```
module "front-shop-vpc" {
    source              = "./modules/vpc"
    env                 = "front-shop"
    vpc_cidr            = var.vpc_cidr
    public_subnet_cidr-a  = var.public_subnet_cidr-a
    private_subnet_cidr-a = var.private_subnet_cidr-a
    public_subnet_cidr-b  = var.public_subnet_cidr-b
    private_subnet_cidr-b = var.private_subnet_cidr-b

}

module "postgresql" {
    depends_on = [ module.front-shop-vpc ]
    source              = "./modules/rds"
    vpc_id_db           = module.front-shop-vpc.vpc_id
    db_security_group   = module.front-shop-vpc.db_security_group_id
    # db_private_subnet   = module.front-shop-vpc.private_subnet_id
    db_private_subnet_a  = module.front-shop-vpc.private_subnet_id_a
    db_private_subnet_b = module.front-shop-vpc.private_subnet_id_b
}

module "storage"{
    source = "./modules/storage"
    environment = "frontshop"
}


module "frontshop-ecr" {
    depends_on = [ module.front-shop-vpc ]
    source              = "./modules/ecr"
    alb_security_group  = module.front-shop-vpc.ecs_security_group_id
    alb_public_subnet_a  = module.front-shop-vpc.public_subnet_id_a
    alb_public_subnet_b = module.front-shop-vpc.public_subnet_id_b
    vpc_id_alb          = module.front-shop-vpc.vpc_id
    cert_arn = module.cloudfront.cert_arn
    s3_bucket = module.storage.s3_bucket
}
```

Рисунок 2.1 – Головний код для створення архітектури

Головний файл організовує взаємодію всіх компонентів, створюючи масштабовану, безпечну та продуктивну архітектуру для підтримки веб-додатка на AWS.

Сертифікація SSL/TLS для домену. aws_acm_certificate.listener_cert та aws_acm_certificate.cert — ресурси для створення SSL-сертифікатів для доменів `api.frontshop.tech` та `frontshop.tech` з використанням AWS ACM (Certificate Manager) для захищеного доступу.

Такі значення як aws_route53_record.listener_cert_validation та aws_route53_record.cert_validation — створюють записи в DNS для підтвердження домену через Route 53.

Також використовуємо aws_acm_certificate_validation.listener_cert та aws_acm_certificate_validation.cert — підтверджують SSL-сертифікати після створення DNS-записів.

```
resource "aws_acm_certificate" "listener_cert" {
  domain_name       = "api.frontshop.tech"
  validation_method = "DNS"

  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_route53_record" "listener_cert_validation" {
  name = tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_name
  type    = tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_type
  zone_id = var.aws_route53_zone
  records = [ tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_value ]

  ttl = 60
}

resource "aws_acm_certificate_validation" "listener_cert" {
  certificate_arn         = aws_acm_certificate.listener_cert.arn
  validation_record_fqdns = [aws_route53_record.listener_cert_validation.fqdn]
}
```

Рисунок 2.2 – Створення сертифікатів для додатку

Amazon Web Services (CloudFront) для доставки контенту. aws_cloudfront_origin_access_control.origin_access — налаштовує контроль доступу для CloudFront, щоб захистити доступ до S3-бакету з CloudFront.

aws_cloudfront_distribution.cloudfront — створює CloudFront дистрибуцію, що використовується для кешування та доставки контенту.

origin — вказує на S3-бакет, який зберігає контент для доставки.

default_cache_behavior — налаштовує методи доступу, політики кешування та перенаправлення запитів на HTTPS.

viewer_certificate — використовує SSL-сертифікат для забезпечення захищеного підключення.

custom_error_response — встановлює обробку помилок для 403 та 404, перенаправляючи їх на `/index.html`.

```
resource "aws_cloudfront_distribution" "cloudfront" {
  origin {
    domain_name = var.s3_domain_name
    origin_access_control_id = aws_cloudfront_origin_access_control.origin_access.id
    origin_id = local.s3_origin_id
  }

  enabled = true
  is_ipv6_enabled = true
  default_root_object = "index.html"

  logging_config {
    include_cookies = false
    bucket          = "frontshop-bucket.s3.amazonaws.com"
    prefix          = "cf"
  }

  aliases = ["frontshop.tech"]

  restrictions {
    geo_restriction {
      restriction_type = "none"
      locations = []
    }
  }
```

Рисунок 2.3 – Використання сервісу для доставки контенту

Політики кешування та запитів в Amazon Web Services. data.aws_cloudfront_cache_policy.cache_policy — політика кешування для CloudFront, яка оптимізує кешування.

data.aws_cloudfront_origin_request_policy.origin_request_policy_id — політика для управління запитами до S3, зокрема для крос-доменних запитів (CORS).

```
default_cache_behavior {
  allowed_methods = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT" ]
  cached_methods = ["GET", "HEAD"]
  target_origin_id = local.s3_origin_id
  compress         = true
  # forwarded_values {
  #   query_string = false

  #   cookies {
  #     forward = "all"
  #   }
  # }

  origin_request_policy_id = data.aws_cloudfront_origin_request_policy.origin_request_policy_id.id
  cache_policy_id = data.aws_cloudfront_cache_policy.cache_policy.id

  viewer_protocol_policy = "redirect-to-https"
}
```

Рисунок 2.4 – Встановлення політик кешування та запитів

DNS-запис для CloudFront. aws_route53_record.records — додає запис у Route 53, щоб прив'язати домен `frontshop.tech` до CloudFront, що дозволяє направляти трафік на CloudFront дистрибуцію.

```
resource "aws_route53_record" "listener_cert_validation" {
  name = tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_name
  type    = tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_type
  zone_id = var.aws_route53_zone
  records = [ tolist(aws_acm_certificate.listener_cert.domain_validation_options)[0].resource_record_value ]

  ttl = 60
}

resource "aws_acm_certificate_validation" "listener_cert" {
  certificate_arn         = aws_acm_certificate.listener_cert.arn
  validation_record_fqdns = [aws_route53_record.listener_cert_validation.fqdn]
}


resource "aws_acm_certificate" "cert" {
  provider = aws.acm
  domain_name       = "frontshop.tech"
  validation_method = "DNS"

  lifecycle {
    create_before_destroy = true
  }
}
```

Рисунок 2.5 – Створення доменних записів

Цей модуль забезпечує захищену доставку контенту з S3 за допомогою CloudFront, використовуючи SSL-сертифікати для доменів, що підвищує безпеку та продуктивність веб-додатку.

Модуль ECR. Цей модуль забезпечує налаштування інфраструктури для запуску Amazon ECS із використанням Elastic Load Balancer (ELB) та Elastic Container Registry (ECR) для управління контейнерами. Основні функції коду:

Реєстр контейнерів (ECR). aws_ecr_repository.frontshop-ecr — створює репозиторій ECR з назвою `frontshop-ecr`. Цей репозиторій використовується для зберігання Docker-образів додатку. Опція `scan_on_push = true` забезпечує автоматичне сканування образів на вразливості під час завантаження.

```
resource "aws_ecr_repository" "frontshop-ecr" {
  name                 = "frontshop-ecr"
  image_tag_mutability = "MUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }
}
```

Рисунок 2.6 – Використання сервісу для зберігання контейнерів

Application Load Balancer (ALB). aws_lb.front_shop_lb — створює ALB, який відповідає за розподіл трафіку між екземплярами ECS. Він налаштований як публічний (`internal = false`) і розміщений у зазначених публічних підмережах (`subnets`), із застосуванням певних правил безпеки (`security_groups`).

access_logs — логування доступу зберігається в S3-бакеті, з префіксом `"lb"`, що спрощує моніторинг та аналіз доступу.

```
resource "aws_lb" "front_shop_lb" {
  name               = "front-shop-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [var.alb_security_group]
  subnets            = [var.alb_public_subnet_a, var.alb_public_subnet_b]
  access_logs {
    bucket  = var.s3_bucket
    prefix  = "lb"
    enabled = true
  }
}
```

Рисунок 2.7 – Створення балансувальника навантаження

Група цілей для ALB. aws_lb_target_group.front_shop_target_group — створює групу цілей для ALB, яка вказує на екземпляри додатку, які працюють

на порту 5050. Група цілей забезпечує рівномірний розподіл запитів між екземплярами.

health_check — встановлює шлях `/health` для перевірки стану цілей, що дозволяє ALB визначати, чи екземпляр додатку працює належним чином.

```
resource "aws_lb_target_group" "front_shop_target_group" {
  name        = "front-shop-target-group"
  port        = 5050 # Порт, на який направляє ALB
  protocol    = "HTTP"
  vpc_id      = var.vpc_id_alb
  target_type = "instance"

    health_check {
    path     = "/health" # Додайте шлях для перевірки стану
    port     = "5050"
    protocol = "HTTP"
  }
}
```

Рисунок 2.8 – Підключення ресурсів до балансувальника

Route 53 DNS-зона. aws_route53_zone.primary — створює DNS-зону для домену `frontshop.tech`, яка використовується для налаштування DNS-записів і маршрутизації трафіку на ALB.

```
resource "aws_route53_zone" "primary" {
    name                = "frontshop.tech"
}
```

Рисунок 2.9 – Створення DNS зони для доменного імені

Слухачі ALB. aws_lb_listener.front_shop_listener — налаштовує слухач на порту 443 з підтримкою HTTPS, використовуючи SSL-сертифікат (`certificate_arn`). Усі запити, що надходять через HTTPS, перенаправляються на групу цілей.

aws_lb_listener.front_shop_listener_80 — налаштовує слухач на порту 80 (HTTP), який автоматично перенаправляє всі HTTP-запити на HTTPS (порт 443), забезпечуючи шифрування з'єднання.

```
resource "aws_lb_listener" "front_shop_listener" {
  load_balancer_arn = aws_lb.front_shop_lb.arn
  port              = 443
  protocol          = "HTTPS"
  certificate_arn = var.cert_arn
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.front_shop_target_group.arn
  }
}

resource "aws_lb_listener" "front_shop_listener_80" {
  load_balancer_arn = aws_lb.front_shop_lb.arn
  port              = 80
  protocol          = "HTTP"
  default_action {
    type = "redirect"
    redirect {
      port        = "443"
      protocol    = "HTTPS"
      status_code = "HTTP_301"
    }
  }
```

Рисунок 2.10 – Налаштування портів на HTTPS

Цей модуль забезпечує створення надійної та безпечної інфраструктури для запуску контейнерних додатків із розподілом навантаження через ALB, SSL-зашифрованим з'єднанням, і автоматичною перевіркою стану додатку.

