Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Безпеки інформаційних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Використання клаудпровайдера AWS для забезпечення безпеки веб-застосувань
(тема)

Виконав:
студент 2 курсу, групи БІКСм-20-1

Лисаков В.І.
(прізвище, ініціали)

Спеціальності 125 Кібербезпека
(код і повна назва спеціальності)

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

Освітня програма Безпека інформаційних і комунікаційних систем
(повна назва освітньої програми)

Керівник доцент Сєвєрінов О. В.
(посада. прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Безпеки інформаційних технологій_____

Рівень вищої освіти_____ другий (магістерський)_____

Спеціальність_____125 Кібербезпека _____
(код і повна назва)

Освітня програма_____«Безпека інформаційних і комунікаційних систем»____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Лисакову Владиславу Ігоровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи Використання клаудпровайдера AWS для забезпечення безпеки веб-застосувань
_____
затверджена наказом по університету від _8_ __листопада__2021 р. №1685Ст _____
2. Термін подання студентом роботи до екзаменаційної комісії _14_грудня_____2021 р.
3. Вихідні дані до роботи ідентифікація та автентифікація користувача, методологія CI/CD, та повністю ізольована приватна мережа в клаудпровайдері AWS, програмний код для _____ оркестрування додатків.
_____
_____
_____
_____
_____
_____
_____
_____
_____
4. Перелік питань, що потрібно опрацювати в роботі _____
Аналіз моделей загроз та порушника.
Розгляд технологій Terraform та Terragrunt.
Дослідження безпеки сервісів AWS, що використовуються для розгортання веб-додатків.
Реалізація власного веб-додатку з використанням сервісів AWS.
Написання програмного коду для оркестрації сервісів AWS.
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) _презентаційний матеріал у вигляді слайдів_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача завдання	01.09.2021	Виконано
2	Робота з джерелами за тематикою роботи	19.09.2021 – 23.09.2021	Виконано
3	Вивчення основних понять в сфері хмарних систем	24.09.2021 – 25.09.2021	Виконано
4	Дослідження документації використання сервісів AWS	26.09.2021 – 27.10.2021	Виконано
5	Створення власного акаунту в AWS	28.10.2021 – 05.11.2021	Виконано
6	Отримання практичних навичок конфігурації сервісів для CI/CD та автентифікації користувачів.	06.11.2021 – 15.11.2021	Виконано
7	Конфігурація приватної мережі в клаудпровайдері	16.11.2021 –27.11.2021	Виконано
8	Написання програмного кода за допомогою мови програмування Python та мови конфігурації Terraform (Terragrunt)	28.11.2021 – 04.12.2021	Виконано
9	Оформлення пояснювальної записки	05.12.2021 – 09.12.2021	Виконано

Дата видачі завдання __1__ _ вересня_____2021 р.

Студент _____
(підпис)

Керівник роботи _____ _____ доцент Сєвєрінов О. В.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до курсової роботи містить: 71 с., 38 рис., 24 джерела, 1 додаток

ВЕБ ДОДАТОК, СЕРВІС, AWS, COGNITO, CODEPIPELINE, CODEBUILD, CODEDEPLOY, NAT, VPC, LOAD BALANCER, SECRET MANAGER, ECR, ECS, DOCKER, SDLC.

Об’єкт дослідження – безпека веб–додатків, побудованих за допомогою сервісів AWS.

Предмет дослідження – сервіси AWS для створення веб–додатків.

Мета роботи –моделювання та побудова безпечного веб–додатку за допомогою сервісів клаудпровайдера AWS.

Методи дослідження – вивчення документації сервісів, які використовуються при побудуванні веб-додатків, перегляд інструкцій інтегрування сервісів з власним додатком, написання програмного кода на мові програмування Python та мові конфігурації хмарних провайдерів Terraform (Terragrunt). Спроба створення власного secure CI/CD (SDLC).

Були розглянуті аспекти безпеки при використанні розповсюджених сервісів клаудпровайдера AWS, ймовірні загрози, надана загальна їх характеристика. Розглянуті та проаналізовані моделі порушника та загроз. Створено систему CI/CD для власного веб–додатку, а також систему авторизації та автентифікації користувача, як окремі програмні модулі. Вивчена та реалізована приватна мережа за допомогою AWS VPC. Розгортання власних додатків та використаних сервісів в приватній мережі для забезпечення безпеки.

ABSTRACT

Explanatory note to the thesis contains 71 pages, 38 figures, 24 references, 1 appendix.

WEB-APPLICATION, SERVICE, AWS, COGNITO, CODEPIPELINE, CODEBUILD, CODEDEPLOY, NAT, VPC, LOAD BALANCER, SECRET MANAGER, ECR, ECS, DOCKER, SDLC

The object of research is a web applications' security which are built with the AWS cloud provider.

The subject of research are AWS services which use to build web–application.

The purpose of the work is to study and investigate of AWS services, modeling and building a secure web–application using these services.

A methods of research are studying services' documentations, which use for development and deploy of web–application, reviewing the instructions for the integration with own app, writing programming code both on Python programming and Terraform (Terragrunt) configuration languages. An attempt to create our own secure CI/CD.

The security aspects of using common AWS cloud provider services were reviewed, probable threats, and their general characteristics were covered. The intruder and threat models were considered and analyzed. Created CI/CD system for own web application, as well as user authentication and authorization system as separate software modules. Studied and implemented a private network using AWS VPC. Deployed own applications and used services on a private network for security

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

AWS – англ. Amazon Web Services – компанія, що надає платформу хмарних обчислень.

VPC – англ. Virtual Private Cloud – сервіс для створення логічно ізольованих віртуальних мереж.

IAM – англ. Identity and Access Management – сервіс для безпечного управління доступом до сервісів та ресурсів.

EC2 – англ. Elastic Compute Cloud – сервіс для отримання безпечних масштабованих обчислювальних ресурсів у хмарі.

ECS – англ. Elastic Container Service – сервіс для оркестрації Docker контейнерів.

ECR – англ. Elastic Container Registry – сервіс для зберігання, управління та запуску Docker контейнерів.

SSL – англ. Secure Sockets Layer – рівень захищених сокетів.

SSDLC – англ. Secure Software Development Life Cycle – життєвий цикл безпечної розробки.

NAT – англ. Network Address Translation – перетворення мережевих адрес.

ВСТУП

В сучасних умовах велика кількість компаній, що діють в різних сферах, відходять від використання приватних серверів та вдаються до співпраці з платформами хмарних обчислень. Поряд з кроками в напрямку розвитку інформаційного суспільства, важливими є питання його захисту, захисту інформаційної безпеки та інформаційних національних інтересів України.

Для розвитку інформаційного суспільства характерне поєднання глобальних інформаційно-комунікаційних систем, інтеграція баз даних та знань, зростання спектру послуг і сервісів, різке збільшення клієнтського навантаження на серверний простір. Результатом еволюційного розвитку інформаційних технологій (далі – ІТ) за останнє десятиліття стали «cloud computing» – «хмарні технології» (обчислення).

Хмарні обчислення (cloud computing) – це технологія розподіленої обробки даних в якій комп'ютерні ресурси і потужності надаються користувачеві як інтернет-сервіс.

За допомогою їх впровадження стає можливим вирішення виниклого протиріччя – невідповідності між бурхливим зростанням обсягів інформаційних потоків і мережевих сервісів та сучасним технічним станом програмно-апаратного забезпечення мережевої інфраструктури інформаційних систем. Під час використання хмарних технологій програмне та технічне забезпечення надається користувачеві як Інтернет-сервіс; він має доступ лише до власних даних, а не до інфраструктури, операційної системи і програмного забезпечення, з якими працює. "Хмара" – це Інтернет, який приховує усі технічні деталі.

Інтенсивне освоєння хмарних технологій ІТ-компаніями, застосування у малому та середньому бізнесі, у навчальному процесі, для управління підприємствами, перехід на хмарні технології ІТ-інфраструктур державних органів – все це свідчить про черговий якісний стрибок у галузі інформаційних

технологій. З іншого боку, як вказано в [1], “важливе рішення щодо використання cloud computing несе в собі реальні й перспективні загрози безпеки бізнесам, якими вони керують, і це може розглядатися, як новий технологічний, економічний, фінансовий і безпековий виклик XXI ст.”

Основні переваги, які можуть дати хмарні сервіси:

– економія засобів на придбання програмного забезпечення;

– зниження потреби в спеціалізованих приміщеннях;

– економія дискового простору;

– високий рівень безпеки, підтверджений сертифікатами;

– зниження потреби в великій кількості матеріальних ресурсів;

– зниження потреби в адміністраторах;

– підвищення якості та швидкості програмного продукту;

– полегшення процесів розробки;

– реалізація найпоширеніших сервісів та можливість швидкої їх інтеграції в свій продукт.

На сьогодні більшість передових компаній світу, які спеціалізуються на розробці комерційного програмного забезпечення впроваджують та рекомендують послуги надані AWS, так як даний хмарний провайдер пройшов достатню кількість атестацій у сфері безпеки.

Хмарна платформа Amazon, яка створена в 2006 році стала першовідкривачем в даній області, завдяки чому отримала значну долю ринку. З постійними нововведеннями та покращеннями протягом багатьох років, AWS ввела більше 70 послуг з широким відсотком покриття по всьому світу. Серверни доступне в 14 географічних регіонах. Відсоток компанії на ринку нестримно зростає, у другому кварталі 2020 року хмарні технології компанії Amazon займали 33% ринку.

В сучасних реаліях постає необхідність безпечно розгортати інфраструктуру в Amazon Web Services через використання програмного коду.

Це стосується налаштування та керування обчислювальними та мережевими ресурсами за допомогою програмних продуктів, який набагато швидший, зручніший та безпечніший на відміну від внесення необхідних налаштувань обладнання власноруч чи з допомогою додаткових інструментів.

Поряд з перевагами хмарних обчислень мова йде і про ризики, з якими пов'язаний перехід на "хмари", найістотніший з яких – загроза інформаційній безпеці. Відповідно, основними правовими проблемами, що пов'язані з використанням хмарних обчислень, є забезпечення інформаційної безпеки та захист персональних даних у віртуальному середовищі. Ці важливі проблеми стосуються дотримання конституційних принципів, соціальних проблем, захисту національної безпеки [3]. Важливість та актуальність дослідження випливає і з того, що нормативно-правова база "відстає" від темпів розвитку IT-сфери (і, зокрема, хмарних технологій).

Тому актуальною темою є дослідження та аналіз можливостей сервісів AWS по забезпеченню безпеки в веб–додатках та спробувати за їх допомогою реалізувати повний безпечний життєвий цикл програмного забезпечення.

Об'єкт дослідження – безпека веб–додатків, побудованих за допомогою сервісів AWS.

Предмет дослідження – сервіси AWS для створення веб–додатків.

Мета роботи –моделювання та побудова безпечного веб–додатку за допомогою сервісів клаудпровайдера AWS.

Методи дослідження – вивчення документації сервісів, які використовуються при побудуванні веб-додатків, перегляд інструкцій інтегрування сервісів з власним додатком, написання програмного кода на мові програмування Python та мові конфігурації хмарних провайдерів Terraform (Terragrunt). Спроба створення власного secure CI/CD (SDLC).

1 КОНЦЕПЦІЇ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ В ХМАРІ

1.1 Вимоги до безпеки даних в хмарних сервісах

Хмарні обчислення – це модель, яка забезпечує повсюдний та зручний доступ через мережу до спільного набору обчислювальних ресурсів, що підлягають налаштуванню, наприклад, до комунікаційних мереж, серверів, засобів збереження даних, прикладних програм та сервісів тощо. Вони можуть бути швидко надані та звільнені з мінімальними експлуатаційними затратами або зверненням до провайдера. Основними моделями розгортання хмарних сервісів є такі: приватна, громадська, публічна та гібридна хмари. Ознакою такої класифікації є категорії користувачів, що мають доступ та можуть використовувати ресурс та дані хмарного сервісу. При цьому постачальники хмарних ресурсів можуть надавати, а користувачі отримувати такі послуги, як: програмне забезпечення як послуга (SaaS), платформа як послуга (PaaS) та інфраструктура як послуга (IaaS)[22]. Аналіз показує, що незалежно від моделі розгортання та обслуговування хмари, всі ключі, що використовуються в середовищі хмарних обчислень, можна поділити за призначенням та власником на такі два класи :

– ключі, що використовуються провайдером хмарних послуг та є його власністю;

– ключі, що використовуються клієнтами провайдера хмарних послуг та є їх власністю.

Наприклад, якщо хмара розгорнута як публічна та надає послуги PaaS, то користувач хмари на основі сервісу, що надається провайдером, реалізує свої рішення, послугами якого користуються клієнти користувача. За цих умов користувач для своїх клієнтів буде виступати в якості провайдера хмарних послуг, а отже в цьому випадку будуть існувати також два класи ключів: – клас ключів провайдера хмарних послуг, до якого відносяться ключі провайдера

хмарних послуг публічної хмари, що надає послуги PaaS та ключі користувача хмари, по відношенню до клієнтів користувача; – клас ключів користувача хмарних послуг, до якого відносяться по відношенню до провайдера хмарних послуг, ключі користувача хмарних послуг та ключі клієнтів користувача хмарних послуг. Аналогічним чином можна виділити та показати існування лише двох класів ключів для інших моделей розгортання та надання послуг в хмарі [18]. Така модель хмари, у якій існує тільки дві ролі – користувач та провайдер дозволяє зменшити складність аналізу безпеки управління ключами, включаючи крипто живучість. Це досягається за рахунок виключення ролі аудитора хмари, яка виступає в якості пасивного елементу хмари, та має доступ до хмари лише під час проведення аудиту з використанням строгого переліку доступних можливостей.

Вказане є справедливим і до посередника (брокера) хмарних послуг, який для провайдера хмарних послуг розглядається з точки зору клієнта, а для клієнта брокер є провайдером хмарних послуг. Теж саме можна прийняти і до транспортувальника хмарних послуг – в моделі безпеки його головною задачею повинно бути забезпечення доступності сервісів, забезпечення конфіденційність та цілісність даних, що передаються забезпечується тільки користувачем та провайдером хмарних послуг.