Модуль ECS. Забезпечує розгортання контейнерів для сервісу через кластер ECS, використовуючи EC2-екземпляри. Нижче наведено основні компоненти та їх функціональні ролі в інфраструктурі:

Роль IAM для ECS. Створюється IAM роль `ecs-role`, яка дозволяє екземплярам EC2 і задачам ECS використовувати потрібні ресурси.

Додаткова політика `kms_access` надає доступ до KMS для розшифровки конфіденційних даних із AWS Secrets Manager.

```
resource "aws_iam_role" "ecs_role" {
  name = "ecs-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect    = "Allow",
      Principal = {
        Service = ["ecs-tasks.amazonaws.com", "ec2.amazonaws.com"]
      },
      Action    = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role_policy" "kms_access" {
  name = "kms-access"
  role = aws_iam_role.ecs_role.name

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [{
      Effect   = "Allow",
      Action   = [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds",
        "kms:Decrypt",
      ],
      Resource = ["arn:aws:secretsmanager:eu-central-1:165387232810:secret:*", "arn:aws:kms:eu-central-1:165387232810:key/7e1d7202-6369-402f-afb3-8f3948324507"],
    }]
  })
}
```

Рисунок 2.11 – Створення ролі для використання ECS

Профіль екземпляра IAM. Цей профіль асоціює IAM роль з екземплярами Fargate, що дозволяє їм взаємодіяти з ECS та отримувати доступ до ресурсів.

```
resource "aws_iam_instance_profile" "ecs_instance_profile" {
  name = "ecs-instance-profile"
  role = aws_iam_role.ecs_role.name
}
```

Рисунок 2.12 – Створення ролі для використання Fargate

Кластер ECS (aws_ecs_cluster). Створює кластер `front-shop-cluster`, де будуть виконуватись завдання ECS. Це основне середовище для запуску контейнерів у сервісі ECS.

```
resource "aws_ecs_cluster" "front_shop_cluster" {
  name = "front-shop-cluster"
}
```

Рисунок 2.13 – Створення кластеру EKS

Сервіс ECS (aws_ecs_service). Запускає ECS-сервіс `front-shop-service` з одним контейнером. Використовує балансувальник навантаження, який маршрутизує запити на потрібний контейнер, працюючи на порту 5050. Налаштований для використання EC2 як типу запуску.

```
resource "aws_ecs_service" "front_shop_ecs_service" {
  name            = "front-shop-service"
  cluster         = aws_ecs_cluster.front_shop_cluster.id
  task_definition = aws_ecs_task_definition.front_shop_task_definition.arn
  launch_type     = "EC2"
  desired_count   = 1

  load_balancer {
    target_group_arn = var.target_group
    container_name   = "front-shop" # Замініть на ім'я вашого контейнера
    container_port   = 5050 # Порт, на якому працює ваш контейнер
  }

}
```

Рисунок 2.14 – Використання сервісу ECS

Secrets Manager та KMS для секретів. Конфіденційні дані (наприклад, з'єднання з базою даних) зберігаються у AWS Secrets Manager і захищені за допомогою KMS. Це забезпечує безпечне управління секретами для контейнерів ECS, використовуючи `aws_secretsmanager_secret` та `aws_secretsmanager_secret_version`.

```
resource "aws_secretsmanager_secret" "api-env" {
  name = "api-env-1"
  kms_key_id = var.key_id
}


resource "aws_secretsmanager_secret_version" "db-env" {
  secret_id     = aws_secretsmanager_secret.api-env.id
  secret_string = jsonencode(yamldecode(data.aws_kms_secrets.api_secrets.plaintext["db"]))
}
```

Рисунок 2.15 – Створення сервісу для передачі чутливих даних

Завдання ECS (aws_ecs_task_definition). Визначає завдання `front-shop-task`, що містить конфігурацію контейнера, ресурси CPU та пам'яті, порт для роботи контейнера, а також конфігурацію для логування через CloudWatch. Контейнер отримує доступ до секретів через `secrets` параметр.

```
resource "aws_ecs_task_definition" "front_shop_task_definition" {
  family                   = "front-shop-task"
  network_mode             = "bridge"
  requires_compatibilities = ["EC2"]
  cpu                      = "512" # Замініть на необхідні ресурси
  memory                   = "1024" # Замініть на необхідні ресурси
```

Рисунок 2.16 – Завантаження задачі для сервісу ECS

Група логів CloudWatch (aws_cloudwatch_log_group). Логи з контейнерів зберігаються в лог-групі `/ecs/front-shop-log-group` з часом зберігання 7 днів, що дозволяє моніторинг та діагностику.

```
resource "aws_cloudwatch_log_group" "ecs_log_group" {
  name              = "/ecs/front-shop-log-group"
  retention_in_days = 7
}
```

Рисунок 2.17 – Підключення лог-групи для зберігання логів

Конфігурація запуску EC2 (aws_launch_configuration). Конфігурація `front-shop-launch-configuration` створює шаблон для екземплярів EC2 з образом AMI, типом екземпляра `t2.small`, налаштуванням безпеки та скриптом, який додає інстанси в ECS кластер.

```
resource "aws_launch_configuration" "front_shop_launch_configuration" {
  name                 = "front-shop-launch-configuration"
  image_id             = "ami-0029cbd17e0d653cf" #data.aws_ami.linux.id # Замініть на потрібний образ AMI
  instance_type        = "t2.small"
  iam_instance_profile = aws_iam_instance_profile.ecs_instance_profile.arn
  security_groups      = [var.ecs_security_group]
  key_name             = "testkey"
  user_data            = <<-EOF
                           #!/bin/bash
                           echo ECS_CLUSTER=front-shop-cluster >> /etc/ecs/ecs.config
                         EOF
  lifecycle {
    create_before_destroy = true
  }
```

Рисунок 2.18 – Конфігурація запуску серверів

Автоскейлінг-група (aws_autoscaling_group). Група `frontshop-asg` визначає мінімальну, максимальну та бажану кількість інстансів EC2 для обслуговування навантаження. Використовується для забезпечення доступності сервісу і автоматичного масштабування під навантаженням.

```
resource "aws_autoscaling_group" "front_shop_autoscaling_group" {
  name                 = "frontshop-asg"
  desired_capacity     = 1
  max_size             = 1
  min_size             = 1
  health_check_type    = "EC2"
  force_delete         = true
  launch_configuration = aws_launch_configuration.front_shop_launch_configuration.id
  vpc_zone_identifier  = [var.ecs_private_subnet_a, var.ecs_private_subnet_b]
}
```

Рисунок 2.19 – Підключення автоматичного масштабування серверів

Capacity Provider (aws_ecs_capacity_provider). Capacity Provider керує автоматичним масштабуванням, встановлюючи бажану кількість ресурсів для ECS завдань. Це дозволяє оптимально розподіляти ресурси у кластері для забезпечення продуктивності сервісу.

```
resource "aws_ecs_capacity_provider" "ecs_capacity_provider" {
 name = "test1"

 auto_scaling_group_provider {
   auto_scaling_group_arn = aws_autoscaling_group.front_shop_autoscaling_group.arn

   managed_scaling {
     maximum_scaling_step_size = 1000
     minimum_scaling_step_size = 1
     status                    = "ENABLED"
     target_capacity           = 3
   }
 }
}
```

Рисунок 2.20 – Встановлення кількості серверів

Кожен з цих ресурсів відповідає за частину інфраструктури, створюючи надійну та масштабовану архітектуру для контейнерних сервісів в AWS ECS.

Модуль RDS забезпечує створення бази даних PostgreSQL, яка зберігає дані додатку та доступна через приватні підмережі VPC для забезпечення безпеки. Ось основні ресурси та їх роль:

Створює екземпляр бази даних PostgreSQL з наступними параметрами:

— розмір пам'яті: 20 ГБ з використанням `gp2` (General Purpose SSD);

— движок: PostgreSQL версії 12;

— клас екземпляра: `db.t3.micro`, що забезпечує баланс між вартістю та продуктивністю;

— ідентифікатор бази даних, ім'я користувача та пароль: передаються через змінні (`var.db_name`, `var.db_username`, `var.db_password`), що дозволяє динамічно налаштовувати параметри під час розгортання;

— пропущення фінального знімка: `skip_final_snapshot = true` дозволяє уникнути створення знімка перед видаленням бази, що може бути зручним для тестових середовищ;

— база даних доступна лише через VPC за допомогою приватних підмереж і групи безпеки (`var.db_security_group`), яка обмежує доступ.

```
resource "aws_db_instance" "postgresql" {
  allocated_storage    = 20
  storage_type         = "gp2"
  engine               = "postgres"
  engine_version       = "12"
  instance_class       = "db.t3.micro"
  identifier           = var.db_name
  username             = var.db_username
  password             = var.db_password
  skip_final_snapshot  = true
  vpc_security_group_ids = [var.db_security_group]
  db_subnet_group_name = aws_db_subnet_group.db_subnet_group.name
  availability_zone = "eu-central-1b"
}
```

Рисунок 2.21 – Код створення бази даних

Група підмереж для бази даних (aws_db_subnet_group). Визначає групу підмереж, до якої буде підключений екземпляр бази даних. Це необхідно для забезпечення високої доступності та ізоляції бази даних від публічної мережі.

Підключається до приватних підмереж (`var.db_private_subnet_a` і `var.db_private_subnet_b`), що дозволяє розташовувати екземпляр бази даних в приватному сегменті VPC, підвищуючи безпеку інфраструктури.

Група підмереж позначається тегом `Name = "DB Subnet Group"` для зручного керування та ідентифікації.

```
resource "aws_db_subnet_group" "db_subnet_group" {
  name       = "db-subnet-group"
  # subnet_ids = [var.db_private_subnet, var.db_private_subnet1,
  subnet_ids = [var.db_private_subnet_a, var.db_private_subnet_b]
  tags = {
    Name = "DB Subnet Group"
  }
}
```

Рисунок 2.22 – Група підмереж для бази даних

Цей модуль забезпечує надійне та безпечне зберігання даних додатків в базі даних, доступній тільки з внутрішніх ресурсів в VPC, що мінімізує ризик несанкціонованого доступу.

Модуль Storage. Створює S3 бакет з такими налаштуваннями:

— bucket = "frontshop-bucket"`: задається ім’я бакета як `frontshop-bucket`;

— `tags`: додаються теги для ідентифікації середовища (`Environment = "Dev"`) і назви (`Name = "Front-shop bucket"`), що може допомогти при управлінні ресурсами.

```
resource "aws_s3_bucket" "frontshop-bucket" {
  bucket = "frontshop-bucket"
  tags = {
    Name        = "Front-shop bucket"
    Environment = "Dev"
  }
}
```

Рисунок 2.23 – Створення бакету для зберігання статичного контенту

Ресурс `aws_s3_bucket_ownership_controls` "controls". Контролює власність об’єктів у бакеті.

object_ownership = "BucketOwnerPreferred"`: дозволяє, щоб всі об’єкти автоматично належали власнику бакету, навіть якщо завантажені іншими користувачами.

```
resource "aws_s3_bucket_ownership_controls" "controls" {
  bucket = aws_s3_bucket.frontshop-bucket.id
  rule {
    object_ownership = "BucketOwnerPreferred"
  }
}
```

Рисунок 2.24 – Налаштування доступу до бакету

Ресурс `aws_s3_bucket_website_configuration` "front-shop-static-website". Налаштовує бакет для хостингу статичного вебсайту.

`index_document.suffix = "index.html"`: задається `index.html` як головний файл.

`error_document.key = "index.html"`: `index.html` також використовується як сторінка помилок (наприклад, для обробки всіх 404 запитів).

```
resource "aws_s3_bucket_website_configuration" "front-shop-static-website" {
  bucket = aws_s3_bucket.frontshop-bucket.id

  index_document {
    suffix = "index.html"
  }

  error_document {
    key = "index.html"
  }
}
```

Рисунок 2.25 – Налаштування бакету на роботу вебсторінки

Ресурс `aws_s3_bucket_policy` «hosting_bucket_policy». Додає політику доступу для хостингу вебсайту на S3

`policy`: дозволяє публічний доступ до всіх об’єктів у бакеті для всіх користувачів (`»Effect»: «Allow», «Principal»: «*»`), що потрібно для забезпечення доступу до статичних файлів вебсайту.

```
resource "aws_s3_bucket_policy" "hosting_bucket_policy" {
    bucket = aws_s3_bucket.frontshop-bucket.id

    policy = jsonencode({
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": "*",
                "Action": "s3:*",
                "Resource": "arn:aws:s3:::${aws_s3_bucket.frontshop-bucket.bucket}/*"
            },
        ]
    })
}
```

Рисунок 2.26 – Налаштування політик доступу

Ресурс `aws_s3_bucket_acl` "acl". Налаштовує публічний доступ до бакету

`acl = "public-read"`: надає дозвіл на публічне читання всіх об’єктів у бакеті.

```
resource "aws_s3_bucket_acl" "acl" {
  depends_on = [
    aws_s3_bucket_ownership_controls.controls,
    aws_s3_bucket_public_access_block.access_block,
  ]

  bucket = aws_s3_bucket.frontshop-bucket.id
  acl    = "public-read"
}
```

Рисунок 2.27 – Налаштування політик доступу

Ресурс `aws_kms_key` "a". Створює ключ шифрування для забезпечення додаткової безпеки

`description = "KMS key 1"`: опис ключа.

`deletion_window_in_days = 10`: визначає час очікування перед видаленням ключа (10 днів), що дозволяє додатковий захист від випадкового видалення.

```
resource "aws_kms_key" "a" {
  description             = "KMS key 1"
  deletion_window_in_days = 10
}
```

Рисунок 2.28 – Створення ключа шифрування

Цей модуль налаштовує інфраструктуру для зберігання даних і хостингу статичного вебсайту на S3 з використанням AWS KMS для шифрування, забезпечуючи баланс між доступністю (для вебсайту) та безпекою (через KMS).

Модуль VPC. Ресурс `aws_vpc` "front-shop-vpc". Створює віртуальну приватну мережу (VPC) з CIDR-блоком, визначеним змінною `var.vpc_cidr`.

Додає тег із зазначенням середовища (`Name = "${var.env}-vpc"`).

```
resource "aws_vpc" "front-shop-vpc" {
  cidr_block = var.vpc_cidr

  tags = {
    Name = "${var.env}-vpc"
  }
}
```

Рисунок 2.29 – Створення віртуальної приватної мережі

Ресурс `aws_internet_gateway` "main". Інтернет-шлюз для VPC, що забезпечує вихід в інтернет для публічних підмереж.

Теги: Ідентифікатор середовища (`Name = "${var.env}-igw"`).

```
resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.front-shop-vpc.id

  tags = {
    Name = "${var.env}-igw"
  }
}
```

Рисунок 2.30 – Налаштування інтернет-шлюзу

Ресурси `aws_subnet` для публічних і приватних підмереж. Публічні підмережі (`public_subnet_a` і `public_subnet_b`):

— прив’язані до різних доступних зон (`availability_zone = var.region1` і `var.region2`);

— CIDR-блоки: визначені змінними (`var.public_subnet_cidr-a` та `var.public_subnet_cidr-b`);

— включено `map_public_ip_on_launch = true` для автоматичного надання публічних IP-адрес при створенні інстансів.

Приватні підмережі (`private_subnet_a` і `private_subnet_b`):

— прив’язані до різних доступних зон;

— використовують приватні CIDR-блоки (`var.private_subnet_cidr-a` та `var.private_subnet_cidr-b`).

```
resource "aws_subnet" "public_subnet_a" {
  vpc_id            = aws_vpc.front-shop-vpc.id
  cidr_block        = var.public_subnet_cidr-a  # Публічний сабнет
  availability_zone = var.region1  # Замініть на вашу доступну зону
  map_public_ip_on_launch = true

  tags = {
    Name = "${var.env}-public-subnet_a"
  }
}
```

Рисунок 2.31 – Створення публічних та приватних підмереж

Ресурси `aws_route_table` та `aws_route_table_association`*. Публічні таблиці маршрутизації (`public_subnet_route_a` та `public_subnet_route_b`):

— містять маршрут до інтернет-шлюзу (`gateway_id = aws_internet_gateway.main.id`);

— приватні таблиці маршрутизації (`private_subnet_route_a` та `private_subnet_route_b`);

— маршрути налаштовані для NAT шлюзів для забезпечення приватного інтернет-з'єднання з інстансами в приватних підмережах.

```
resource "aws_route_table" "private_subnet_route_a" {
  vpc_id = aws_vpc.front-shop-vpc.id

  tags = {
    Name = "${var.env}-private-route-table-a"
  }
}
```

Рисунок 2.32 – Налаштування таблиці маршрутизації

Ресурси `aws_nat_gateway` та `aws_eip` для NAT шлюзів. Забезпечують вихідний доступ до інтернету для приватних підмереж:

Створено два NAT шлюзи (`nat_a` та `nat_b`) у публічних підмережах з публічними IP-адресами (`aws_eip.nat_eip_a.id` та `aws_eip.nat_eip.id`).

```
resource "aws_eip" "nat_eip_a" {
  domain = "vpc"
}
resource "aws_nat_gateway" "nat_a" {
  subnet_id     = aws_subnet.public_subnet_a.id
  allocation_id = aws_eip.nat_eip_a.id
  tags = {
    Name = "${var.env}-nat-gateway-a"
  }
}
```

Рисунок 2.33 – Створення NAT шлюзи

Ресурси `aws_security_group` для бази даних та ECS. `db_security_group`:

— дозволяє TCP-з'єднання до порту 5432 (PostgreSQL) з приватних підмереж;

— Egress дозволяє весь вихідний трафік.

`ecs_security_group`:

— дозволяє підключення до портів 22 (SSH), 5050, 80 (HTTP) та 443 (HTTPS) з будь-якого IP;

— Egress дозволяє весь вихідний трафік.