Ґрунтуючись на наведеному, розглянемо модель порушника хмарних обчислень, на основі якої побудуємо модель загроз відносно управління ключовими даними.

1.2 Модель порушника

При побудові моделі порушника застосовується методика, що ґрунтує- ться спочатку на побудові моделі порушника, виявленні усіх можливих загроз та визначенні способів їх реалізації, а на останок – побудові моделі загроз. Побудовані моделі порушника та загроз дозволяють сформулювати вимоги до системи захисту інформації в середовищі хмарних обчислень. Під моделлю

Порушника будемо розуміти абстрактний формалізований чи неформалізова-ний опис дій порушника, який відображає його практичні та теоретичні можливості, апріорні знання, час та місце дії тощо (рис. 1.1). Складність побудови моделі порушника для інформаційно-телекомунікаційної системи (ІТС) хмарних обчислень полягає в необхідності врахування моделі розгортання хмари, моделі надання послуг, власника та рівень контролю інформації, а також рівень контролю провайдера та користувачів над інфраструктурою хмари. Також така модель має бути ще адекватною реальному порушнику. Аналіз моделі хмари NIST показує, що по відношенню до ІТС хмари порушники можуть бути внутрішніми (з числа співробітників, користувачів системи) або зовнішніми (сторонні особи) [5]. Особливу небезпеку в середовищі хмарних обчислень становлять внутрішні порушники. Навіть при наданні послуг на рівні IaaS, внутрішня інфраструктура хмари, в тому числі і середовище передачі даних, контролюється провайдером хмарних послуг.

Рисунок 1.1 – Модель хмарних обчислень

Наведена на рис. 1.1 модель визначає:

– категорії осіб, з числа яких може бути порушник;

– припущення про рівень можливостей, кваліфікацію, рівень ознайомлення з системою, характер дій порушника;

– методи та засоби, що може використовувати порушник;

– мета, яку перед собою ставить порушник;

– елементи системи, які порушник буде атакувати.

За категорією осіб порушниками можуть бути:

– внутрішні порушники: працівники провайдера хмарних послуг, працівники користувача, сторонні особи, що отримують доступ до ресурсів ІТС хмари. Для їх визначення слід детально розглянути можливості несанкціонованого доступу до ресурсів ІТС кожного із працівників провайдера та користувача, а також можливості сторонніх осіб щодо несанкціонованого доступу до ресурсів ІТС з урахуванням наявної системи організаційного обмеження їх доступу [1];

– зовнішні порушники: інші особи, що не мають безпосереднього доступу до ресурсів ІТС хмари. Для їх визначення слід детально розглянути можливі канали витоку інформації та вразливі місця системи.

За рівнем можливостей порушників розділимо на чотири класи:

– перший рівень припускає можливість запуску фіксованого набору завдань (програм), що реалізують заздалегідь передбачені функції обробки інформації;

– другий рівень визначається можливістю створення і запуску власних програм з новими функціями обробки інформації;

– третій рівень визначається можливістю управління функціонуванням ІТС хмари, тобто впливом на базове програмне забезпечення системи, на склад і конфігурацію її устаткування;

– четвертий рівень визначається всім обсягом можливостей осіб, що здійснюють проектування, реалізацію і ремонт апаратних компонентів ІТС

хмари, з можливістю включення до складу ІТС хмари власних засобів з новими функціями обробки інформації [11].

1.3 Профіль порушника

Розглянемо можливий профіль порушника (рис 1.2).

№	Категорія	Значення (Позначення)
1	Категорія осіб	внутрішній (В)
		зовнішній (З)
2	Характер дій	випадковий (В)
		пасивний (П)
		активний (А)
3	Рівень доступу та можливостей	перший рівень (1)
		другий рівень (2)
		третій рівень (3)
		четвертий рівень (4)
4	Рівень ознайомленості	користувач (К)
		спеціаліст (С)
		адміністратор системи (А)
		спеціаліст з найвищим рівнем знань (Н)
		адміністратор безпеки (Б)
5	Методи та засоби	агентурні (Р)
		штатні (Ш)
		пасивні (П)
		активні (А)
6	Мета дій порушника	отримання атрибутів доступу персоналу чи користувачів АС (О)
		отримання несанкціонованого доступу до обчислювальних ресурсів хмари (НСД)
		проникнення на територію хмарного ЦОД з метою впливу на фізичне обладнання (ВФ)
		зміна режимів функціонування чи виводу з ладу фізичних ресурсів АС (М)
		встановлення засобів технічної розвідки (ТР)
		встановлення технічних засобів нав'язування (ТЗН)
		встановлення програмних закладок розвідки (ПЗР)
		встановлення програмних засобів нав'язування (ПЗН)

Рисунок 1.2 – Категорування профіля порушника відносно хмарних сервісів

За рівнем ознайомлення з системою, порушників будемо класифікувати за такими рівнями, як:

– що не володіють спеціальними знаннями з обчислювальної техніки та програмування, проектування та експлуатації хмарної ІТС, а є лише користувачами сервісу;

– що володіють базовим або високим рівнем знань у галузі обчислювальної техніки та програмування, проектування та експлуатації хмарних ІТС, а також базовим або високим рівнем знать про системи захисту хмарних ІТС;

– що володіють інформацією про функціональні особливості визначеної ІТС хмари, основні закономірності формування в ній масивів даних та потоків запитів до них, вміють користуватися штатними засобами;

– що володіють високим рівнем знань та досвідом роботи з технічними засобами системи хмари та їхнього обслуговування;

– що володіють інформацією про функції та механізм дії засобів захисту в визначеній ІТС хмари.

Аналіз показує, що незалежно від моделі розгортання хмари та моделі надання послуг найнебезпечнішими порушниками в ІТС хмари є адміністратори хмари та адміністратори безпеки хмари. Порушення з боку цих осіб можуть бути як ненавмисними, так і зловмисними. При цьому якщо «випадковий порушник» здійснює загрози ІТС під час виконання своїх функціональних обов’язків внаслідок помилкових дій, за рахунок неуважності чи недбалості, то порушник, що здійснює зловмисне порушення, чітко розуміє свої дії та має змогу проаналізувати їх вплив. Також необхідно брати до уваги, що порушник в середовищі хмари може здійснювати активні чи пасивні загрози ресурсам ІТС чи АС. Під активною загрозою слід розуміти навмисні та не навмисні несанкціоновані дії порушника, що призвели до зміни стану ІТС (АС), а під пасивною загрозою – дії, що призвели до несанкціонованого проникнення в систему без зміни її стану.

За характером дій порушників можна розділити на 4 типи.