Рисунок 2.34 – Налаштування портів для ресурсів RSD та ECS

Цей модуль створює базову мережеву інфраструктуру з приватними та публічними підмережами, NAT шлюзами для вихідного інтернет-доступу приватних інстансів та базовими групами безпеки для обмеження доступу до ресурсів.

2.6 Опис роботи додатку в хмарному середовищі

Розроблений додаток складається з двох основних частин: Front-end та Back-end, які інтегровані в архітектуру AWS для забезпечення продуктивності, безпеки та масштабованості. Обидві частини додатку мають чітко розмежовані ролі, і кожна з них працює на основі відповідних сервісів AWS.

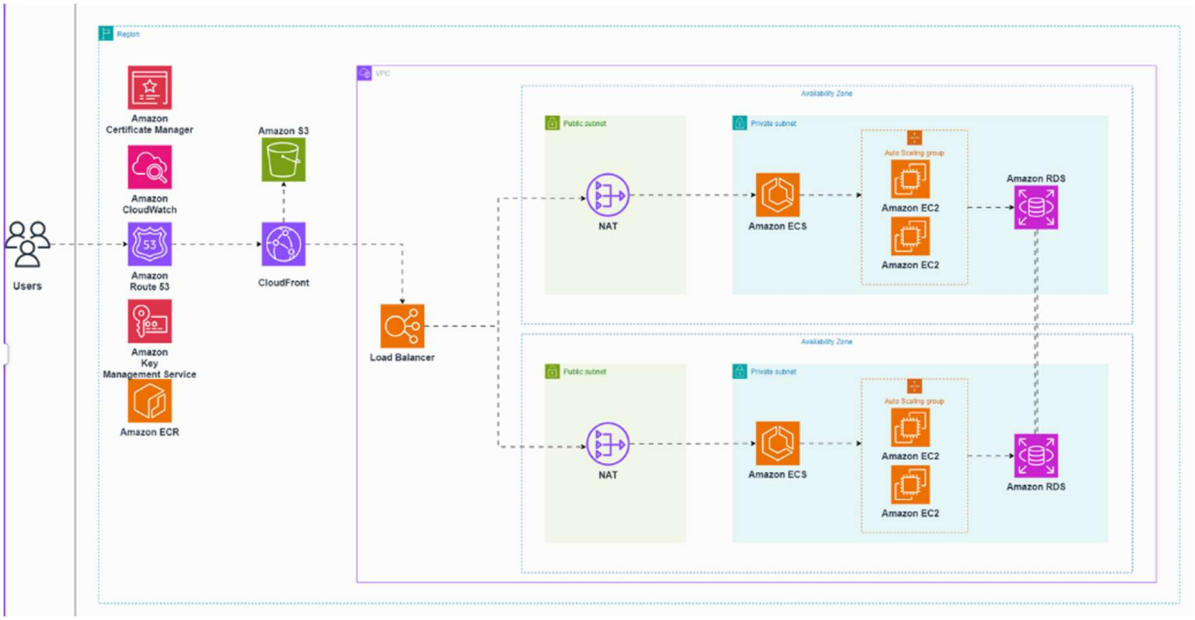


Рисунок 2.35 – Загальна схема роботи додатку в хмарному середовищі

Front-end частина – це статичні файли (HTML, CSS, JavaScript), які відповідають за інтерфейс користувача. Її основна функція – забезпечення швидкого та зручного доступу до додатку з будь-якої точки світу. Робота цієї частини побудована таким чином:

1. Користувачі взаємодіють через DNS-домен. Користувач вводить URL (наприклад, https://frontshop.tech), який обслуговується через Amazon Route 53 (DNS-сервіс). Route 53 перенаправляє запити до CloudFront;
2. Доставка контенту через CloudFront:

— Amazon CloudFront виконує функцію глобальної доставки контенту (CDN). Запити користувачів кешуються на найближчих до них вузлах, що зменшує затримки;

— усі статичні файли (зберігаються в Amazon S3) передаються через CloudFront. Це забезпечує низький час відгуку та мінімізує навантаження на S3.

3. Використовується Amazon Certificate Manager (ACM) для забезпечення HTTPS-з'єднання. Це гарантує безпеку передачі даних між клієнтом і сервером.
4. Логи CloudFront автоматично зберігаються у CloudWatch, що дозволяє аналізувати трафік, виявляти аномалії та підвищувати якість роботи.

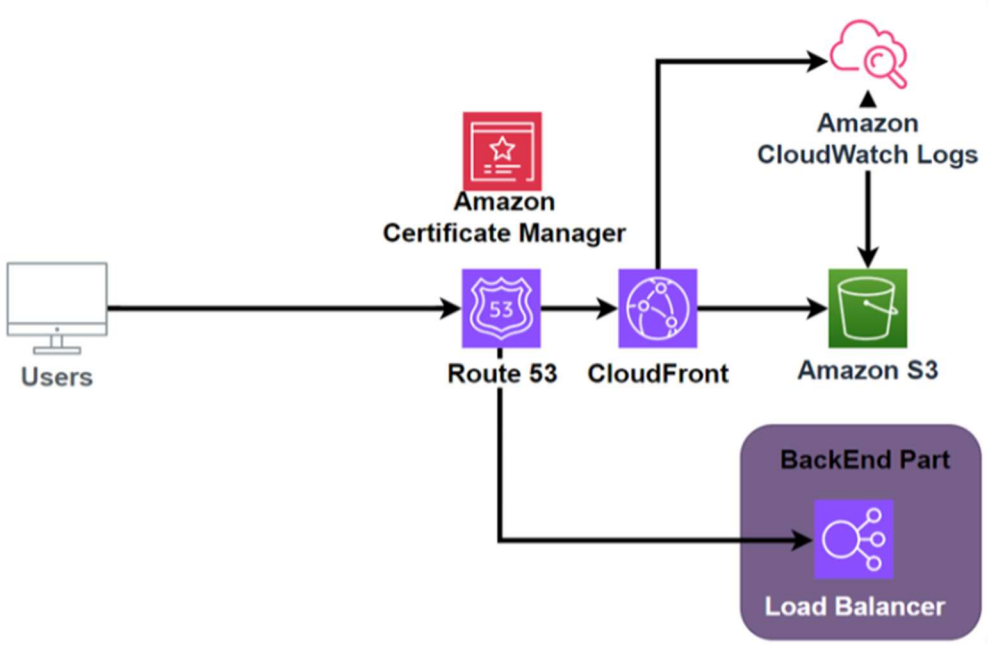


Рисунок 2.36 – Схема роботи статично контенту в додатку

Back-end частина відповідає за бізнес-логіку додатку та взаємодію з базою даних. Її реалізація базується на контейнерах, що працюють у Amazon ECS. Архітектура роботи така:

1. Всі запити на бекенд перенаправляються через Application Load Balancer (ALB). ALB забезпечує рівномірний розподіл запитів між контейнерами.
2. Контейнери в ECS:
   — в Amazon ECS запущено кластер контейнерів, кожен з яких обробляє бізнес-логіку додатку;
   — контейнери розгорнуті в приватних підмережах VPC для забезпечення безпеки.
3. Використовується Auto Scaling для динамічного масштабування кластеру ECS. Якщо навантаження збільшується, автоматично додаються нові EC2-інстанси з контейнерами.
4. Доступ до бази даних:
   — контейнери взаємодіють з Amazon RDS (PostgreSQL) для роботи з даними;
   — доступ до бази даних обмежений лише приватною мережею, що забезпечує безпеку.
5. Ключі доступу до бази даних, API-ключі тощо зберігаються у AWS Secrets Manager, що виключає ризики компрометації.
6. Логи додатку передаються до CloudWatch, що дозволяє оперативно аналізувати помилки, відстежувати виконання запитів та працювати з метриками.[12]

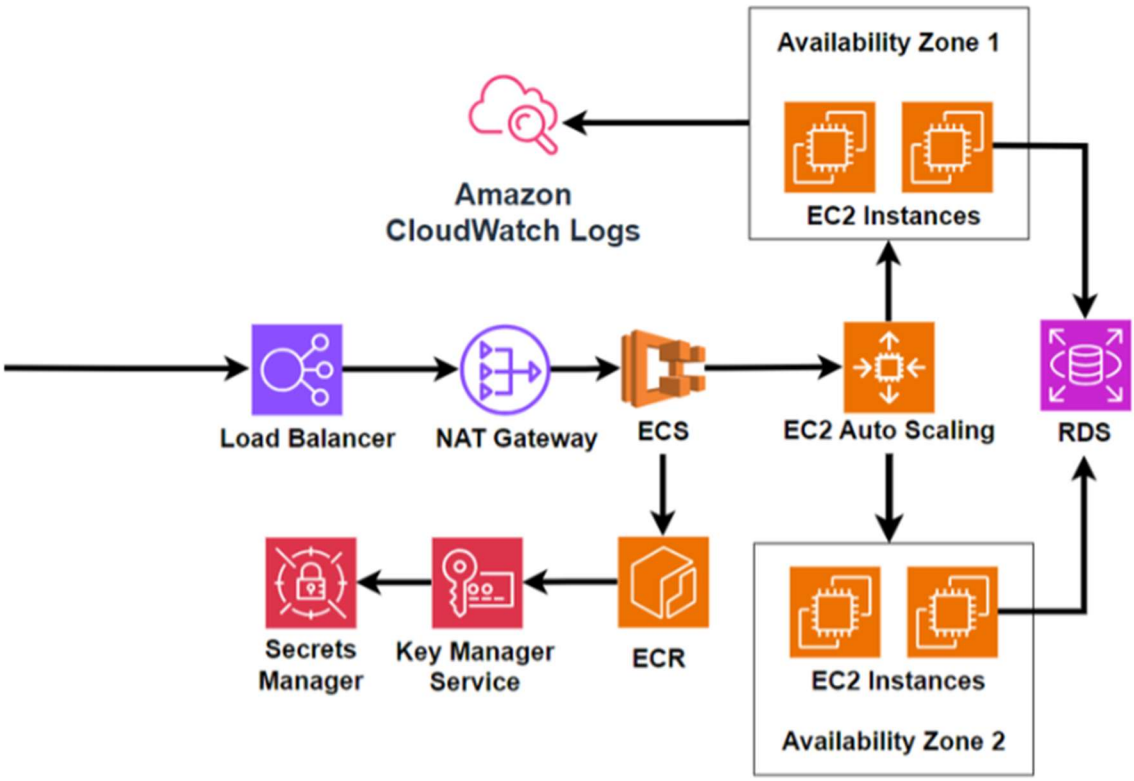


Рисунок 2.37 – Архітектура роботи динамічного контенту в додатку

2.7 Dockerfile для контейнеризації бекенд-додатку

У сучасній розробці програмного забезпечення використання контейнеризації стало стандартом для створення, тестування та розгортання додатків. В рамках проекту був створений Dockerfile для контейнеризації бекенд-додатку на базі ASP.NET Core. Він побудований за принципом багатостадійної збірки, що забезпечує ефективність, безпеку та зменшення розміру кінцевого образу.

На першому етапі використовується офіційний образ .NET SDK версії 6.0. Цей образ включає всі необхідні інструменти для розробки додатків, такі як компілятор, бібліотеки та NuGet для керування залежностями.

Основні дії:

1. Створення робочого каталогу для збірки.

2. Копіювання файлів проекту у контейнер.
3. Виконання команди для завантаження усіх залежностей через NuGet. Це гарантує, що всі бібліотеки та компоненти, необхідні для роботи додатку, будуть доступні.

На другому етапі відбувається безпосередньо компіляція та створення оптимізованої версії додатку. Збірка виконується в режимі "Release", що дозволяє отримати додаток із максимальною продуктивністю та мінімальним розміром.

Результатом є набір скомпільованих файлів, які готові до запуску. Ці файли зберігаються у спеціальному вихідному каталозі, створеному для подальшого використання.

На третьому етапі використовується легкий образ ASP.NET Core Runtime. Цей образ містить лише компоненти, необхідні для виконання додатку, що значно зменшує його розмір порівняно з образом SDK.

Додатково виконуються наступні дії:

1. Оновлення системи та встановлення допоміжних утиліт, таких як curl, для забезпечення діагностики та тестування.
2. Встановлення змінної середовища, яка задає адресу та порт, за якими додаток буде доступний.
3. Копіювання скомпільованих файлів із попереднього етапу в робочий каталог контейнера.

Для забезпечення доступу до додатку ззовні контейнера експонується порт, на якому додаток прийматиме HTTP-запити. У цьому проекті використовується порт 5050, що визначено як стандартний для роботи бекенд-додатку.

Для запуску додатку використовується точка входу, яка викликає виконуваний файл додатку. Цей файл містить основну логіку роботи бекенд-сервісу і починає обробку вхідних запитів одразу після запуску контейнера.

Створений Dockerfile побудований за принципом багатостадійної збірки, що дозволяє:

1. Зменшити розмір фінального образу, використовуючи легкий Runtime-образ для виконання додатку.
2. Забезпечити ізоляцію процесів розробки та виконання, що підвищує безпеку та стабільність.
3. Підготувати додаток до розгортання у продакшн середовищах без необхідності ручного налаштування середовища.

Це рішення демонструє сучасний підхід до контейнеризації, що відповідає вимогам масштабованих та надійних веб-додатків.[16]

```
# Use the ASP.NET Core SDK image to build the project
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /app

# Copy the CSPROJ file and restore any dependencies (via NUGET)
COPY ["ShopFront.WebApi/ShopFront.WebApi.csproj", "ShopFront.WebApi/"]
RUN dotnet restore "ShopFront.WebApi/ShopFront.WebApi.csproj"

# Copy the project files and build the release
COPY . ./
RUN dotnet publish "ShopFront.WebApi/ShopFront.WebApi.csproj" -c Release -o out

# Generate runtime image
FROM mcr.microsoft.com/dotnet/aspnet:6.0

RUN apt update && apt upgrade -y
RUN apt install curl -y

ENV ASPNETCORE_URLS=http://+:5050
WORKDIR /app
COPY --from=build-env /app/out .
EXPOSE 5050
ENTRYPOINT ["dotnet", "ShopFront.WebApi.dll"]
```

Рисунок 2.38 – Код для контейнеризації додатку

2.8 Створення CI/CD процессу

Цей файл YAML реалізує пайплайн автоматизації збирання, тестування, контейнеризації та розгортання застосунку у середовищі Amazon ECS (Elastic

Container Service) з використанням Amazon ECR (Elastic Container Registry). Пайплайн виконується на GitHub Actions і забезпечує повний цикл CI/CD.

У пайплайні визначено джоб build-and-deploy, який виконується на віртуальному середовищі ubuntu-latest. Це середовище забезпечує доступ до всіх необхідних інструментів для виконання завдань з контейнеризації та деплою.[20]

Кроки процесу CI/CD:

1. Checkout Repository

— дія: використовується стандартна GitHub Action actions/checkout@v4;

— мета: завантажити репозиторій з вихідним кодом для подальшої обробки;

— важливість: цей крок забезпечує доступ до файлів проєкту, які потрібні для побудови контейнера.

```
steps:
- name: Checkout Repository
  uses: actions/checkout@v4
```

Рисунок 2.39 – Завантаження репозиторію

2. Configure AWS Credentials

— дія**:** Використовується GitHub Action aws-actions/configure-aws-credentials для налаштування облікових даних AWS;

— Параметри:

1. role-to-assume: Роль AWS (наприклад, із secrets). Це дозволяє отримати доступ до ресурсів AWS.
2. role-session-name: Унікальне ім'я сесії для AWS IAM.
3. aws-region: Регіон AWS, у якому працює інфраструктура (у даному випадку eu-central-1).

— Мета**:** Забезпечити авторизацію у хмарних сервісах AWS для роботи з ресурсами, такими як ECS та ECR.

```
- name: Configure aws credentials
  uses: aws-actions/configure-aws-credentials@master
  with:
    role-to-assume: ${{ secrets.BACKROLE }}
    role-session-name: seesionrole
    aws-region: eu-central-1
```

Рисунок 2.40 – Авторизація у хмарний сервіс

3. Login to Amazon ECR

— дія: Використовується aws-actions/amazon-ecr-login@v2;

— мета: Авторизація в Amazon Elastic Container Registry (ECR), щоб отримати доступ до приватних репозиторіїв контейнерів;

— результат**:** Повертається змінна login-ecr.outputs.registry, яка зберігає URL реєстру, необхідний для подальших дій.

```
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login@v2
```

Рисунок 2.41 – Логін в ECR

4. Build, Tag, and Push Docker Image

— Дія: Створення, маркування та завантаження контейнера у реєстр Amazon ECR;

— Опис дій:

1. REGISTRY: URL реєстру ECR.
2. REPOSITORY: Ім'я репозиторія контейнера (наприклад, frontshop-ecr).
3. Завантаження образу у реєстр ECR:

— Мета: Забезпечити контейнеризацію застосунку та його доступність у реєстрі для подальшого розгортання.

```
- name: Build, tag, and push docker image to Amazon ECR
  env:
    REGISTRY: ${{ steps.login-ecr.outputs.registry }}
    REPOSITORY: frontshop-ecr
  run: |
    docker build -t $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }} .
    docker push $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }}
```

Рисунок 2.42 – Створення та завантаження контейнера у реєстр

5. Update ECS Service

— Дія: Використовується CLI-команда AWS для оновлення ECS сервісу;

— Кроки:

1. Зупинка поточних завдань ECS. Це звільняє ресурси для запуску оновленого контейнера.
2. Оновлення ECS сервісу, щоб використати новий Docker-образ
3. Увімкнено примусове оновлення розгортання, що дозволяє ECS використовувати нову версію контейнера.

— Мета: Забезпечити автоматичне оновлення сервісів у кластері ECS з новим образом.

```
- name: Update ECS Service
  run: |
    aws ecs stop-task --cluster ${{ secrets.ECS_CLUSTER }} --
    aws ecs update-service --cluster ${{ secrets.ECS_CLUSTER
```

Рисунок 2.43 – Оновлення сервісу ECS

Основні переваги реалізованого процесу CI/CD:

— автоматизація: Завдяки GitHub Actions та AWS CLI всі кроки – від збирання до розгортання – виконуються без ручного втручання;

— масштабованість: Підтримка AWS ECS дозволяє легко масштабувати контейнери відповідно до навантаження;

— швидкість: Оновлення сервісів ECS відбувається автоматично після завантаження нового контейнера, що мінімізує простої;

— безпека: Використання секретів для облікових даних AWS гарантує, що чутливі дані захищені;

— гнучкість: Завдяки використанню Docker, код можна запускати у будь-якому середовищі, яке підтримує контейнери.