1. «Випадковий порушник» – це користувачі, обслуговуючий персонал, які не навмисно подолали засоби управління (адміністрування) доступом до об’єкту

захисту, виконали непередбачені дії відносно цього об’єкту, чим спричинили порушення політики безпеки до об’єкта захисту;

2. «Пасивний порушник» – авторизований користувач, або обслуговуючий персонал хмари, який навмисно порушив політику безпеки послуги, але не вживає рішучих дій. З метою прихованого подолання засобів управління (адміністрування), використовує атрибути доступу інших користувачів;

3. «Активний порушник» – порушник, який не приховує своїх дій та може використовувати всі доступні методи та засоби для порушення властивості захищеної інформації. До таких дій відносяться дії, що спрямовані на подолання організаційного обмеження доступу, охоронної сигналізації, управління доступом до фізичних ресурсів, фізичний доступ до засобів оброблення, зберігання чи передавання інформаційних об’єктів з метою виведення їх з ладу, зміни режимів функціонування, крадіжки чи пошкодження носіїв тощо;

4. «Віддалений порушник» – порушник, що використовує засоби віддаленого доступу до інформаційних об’єктів: виток інформації технічними каналами, спеціальний вплив на інформацію по технічним каналам, мережне обладнання локальних чи розподілених мереж, в тому числі і засоби телекомунікаційних мереж.

За методами та засобами, що використовує порушники, їх можна класифікувати як таких, що використовують:

– виключно агентурні методи одержання відомостей;

– пасивні технічні засоби перехоплення інформаційних сигналів;

– для реалізації спроб НСД виключно штатні засоби АС або недоліки проектування КСЗІ;

– способи і засоби активного впливу на АС, що змінюють конфігурацію системи (підключення додаткових або модифікація штатних технічних засобів, підключення до каналів передачі даних, впровадження і використання спеціального ПЗ тощо).

Також при подальших дослідженнях будемо вважати, що метою зловмисника, при здійсненні порушення, може бути наступне:

– авторизація та отриманні атрибутів доступу з найбільшими правами до ресурсів ІТС з метою їх використання для ознайомлення з конфіденційною інформацією, її модифікації чи знищення, використання обчислювальних ресурсів хмари в своїх власних цілях;

– пошук та/або здобуття атрибутів доступу особи з персоналу чи користувачів АС, які мають атрибути доступу з найбільшими правами, з метою заволодіння їх атрибутами доступу. Здобуття атрибутів особи може бути реалізовано шляхом використання технічних засобів, крадіжок, купівлі, чи отримання іншим шляхом;

– проникнення на місця розміщення тих чи інших компонентів, елементів чи ресурсів АС (обчислювальних ресурсів, інформаційних ресурсів, базового, прикладного програмного забезпечення та програмного забезпечення системи ТЗІ, включаючи носії резервних копії, ресурсів вводу/виводу, телекомунікаційного обладнання, включаючи мережу передачі даних) шляхом подолання перешкод (огорожі, елементів будівельних конструкцій, охорони чи охоронної сигналізації та ін.) та нанесення збитків шляхом знищення матеріальних та інформаційних цінностей;

– зміна режимів функціонування чи виводу з ладу ресурсів АС;

– встановлення фізичних засобів (апаратурних закладок) чи інших засобів технічної розвідки в місцях розміщення елементів АС (в тому числі і віддалених, наприклад в елементах комунікаційної мережі зв’язку) для знімання інформації;

– встановлення фізичних чи інших засобів (апаратурних закладок) в місцях розміщення елементів АС (в тому числі і віддалених, наприклад, в елементах комунікаційної мережі зв’язку) для генерації несправжніх сигналів, інформаційних символів чи повідомлень;

– встановлення програмних засобів (програмних закладок) знімання інформації з метою її того чи іншого використання;

– встановлення програмних засобів (програмних закладок чи вірусів) для модифікації як програмних засобів, так і інформації АС, шляхом генерації (впровадження) програмних вірусів, несправжніх сигналів, інформаційних

символів чи повідомлень з метою перевантаження систем АС і порушення, таким чином, доступності компонентів АС чи АС в цілому;

– здійснення спроб несанкціонованого доступу до обчислювальних ресурсів, інформаційних ресурсів, базового та прикладного програмного забезпечення та програмного забезпечення системи ТЗІ як власне АС, так і її телекомунікаційної підсистеми шляхом подолання системи управління доступом [16].

Важливим також є визначення та використання в моделі того, яким чином загрози можуть бути реалізовані при хмарних обчисленнях. Ґрунтуючись на, будемо вважати, що вони реалізуються наступними способами:

– технічними каналами, що включають канали побічних електромагнітних випромінювань і наводок, акустичні, віброакустичні, акустоелектричні, оптичні, радіо- та радіотехнічні, хімічні та інші канали;

– каналами спеціального впливу шляхом формування полів і сигналів з метою руйнування системи захисту або порушення цілісності інформації;

– несанкціонованим доступом шляхом підключення до апаратури та ліній зв’язку, маскування під зареєстрованого користувача, подолання заходів захисту з метою використання інформації або нав’язування хибної інформації, застосування закладних пристроїв чи програм та вкорінення комп’ютерних вірусів, як на стороні користувача так і провайдера хмарних послуг.

1.4 Методи захисту від загроз в хмарі

Запропоновані на основі аналізу сучасного стану стандартизації та застосування хмарних сервісів моделі хмарних обчислень, порушника та загроз хмарних сервісів дозволили встановити, що найбільш проблемними та такими, що вимагають вирішення в частині надання послуг конфіденційності, цілісності, справжності та доступності тощо, є задачі захисту ключів та ключової інформації. Для цього на основі аналізу стану встановлено, що в середовищі хмари відносно ключових даних існують та можуть бути реалізованими такі загрози як

компрометація, несанкціоноване знищення, перехоплення та запам’ятовування, нав’язування слабких та несанкціоноване використання тощо ключів. При цьому встановлено, що найбільшу небезпеку в середовищі хмарних обчислень для ключових даних користувача представляють адміністратори хмарних сервісів, які мають доступ до середовища, в якому розгорнуто хмарні додатки користувача [1].

Також на основі детального аналізу стану та вимог відносно безпечності управління інформаційною безпекою зі сторони нормативно-правових документі та стандартів. Вони зводяться до використання для забезпечення високого рівня безпеки, тобто високого рівня ймовірностей реалізації загроз в середовищі хмарних обчислень, комплексу технічних, організаційних та організаційно-технічних заходів та засобів, в тому числі до використання:

– на рівні користувача захищених з необхідним рівнем безпеки ключових носіїв;

– на рівні каналів зв’язку між користувачем та хмарою захищених каналів зв’язку з взаємною автентифікацією сторін та стійкістю вищою за стійкість ключів, що передаються;

– на рівні сервісів ідентифікації, автентифікації, авторизації та керування правами доступом надійних протоколів автентифікації з стійкими криптографічними алгоритмами.

Відштовхуючись від вищезазначених пунктів визначимо, що найоптимальнішим варіантом буде використання IAC (інфраструктура як код) інструментів. Подібні інструменти використовують ключі шифрування та дозволяють керувати хмарною інфраструктурою через запити у вигляді коду. Такий підхід мінімізує кількість осіб, які мають адміністративний доступ до хмари та час, за який відбувається побудова та редагування інфраструктури. Результатом зменшення часу розгортання інфраструктури є зменшення імовірність перехоплення даних [13].