Цей процес демонструє сучасний підхід до DevOps, який забезпечує високу продуктивність, надійність і автоматизацію в управлінні інфраструктурою.[21]

```yaml
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v4

    - name: Configure aws credentials
      uses: aws-actions/configure-aws-credentials@master
      with:
        role-to-assume: ${{ secrets.BACKROLE }}
        role-session-name: seesionrole
        aws-region: eu-central-1

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v2

    - name: Build, tag, and push docker image to Amazon ECR
      env:
        REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        REPOSITORY: frontshop-ecr
      run: |
        docker build -t $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }} .
        docker push $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }}

    - name: Update ECS Service
      run: |
        aws ecs stop-task --cluster ${{ secrets.ECS_CLUSTER }} --task $(aws ecs list-tasks --cluster ${{ secrets.ECS_CLUSTER }} --service ${{ secrets.ECS_SERVICE }}
        aws ecs update-service --cluster ${{ secrets.ECS_CLUSTER }} --service ${{ secrets.ECS_SERVICE }} > /dev/null
```

Рисунок 2.44 – Загальний код CI/CD процессу

РОЗДІЛ 3 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

У цій частині наведемо техніко-економічне обґрунтування використання хмарних сервісів Amazon Web Services (AWS) для підтримки інфраструктури проєкту. На основі зведених витрат за перші два місяці можна оцінити загальні витрати на інфраструктуру та визначити економічну ефективність обраних рішень.

Під час побудови інфраструктури в AWS важливим аспектом є оптимізація витрат. У цьому аналізі ми порівняли витрати на два підходи до організації контейнерних оркестраційних платформ: використання Amazon ECS з Fargate та Amazon EKS з EC2-інстансами.

Щомісячні витрати при використанні ECS/Fargate

1. Amazon S3: Хмарне сховище для статичних файлів, таких як журнали, бекапи та інші дані. Його витрати складають 0.27 USD на місяць, що еквівалентно ~3.24 USD на рік.
2. Amazon ECR: Сервіс для зберігання контейнерних образів, який коштує 0.14 USD на місяць або ~1.68 USD на рік.
3. Secrets Manager: Управління конфіденційними даними (наприклад, паролями або API-ключами) обійдеться у 0.66 USD на місяць або ~7.92 USD на рік.
4. Route 53: DNS-сервіс для маршрутизації запитів до ваших додатків, витрати якого складають 0.21 USD на місяць або ~2.52 USD на рік.
5. Elastic Load Balancer (ELB): Балансування навантаження між контейнерами або інстансами. Вартість – 15.24 USD на місяць, або ~182.88 USD на рік.
6. Amazon RDS: Реляційна база даних, що використовується для зберігання структурованих даних. Її щомісячна вартість – 18.32 USD, а річна – ~219.84 USD.

7. CloudWatch: Моніторинг продуктивності сервісів та ресурсів. Вартість складає 2.26 USD на місяць, або ~27.12 USD на рік.
8. ECS/Fargate: Платформа для запуску контейнерів без управління серверами. Вартість сервісу складає 56.83 USD на місяць, що еквівалентно ~681.96 USD на рік.

Загальна вартість з ECS/Fargate складає**:**

— 93.93 USD на місяць;

— ~1127.16 USD на рік.

Щомісячні витрати при використанні EKS/EC2**.** Всі сервіси, такі як Amazon S3, ECR, Secrets Manager, Route 53, ELB, RDS та CloudWatch, мають ті ж самі витрати, що і в ECS/Fargate.

Контейнерна оркестрація на базі Kubernetes із використанням EC2-інстансів для роботи кластерів. Цей варіант коштує 86.62 USD на місяць, що становить ~1039.44 USD на рік.

Загальна вартість з EKS/EC2 складає:

— 123.72 USD на місяць;

— ~1484.64 USD на рік.

3.1 Порівняння витрат.

Отже, зробивши обрахунки використатння обох сервісів можемо зазначити що:

1. Економія при використанні ECS/Fargate:
   — Щомісячна економія становить 29.79 USD, що еквівалентно ~24% зниження витрат у порівнянні з EKS/EC2;
   — За рік заощадження складає 357.48 USD, або ~24% економії.

2. Причини економії:
   - Використання Fargate дозволяє уникнути витрат на управління EC2-інстансами, які необхідні для EKS.
   - У випадку Fargate ви платите тільки за використані ресурси, тоді як для EKS потрібно оплачувати EC2-інстанси незалежно від їх завантаження.

Використання Amazon ECS з Fargate є економічно вигіднішим рішенням для організації інфраструктури. Воно дозволяє знизити витрати на ~24% щомісяця та забезпечує більш просте управління контейнерами за рахунок відсутності потреби в управлінні серверами.

Крім економії коштів, Fargate пропонує значну спрощеність у розгортанні та масштабуванні додатків, адже вам не потрібно керувати EC2-інстансами або самим кластером Kubernetes.

Таким чином, вибір ECS/Fargate є оптимальним для бізнесів, які прагнуть знизити витрати на інфраструктуру, зберігаючи при цьому її надійність та масштабованість.

ВИСНОВКИ

У ході виконання даної роботи було розроблено автоматизовану, масштабовану та надійну інфраструктуру для підтримки додатків на базі хмарних сервісів Amazon Web Services (AWS) з використанням інструменту Terraform. На основі проведених досліджень та практичної реалізації досягнуто наступних результатів.

Було проведено аналіз потреб додатків аутсорсингових компаній, що дозволило визначити ключові компоненти інфраструктури, такі як ECS для контейнеризації, RDS для баз даних, S3 для зберігання даних, а також Route 53 і CloudFront для управління доменами та доставки контенту.

Розроблено та впроваджено код Terraform, який дозволяє автоматизувати створення цих компонентів у хмарі AWS. Особлива увага приділялася оптимізації процесу розгортання інфраструктури для забезпечення її надійності, масштабованості та безпеки.

Налаштовано CI/CD-процеси з використанням GitHub Actions для автоматизації розгортання додатків, що забезпечило швидке і надійне оновлення кодової бази без ручного втручання.

Інтегровано інструменти моніторингу, такі як AWS CloudWatch, для відстеження стану інфраструктури, а також реалізовано політики безпеки з використанням AWS IAM, Security Groups та шифрування даних.

Впровадження інфраструктури як коду з використанням Terraform в поєднанні з AWS дозволило створити стійку, масштабовану та безпечну інфраструктуру, яка відповідає сучасним вимогам аутсорсингових компаній. Результати роботи підтверджують ефективність такого підходу для забезпечення автоматизованого управління ресурсами та підтримки численних клієнтських додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Infrastructure as Code (IaC) - Red Hat. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (дата звернення: 22.12.2024).
2. Enabling Infrastructure as Code (IaC) with CI/CD: Key Benefits for Customers - Bilal. LinkedIn. URL: https://www.linkedin.com/pulse/enabling-infrastructure-code-iac-cicd-key-benefits-customers-bilal/ (дата звернення: 22.12.2024).
3. What is IaC? - Cisco. URL: https://www.cisco.com/c/en/us/solutions/cloud/what-is-iac.html (дата звернення: 22.12.2024).
4. Infrastructure as Code - Wikipedia. URL: https://en.wikipedia.org/wiki/Infrastructure_as_code (дата звернення: 22.12.2024).
5. Infrastructure as Code (IaC) - TechTarget. URL: https://www.techtarget.com/searchitoperations/definition/Infrastructure-as-Code-IAC (дата звернення: 22.12.2024).
6. Terraform Documentation - HashiCorp. URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 22.12.2024).
7. AWS Provider Documentation - Terraform Registry. URL: https://registry.terraform.io/providers/hashicorp/aws/latest/docs (дата звернення: 22.12.2024).
8. How to Create AWS Resources with Terraform - Selim Abidin. Medium. URL: https://medium.com/@SelimAbidin/how-to-create-aws-resource-with-terraform-d637097ea89a (дата звернення: 22.12.2024).
9. Cloud compute by Amazon Web Services – Amazon URL: https://aws.amazon.com/en/products
10. A 10,000 Feet Overview of AWS and Its Resources (Part 2) - Yash Kukreja. Medium. URL: https://yash-kukreja-98.medium.com/a-10-000-feet-overview-of-aws-and-its-resources-part-2-7ba48aee39b0 (дата звернення: 22.12.2024).
11. Enterprise Resources - AWS. URL: https://aws.amazon.com/en/enterprise/resources/ (дата звернення: 22.12.2024).
12. AWS CloudFormation User Guide - Amazon. URL: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html (дата звернення: 22.12.2024).
13. Docker Overview - Docker Documentation. URL: https://docs.docker.com/get-started/docker-overview/ (дата звернення: 22.12.2024).
14. What is Containerization? - NextLabs. URL: https://www.nextlabs.com/what-is-containerization/ (дата звернення: 22.12.2024).
15. Docker Alternatives - SigNoz. URL: https://signoz.io/comparisons/docker-alternatives/ (дата звернення: 22.12.2024).

16. How Docker Containers Work - FreeCodeCamp. URL: https://www.freecodecamp.org/news/how-docker-containers-work/ (дата звернення: 22.12.2024).
17. What is CI/CD? - AB Tasty. URL: https://www.abtasty.com/resources/ci-cd/ (дата звернення: 22.12.2024).
18. CI/CD Topics - GitLab. URL: https://about.gitlab.com/topics/ci-cd/ (дата звернення: 22.12.2024).
19. What is CI/CD? - Black Duck. URL: https://www.blackduck.com/glossary/what-is-cicd.html (дата звернення: 22.12.2024).
20. GitHub Actions Documentation - GitHub. URL: https://docs.github.com/en/actions (дата звернення: 22.12.2024).
21. How to Set Up a Continuous Deployment Pipeline with GitLab - DigitalOcean. URL: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-continuous-deployment-pipeline-with-gitlab-on-ubuntu (дата звернення: 22.12.2024).

Додадот А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістра

НА ТЕМУ: «СИСТЕМА АВТОМАТИЗОВАНОГО КЕРУВАННЯ ІНФРАСТРУКТУРОЮ ДЛЯ АУТСОРСИНГОВИХ ДОДАТКІВ У ХМАРНОМУ СЕРЕДОВИЩІ»

Виконав студент групи КСДМ-62
Федоренко Максим Андрійович

Керівник дипломної роботи:
Волохін Віталій Васильович,
канд.техн.наук, доцент

Київ – 2024

Об`єкт дослідження – процес розробки та реалізації інфраструктури інформаційної системи з використанням інструментарію Terraform та хмарних сервісів Amazon Web Services

Предмет дослідження – Застосування технологій Infrastructure as Code (IaC) з використанням AWS і Terraform для автоматизації, масштабування та управління інфраструктурою для додатків аутсорсингових компаній

Мета роботи – Розробити автоматизовану, масштабовану та надійну хмарну інфраструктуру для підтримки додатків на основі AWS, використовуючи Terraform для спрощення процесу розгортання та управління, що підвищить ефективність та зменшить помилки в процесі управління інфраструктурою

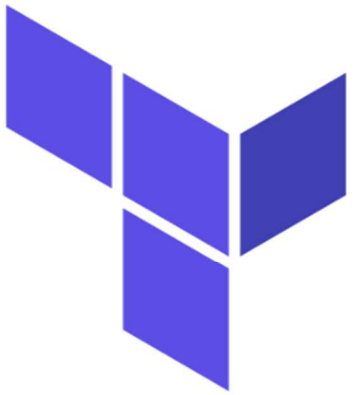
Актуальність обраної теми

Висока конкуренція на ринку змушує компанії шукати рішення, що дозволять автоматизувати процеси, зменшити витрати та забезпечити безперервність надання послуг.

Одним із таких рішень є підхід інфраструктури як коду (IaC), що дозволяє створювати, підтримувати та оновлювати інфраструктуру за допомогою програмного коду.

Особливості використання IaC

- Автоматизація
- Масштабування
- Простота управління
- Відтворюваність
- Прозорість і контроль версій
- Економія ресурсів

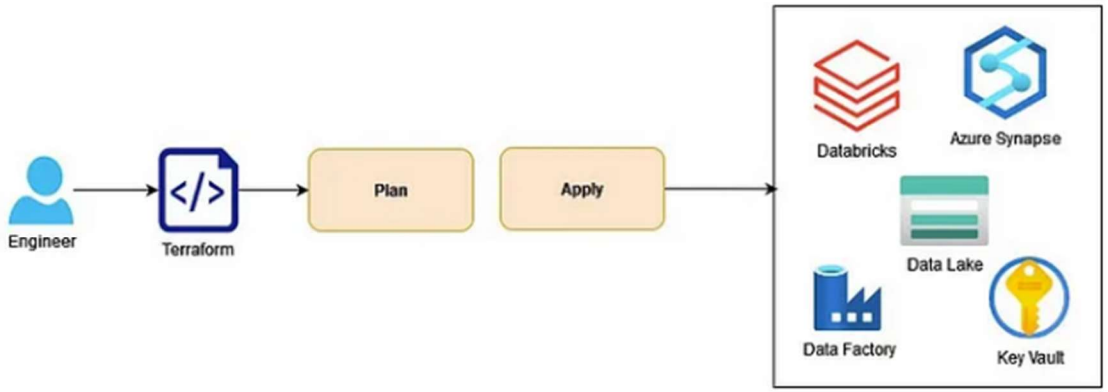


HashiCorp
Terraform

ІНСТРУМЕНТИ ДЛЯ РОЗГОРТАННЯ ТА УПРАВЛІННЯ КОНФІГУРАЦІЄЮ

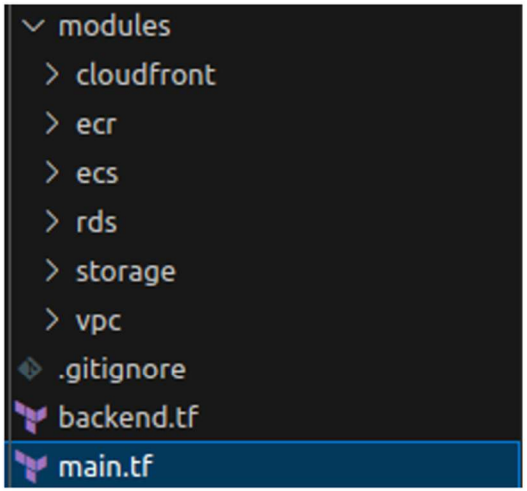
Інструмент	Переваги	Недоліки
Puppet	Масштабованість, підходить для великих систем.	Складний синтаксис, висока вартість.
SaltStack	Швидкість, масштабованість.	Порівняно складний для налаштування та освоєння.
Chef	Гнучкість, сильна спільнота.	Висока крива навчання через складний DSL-синтаксис
Ansible	Простота, YAML-синтаксис, без агентів.	Може бути повільнішим.
Docker	Просте масштабування, інтеграція в DevOps.	Не для управління інфраструктурою.
Terraform	AWS-підтримка, масштабованість, модульність.	Менш ефективний для конфігурацій.

ГРАФІЧНА СХЕМА РОБОТИ TERRAFORM




```
  # module.s3.aws_s3_bucket_website_configuration.reactapp-static-website will
be created
  + resource "aws_s3_bucket_website_configuration" "reactapp-static-website" {
      + bucket           = (known after apply)
      + id               = (known after apply)
      + routing_rules    = (known after apply)
      + website_domain   = (known after apply)
      + website_endpoint = (known after apply)

      + error_document {
          + key = "index.html"
        }

      + index_document {
          + suffix = "index.html"
        }
    }

Plan: 12 to add, 0 to change, 0 to destroy.
```