2 AWS

2.1 Основні властивості Amazon Web Services

Amazon Web Services (AWS) - найбільш всеосяжна і широко використовувана хмарна платформа у світі, яка стала гігантським компонентом бізнес-портфеля Amazon. Так само, як Amazon.com вразили простір онлайн-торгівлі, AWS вразив світ обчислювальної техніки. Amazon Web Services, що включає кілька продуктів і сервісів для хмарних обчислень, таких як сервери, сховища і мережі, став лідером для платформ хмарних обчислень, випереджаючи конкурентів, таких як Microsoft Azure та Google Cloud Platform [1].

Amazon Web Services (AWS), дочірня компанія Amazon.com, пропонує недорогі послуги хмарних обчислень, що дозволяють зміцнити свою клієнтську базу від невеликих компаній, таких як Pinterest, до великих підприємств, таких як D-Link.

Хмарні обчислення - це використання віддалених серверів в Інтернеті для зберігання, керування та обробки даних, а не локального сервера або персонального комп'ютера. Хмарні обчислення поділяються на три основні категорії, починаючи з програмного забезпечення як послуги (SaaS), яке скорочує витрати компанії, надаючи хмарному серверу вже включене програмне забезпечення. Платформа як послуга (PaaS) дозволяє розробникам створювати програми та спільно працювати над різними проектами без необхідності купувати або підтримувати інфраструктуру. А також Інфраструктуру як послугу (IaaS), яка підпадає під AWS більше, ніж інші. IaaS надає компаніям можливість орендувати сервери, сховища, безпеку і т.д. У хмарного провайдера [6].

2.2 Загальна характеристика послуг безпеки AWS

AWS забезпечує безліч засобів контролю, щоб захистити робочі навантаження клієнтів, і досить часто клієнти не знають про свою частку відповідальності за безпеку та необхідний контроль за безпекою користування та розміщення своїх ресурсів в хмарі AWS. AWS надає безліч послуг, інструментів і методів, таких як контроль доступу, брандмауер, шифрування, реєстрація, моніторинг, відповідність тощо, щоб забезпечити безпеку у хмарі. Ці служби AWS підтримують безліч випадків та сценаріїв, щоб дотриматись усіх вимог щодо безпеки, реєстрації користувачів, аудиту та дотримання вимог у хмарному середовищі. Існує послуга AWS Identity and Access Management (IAM), яка дозволяє вам контролювати безпечний доступ та дії для ваших користувачів AWS, Virtual Private Cloud (VPC) дозволяє захистити свою інфраструктуру в хмарі AWS шляхом створення віртуальної мережі до вашої приватної мережі у локальному центрі обробки даних. Більше того, існують веб-сервіси, такі як Служби управління ключами (KMS), що використовуються для управління ключами шифрування для додатуового захисту ваших даних. Існує AWS Shield та веб-брандмауер веб-додатків AWS (WAF) для захисту ваших ресурсів AWS та програм від загальних загроз безпеці, таких як розподілена відмова в обслуговуванні (DDoS) шляхом налаштування брандмауерів на різних рівнях. AWS Config разом із AWS CloudTrail та AWS CloudWatch підтримує аудит та управління конфігурацією для всіх ваших ресурсів AWS. Артефакт AWS - це керований сервіс самообслуговування, який надає вам документи про відповідність на вимогу вашого аудитора [11].

2.3 Обов’язки AWS щодо безпеки

AWS відповідає за забезпечення глобальної інфраструктури, що включає регіони та зони доступносі, що працюють на хмарі AWS. Ці зони доступності містять кілька центрів обробки даних, в яких розміщується обладнання,

програмне забезпечення, мережі та інші ресурси, на яких розгорнуті послуги AWS. Забезпечення цієї інфраструктури є першочерговим завданням AWS, і AWS регулярно перевіряється відомими агенціями по всьому світу для забезпечення необхідної безпеки та відповідності стандартам. Ці звіти про аудит доступні для клієнтів AWS, оскільки клієнти не можуть особисто відвідувати центри обробки даних AWS.

Дані клієнтів та робочі навантаження зберігаються в центрах обробки даних AWS, ці центри обробки даних поширюються в географічних регіонах по всьому світу. Ці центри обробки даних належать, експлуатуються та контролюється AWS. Цей контроль включає фізичний доступ та вхід до цих центрів обробки даних і всі мережеві компоненти та обладнання, а також усі інші додаткові центри обробки даних, які є частиною глобальної інфраструктури AWS.

Отже, найперша думка, яка вразить кожного, хто роздумує про перенесення свого навантаження хмара, де фактично зберігаються мої дані? Де фізичні сервери та жорсткі диски знаходиться, що я створив за допомогою хмари AWS? А як ці апаратні ресурси і хто їх забезпечує? Адже хмара просто віртуалізує всі ресурси, доступні в центрах обробки даних, але ці ресурси є десь фізично. Отже, AWS повністю відповідає за фізичну та логічну безпеку всього обладнання та ресурси, розташовані в центрах обробки даних по всьому світу.

AWS має багаторічний досвід створення, управління та забезпечення великих центрів обробки даних по всьому світу через материнську компанію Amazon. AWS гарантує, що всі його центри обробки даних використовують найкращі технологій та процеси, такі як розміщення їх у непомітному вигляді, 21 дотримання політики найменших привілеїв, відеоспостереження, двофакторна автентифікація для введення дата-центрів та поверхів [8].

Персонал забороняється розміщувати на поверхах ЦОД, якщо у них немає потреби отримати доступ до фізичних пристроїв зберігання даних особисто. Більше того, AWS твердо впроваджує сегрегацію відповідальності, тому будь-який персонал, який має доступ до фізичного пристрою, не матиме доступ кореневого користувача для цього пристрою, тому він не може отримати доступ

до даних на цьому фізичному пристрої.

Це дуже важлива частина моделі спільної відповідальності за безпеку, де AWS робить все щоб турбуватися про фізичну та логічну безпеку центрів обробки даних. Вам не доведеться турбуватися про моніторинг, крадіжки, вторгнення, пожежу, природні лиха, збій живлення тощо для ваших центрів обробки даних. Цими речами опікуються AWS від вашого імені, і вони постійно вдосконалюють свої процедури безпеки, щоб не відставати від зростаючих загроз.

AWS розпочне процес виведення з експлуатації, коли запам'ятовуючий пристрій досягне кінця терміну його корисного використання. Цей процес гарантує, що дані клієнтів не піддаються несанкціонованому доступу інших фізичних осіб. Цей апаратний пристрій буде фізично зруйнований або знешкоджений, якщо він вийде з ладу.

AWS зберігає ваші дані та інші ресурси в центрах обробки даних у різних географічних регіонах по всій земній кулі; ці місця відомі як регіони. У кожному регіоні є дві або більше зон доступності для високої стійкості до несправностей. Ці зони доступності складається з одного або декількох центрів обробки даних. Всі ці центри обробки даних використовуються, і жоден не зберігається офлайн; тобто немає холодних центрів обробки даних. Ці центри обробки даних містять усе фізичні так апаратні ресурси, як сервери, сховища та мережеві пристрої тощо необхідні для підтримання роботи всіх служб AWS відповідно до угоди про рівень обслуговування. Усі основні програми AWS, такі як обчислення, зберігання, бази даних, мережі розгортаються у конфігурації N + 1, так що, у випадку пошкодження центру обробки даних внаслідок стихійного лиха, людської помилки чи будь-якої іншої непередбаченої обставини є достатня потужність для балансування навантаження на решту сайтів [15].