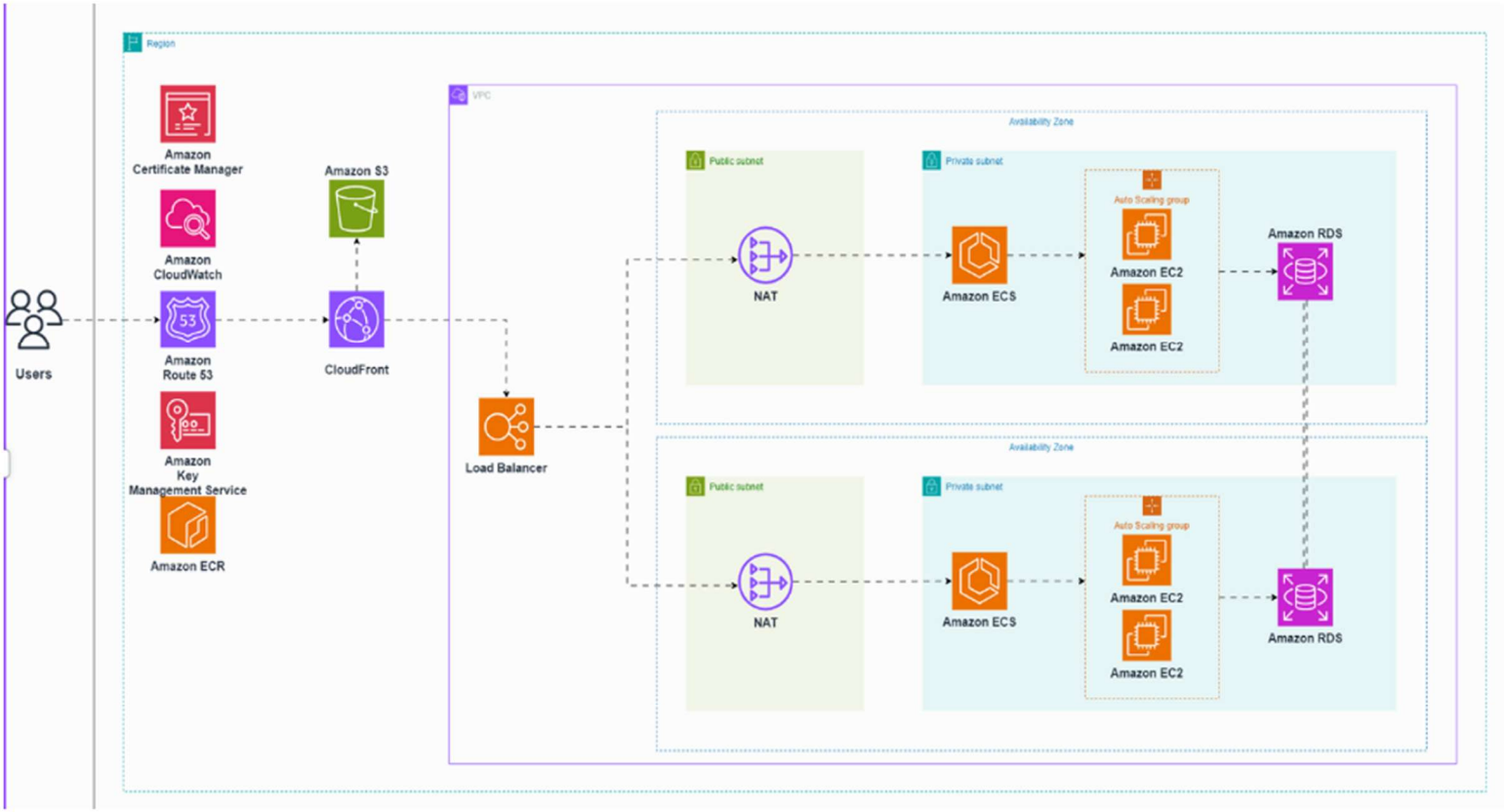
Приклад реалізації коду в репозиторії



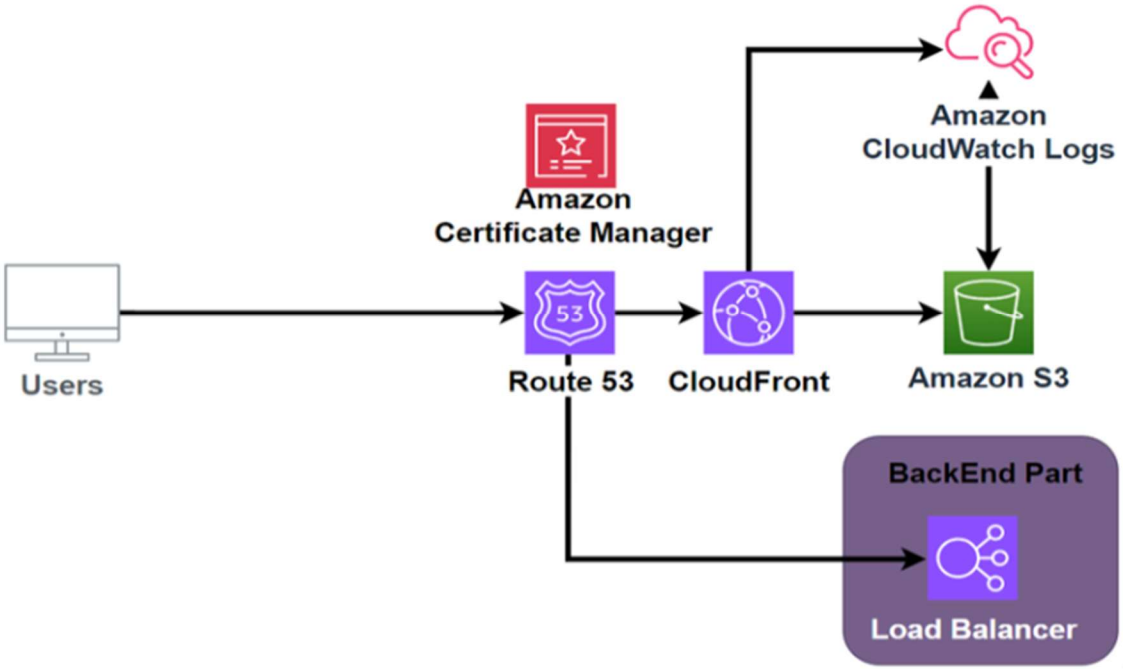

```
resource "aws_s3_bucket" "frontshop-bucket" {
  bucket = "frontshop-bucket"
  tags = {
    Name        = "Front-shop bucket"
    Environment = "Dev"
  }
}

resource "aws_s3_bucket_ownership_controls" "controls" {
  bucket = aws_s3_bucket.frontshop-bucket.id
  rule {
    object_ownership = "BucketOwnerPreferred"
  }
}
```

```
module "front-shop-vpc" {
    source               = "./modules/vpc"
    env                  = "front-shop"
    vpc_cidr             = var.vpc_cidr
    public_subnet_cidr-a  = var.public_subnet_cidr-a
    private_subnet_cidr-a = var.private_subnet_cidr-a
    public_subnet_cidr-b  = var.public_subnet_cidr-b
    private_subnet_cidr-b = var.private_subnet_cidr-b
```

Система хмарної архітектури додатку

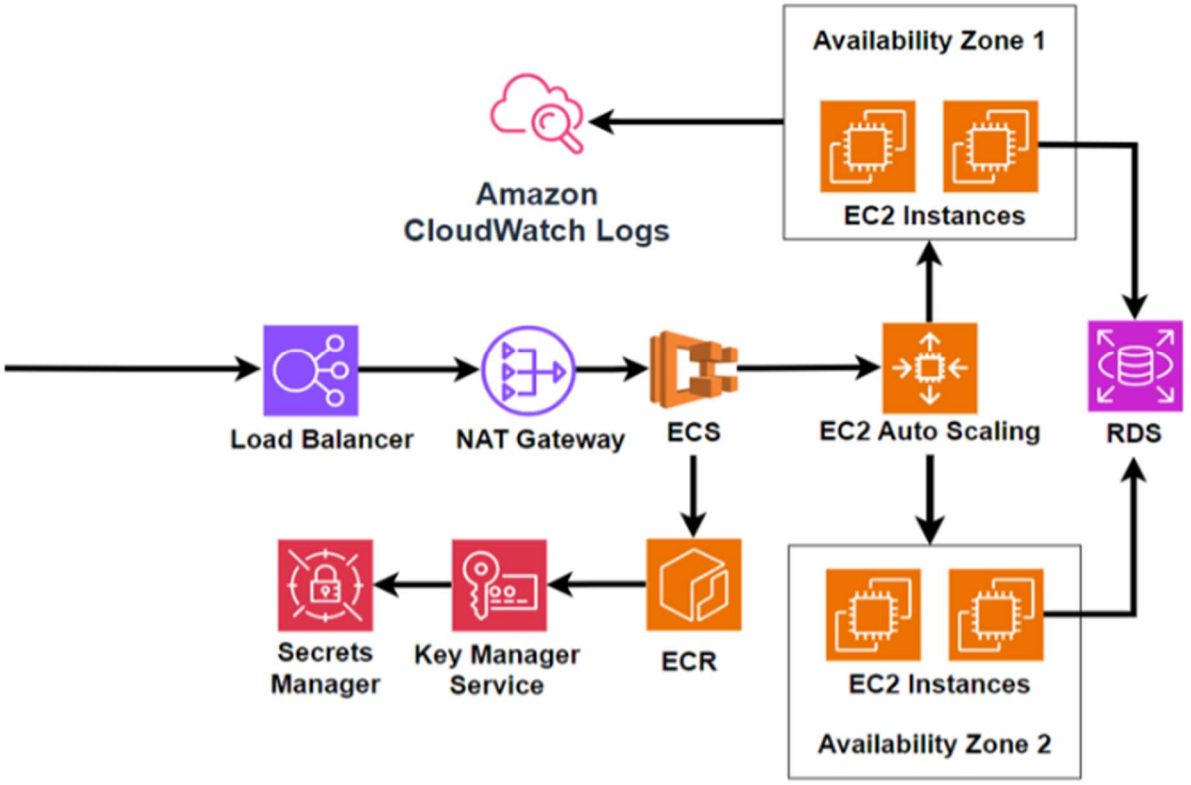


Як працює статичний контент додатку в AWS



9

Динамічний контент додатку в Amazon



Використання CD процесу для доставки коду

```yaml
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v4

    - name: Configure aws credentials
      uses: aws-actions/configure-aws-credentials@master
      with:
        role-to-assume: ${{ secrets.BACKROLE }}
        role-session-name: seesionrole
        aws-region: eu-central-1

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v2

    - name: Build, tag, and push docker image to Amazon ECR
      env:
        REGISTRY: ${{ steps.login-ecr.outputs.registry }}
        REPOSITORY: frontshop-ecr
      run: |
        docker build -t $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }} .
        docker push $REGISTRY/$REPOSITORY:${{ secrets.IMAGE_TAG }}

    - name: Update ECS Service
      run: |
        aws ecs stop-task --cluster ${{ secrets.ECS_CLUSTER }} --task $(aws ecs list-tasks --cluster ${{ secrets.ECS_CLUSTER }} --service ${{ secrets.ECS_SERVICE }}
        aws ecs update-service --cluster ${{ secrets.ECS_CLUSTER }} --service ${{ secrets.ECS_SERVICE }} > /dev/null
```

Економія використання

Економія при використанні ECS/Fargate:

Щомісячна економія становить 29.79 USD, що еквівалентно ~24% зниження витрат порівняно з EKS/EC2, що в річному обчисленні становить 357.48 USD (~24% економії). Основною причиною є відсутність потреби в управлінні EC2-інстансами, адже Fargate дозволяє оплачувати лише використані ресурси.

Таким чином, вибір ECS/Fargate є оптимальним для бізнесів, які прагнуть знизити витрати на інфраструктуру, зберігаючи при цьому її надійність та масштабованість.

Висновки

У магістерській роботі було проведено процес впровадження інфраструктури як коду для підтримки додатків на базі хмарних сервісів AWS з використанням Terraform. Було проведено аналіз концепцій IaC, що дало можливість оцінити переваги автоматизації інфраструктури та її важливість для аутсорсингових компаній.

В результаті було визначено основні компоненти інфраструктури, які необхідно розгорнути, такі як ECS, CloudFront, RDS, а також балансувальник навантаження. Було розроблено та протестовано код Terraform для автоматизованого створення цих компонентів у хмарі AWS, що забезпечує надійність, масштабованість та безпеку інфраструктури.

У результаті роботи було успішно розгорнуто автоматизовану інфраструктуру, що включає CD-процеси для безперервного впровадження змін, а також моніторинг стану компонентів. Цей підхід значно полегшив управління ресурсами, знизив ризик помилок і дозволив швидко реагувати на зміни в середовищі.