Кожна зона доступності розроблена як незалежна зона відмов, щоб врахувати можливі збої в роботі. Вони фізично відокремлені в межах географічного розташування і розташовані в верхній частині рівнин.

На рисунку 2.1 показані типові регіони з їх зонами доступності.

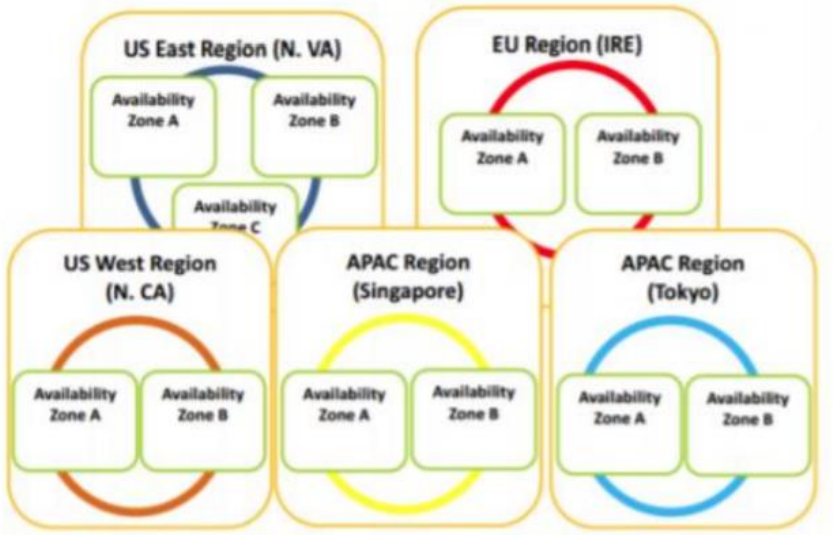


Рисунок 2.1 – Типові AWS регіони з їх зонами доступності

Залежно від характеру вашого бізнесу, дотримання нормативних вимог, аварійне відновлення, відмовостійкість тощо, ви можете вирішити розробити свої програми, які будуть розподілені по кількох регіонах, щоб вони були доступними, навіть якщо один регіон недоступний.

2.4 Переваги AWS

Даний хмарний провайдер надає багато серйозних переваг, які чітко відрізняють його від інших, які тільки набирають популярність. Розглянемо переваги AWS.

1. Безпека та надійність. AWS має сервери, розташовані в 76 зонах доступності, що охоплюють 245 країн та територій. Ці зони були розділені не тільки для того, щоб надати користувачам можливість встановлювати географічні обмеження для своїх послуг, але й для диверсифікації фізичних місць розташування, забезпечення більш високої безпеки та запобігання постійної втрати даних у всьому світі. Кожен центр обробки даних по всьому світу підтримується та постійно контролюється. Можна стверджувати, що Amazon Web Services набагато безпечніший, ніж компанія, яка надає власний веб-сайт або

сховище. Це пов'язано з тим, що локалізація даних в одному місці, що легко ідентифікується, може виявитися вразливою. AWS надає доступ до своїх фізичних центрів обробки даних лише на потрібній основі. Це крім того, що ці місця приховані від громадськості, створює сильніший бар'єр для фізичних вторгнень. Відключення та потенційні атаки на сервери також безпечніші завдяки досвіду Amazon з хмарними сервісами. Цілодобовий інтенсивний моніторинг дозволяє швидко ідентифікувати будь-яку атаку та впоратися з нею [1]. На рисунку 2.2 зображено базові блоки, за допомогою яких забезпечується безпека сервісів.

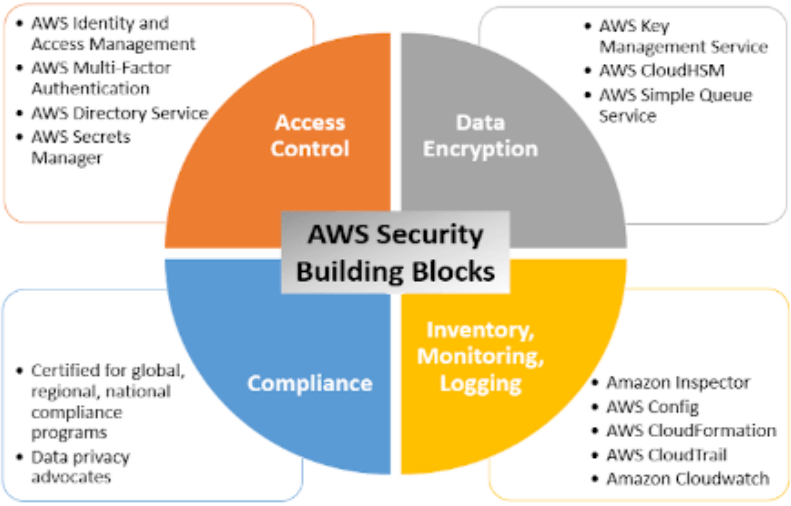


Рисунок 2.2 – Блок, з яких будується безпека AWS

2. Масштабованість. Початківці та малі підприємства отримують очевидні переваги від масштабованості, яку пропонує Amazon Web Services. Вартість змінюється в залежності від використання клієнтів, що робить AWS чудовим вибором для побудови бізнесу знизу нагору. Цей бізнес отримає всі інструменти, необхідні для створення стартапу з хмарою. Вже створені компанії отримають зиск від недорогих сервісів міграції Amazon, що дозволяють плавно перевести інфраструктуру на AWS. AWS продовжує надавати ресурси з метою розширення, оскільки бізнес продовжує зростати завдяки гнучкій бізнес-моделі використання. Клієнтам більше не потрібно повертатися до обговорення того, чи

потрібен їхній поточний бюджет для використання комп'ютера. У деяких випадках, коли йдеться про масштабованість AWS, компанія може використовувати AWS, щоб встановити і забути. На рисунку 2.3 зображена базова модель масштабованості серверних машин [2].

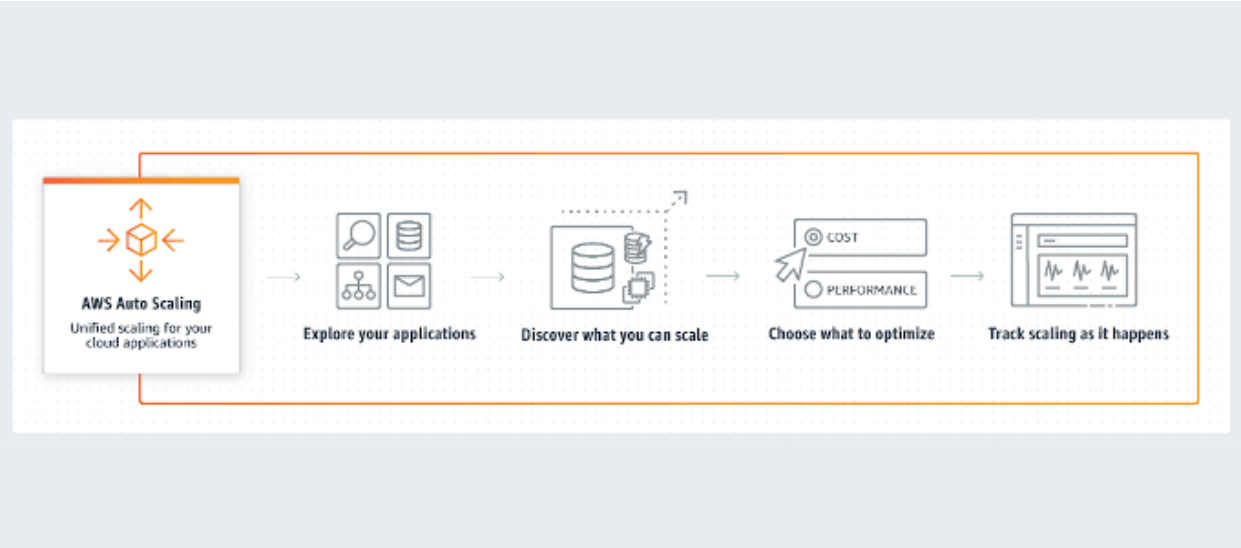


Рисунок 2.3 – Базова модель масштабованості серверних машин

3. Вартість. Раніше компаніям доводилося фізично створювати простір для зберігання, щоб зберігати великі обсяги даних та обслуговувати їх самостійно. Вибір зберігання даних у хмарі коштував значних витрат у зв'язку з тривалими контрактами, які могли перешкодити зростанню компанії. Купівля більшої кількості місця, ніж необхідно, може стати занадто дорогою, тоді як покупка занадто малої кількості може виявитися катастрофічною. Той самий принцип застосовується до обчислювальної потужності. Підтримка бізнесу в піковий час вимагала, щоб компанії купували більше електроенергії. Проблема була в періоди непікового навантаження, коли споживання більше не потрібно. Компанії все одно доведеться платити за електроенергію, яку вони навіть не використали. AWS дозволяє компаніям платити лише за те, що вони використовують. За відсутності початкових витрат на зберігання та інфраструктуру більше немає потреби оцінювати використання порівняно з витратами. AWS автоматично масштабує витрати відповідно до потреб своїх клієнтів [17].

Головні переваги клаудпровайдера AWS зображені на рисунку 2.4.

Рисунок 2.4 – Головні переваги клаудпровайдера AWS

В результаті проведення дослідження базової інформації клаудпровайдера AWS та вивчення документації були відокремлені та перераховані найголовніші переваги даного сервісу серед інших. В ході роботи було виявлено, що забезпечення безпеки – це одна з найсильніших сторін Amazon Web Services.

3 СЕРВІСИ ДЛЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В ХМАРНИХ ТЕХНОЛОГІЯХ

3.1 Технологія для конфігурації хмарних сервісів

Terraform – це інструмент з відкритим кодом від компанії HashiCorp. Він дозволяє описувати інфраструктуру у вигляді коду простою декларативною мовою та розгортати її/керувати нею у різних публічних хмарних сервісах (скажімо, Amazon Web Services, Microsoft Azure, Google Cloud Platform, DigitalOcean), а також приватних хмарах та платформах віртуалізації (OpenStack, VMWare та ін) всього кількома командами. Наприклад, замість того, щоб вручну клацати кнопкою миші на веб-сторінці або вводити десятки команд в консоль, ви можете скористатися наступним кодом і налаштувати сервер в AWS (рис. 3.1).

```
resource "aws_instance" "simple_server" {
  name          = "${var.developer}-simple-server"
  ami           = var.image_id
  instance_type = var.instance_type
}
```

Рисунок 3.1 – Terraform код для створення серверу в AWS

Завдяки своїй простоті та потужності Terraform став ключовим гравцем у світі DevOps. Він дозволяє замінити громіздкі, тендітні та неавтоматизовані засоби управління інфраструктурою на надійний автоматизований інструмент, поверх якого є можливість об'єднувати всі інші, доступні елементи DevOps (автоматичне тестування, безперервну інтеграцію та безперервне розгортання) та супутній інструментарій (наприклад Docker, Chef, Puppet) [24].

В ході виконання даної роботи використовувалися такі мови програмування як:

– Python. Для написання backend-частини веб-додатка;

– Terraform & Terragrunt. Для створення конфігураційних файлів для оркестрації веб-сервісів AWS. Саме програмний код даної мови зустрічається найбільше на скріншотах в даній роботі.

3.2 Сервіс для ідентифікації та доступу

AWS IAM - управління ідентифікацією та доступом AWS (Identity and Access Management). Сервіс надає можливість безпечного доступу до ресурсів та сервісам AWS. За допомогою IAM можна створювати користувачі та групи користувачів, а також керувати ними, використовуючи дозволи («permissions»), для надання або заборони доступу до ресурсів.

IAM також дозволяє додавати особливі умови, при дотриманні яких користувач зможе використовувати AWS, наприклад час доби, IP-адреса, можливість використання протоколу SSL або необхідність використання багатофакторної аутентифікації [1].

За допомогою IAM можна аналізувати права доступу до сервісів середовища AWS. Команди по забезпеченню безпеки і адміністратори можуть швидко перевірити, що згідно з правилами публічного і міжакаунтного доступу до ресурсів, доступ отримають тільки користувачі з відповідними правами. Також можна легко визначити і уточнити свої правила, щоб вони дозволяли доступ тільки до активних сервісів. Завдяки таким функціям буде простіше дотримуватися принцип мінімальних привілеїв.

За допомогою дозволів можна надавати доступ до ресурсів AWS. Дозволи надаються об'єктам IAM (користувачам, групам і ролям), при цьому, за замовчуванням, у об'єктів відсутні будь-які дозволи. Іншими словами, об'єкти IAM не можуть виконувати ніяких дій на платформі AWS, поки їм не надані необхідні права. Щоб надати об'єктам дозволу, можна призначити правило, яке буде визначати тип доступу, дії, які можуть бути виконані, а також ресурси, на яких можуть виконуватися певні дії. Крім того, можна вказати будь-які умови, які повинні виконуватися для дозволу або заборони доступу.

Щоб встановити дозволи для користувача, групи, ролі або ресурсу, потрібно створити політику (policy). Розглянемо з чого складається політика.

1. Actions. Дозволені дії для сервісу AWS. Наприклад, можна дозволити користувачеві викликати дію Amazon S3 ListBucket. Будь-які дії, які ви явно не вказуєте виконувати, забороняються.

2. Resources. Ресурси AWS, для яких дозволено виконувати певні дії. Наприклад, список кошиків Amazon S3, для яких ви дозволяєте користувачеві виконувати дію ListBucket. Його користувачі не можуть отримати доступ до тих ресурсів, для яких ви явно не надаєте дозволу.

3. Effect. Дозволяти або забороняти доступ. Оскільки за замовчуванням доступ заборонений, користувач зазвичай створюєте правила, які дозволяють певні дії.

4. Conditions. Умови, які повинні виконуватися, щоб застосовувалася політика. Наприклад, можна дозволити доступ тільки до певних кошиків S3, якщо користувач підключається з певного діапазону IP-адрес або використовує при вході в систему багатофакторну аутентифікацію.

Політики створюються за допомогою візуального редактора в консолі або в форматі JSON. Політика складається з одного або декількох виразів, кожне з яких описує один набір дозволів. Візуальний редактор надає можливість створення та надання дозволів за допомогою правил IAM без необхідності писати правила в форматі JSON (при цьому можливість створювати і редагувати правила в JSON зберігається) в графічному інтерфейсі веб консолі. Правило, продемонстроване на рисунку 3.2, було створено за допомогою візуального редактора. Воно надає дозвіл на п'ять дій Amazon S3 типу List і Read для кошика S3 і об'єктів в SampleBucket, префікс яких починається з MyPrefix.

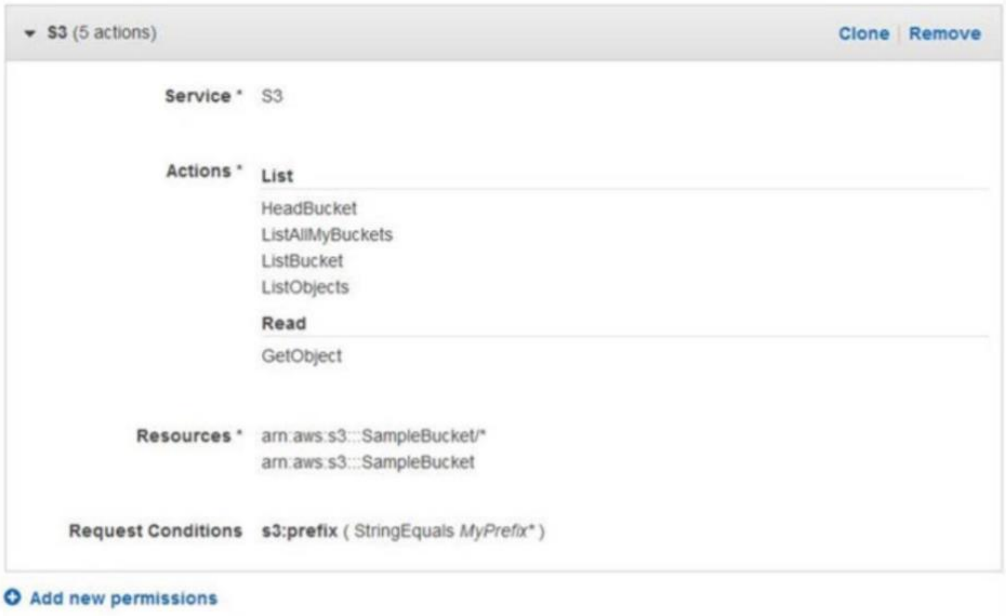


Рисунок 3.2 – Приклад створення правил доступу до ресурса за допомогою AWS IAM

Розглянемо основні складові сервісу IAM.

1. Користувачі. Сутність, створена для представлення людини чи сервісу, які використовують її для взаємодії з хмарним середовищем. Користувач визначається по імені та має набір облікових даних для доступу до хмари (секретний ключ та ключі доступу).

2. Ролі. Сутність, що відображає тип доступу, який може знадобитися окремій особі, машині або сервісу. Роль схожа на користувача, але передбачається, що її може прийняти на себе будь-хто, кому потрібні привілеї ролі.

3. Політики. Це правило, реалізоване у коді, яке явно чи неявно надає доступ до хмарних сервісів, ресурсів чи об'єктів або забороняє його. Політики прив'язані до ролей чи користувачів, вони дозволяють чи обмежують їхні дії серед.

4. Тимчасові облікові дані безпеки. Сервіс, що надає тимчасові облікові дані з обмеженими привілеями для користувачів IAM, об'єднані або автентифіковані сервісом IAM.

Повна схема сервісу ідентифікацією та доступом зображено на рисунку 3.3.

Рисунок 3.3 – повне зображення сервісу AWS IAM

3.3 Віртуальна приватна мережа

3.3.1 Virtual Private Cloud

Amazon Virtual Private Cloud (Amazon VPC) – це сервіс, який дає можливість запускати ресурси AWS в обумовленій користувачем логічно ізольованій віртуальній мережі. Це дозволяє повністю контролювати середу віртуальної мережі, в тому числі вибирати власний діапазон IP-адрес, створювати підмережі, а також налаштовувати таблиці маршрутизації і мережеві шлюзи. Для більшості ресурсів в Virtual Private Cloud можна використовувати і IPv4, і IPv6, і таким чином отримати безпечний і зручний доступ до ресурсів і додатків.

Як один з основних сервісів AWS, Amazon VPC спрощує індивідуальну налаштування параметрів мережі. Можна створити загальнодоступну підмережу для власних веб-серверів з доступом до Інтернету. Можна також помістити свої серверні системи, такі як бази даних або сервери додатків, в приватну підмережа без доступу до Інтернету. Amazon VPC дає можливість використовувати багаторівневу систему безпеки, яка складається з груп безпеки та мережевих списків контролю доступу. Така система дозволяє контролювати доступ до інстанси Amazon EC2 в кожній підмережі [9].

AWS VPC надає наступні можливості для налаштування власних мереж:

– вибирати діапазон використовуваних IP-адрес;

– створювати підмережі;

– налаштовувати маршрутизацію;

– створювати мережеві шлюзи;

– створювати Security groups і NACL (Network Access Control List).

Розглянемо з чого складається Amazon VPC.

1. Virtual Private Cloud – логічно ізольована віртуальна мережа в хмарі AWS. Простір IP-адрес хмари VPC задається з обраних клієнтом діапазонів адрес. Код для створення віртуальної мережі зображений на рисунку 3.4.

```
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = "true"
  tags                 = {
    Name = "Default vpc. Developer - ${var.developer}."
  }
}
```

Рисунок 3.4 – Код для створення віртуальної мережі

2. Підмережа – сегмент діапазону IP-адрес VPC, в якому можна розмістити групи ізольованих ресурсів. Код для створення публічної підмережі зображений на рисунку 3.5.

```
resource "aws_subnet" "public_subnet" {
  count                   = length(var.availability_zones)
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.public_cidr_block
  map_public_ip_on_launch = true
  availability_zone       = "${var.aws_region}${var.availability_zones[count.index]}"

  tags = {
    Name = "Default public subnet. Developer - ${var.developer}."
  }
}
```

Рисунок 3.5 – Код для створення публічної підмережі

3. Інтернет-шлюз – сторона Amazon VPC при підключенні до публічного Інтернету. Код для створення інтернет-шлюза зображений на рисунку 3.6.

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

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

Рисунок 3.6 – Код для створення інтернет-шлюза

4. Шлюз NAT – високодоступний керований сервіс трансляції мережевих адрес (NAT) для забезпечення доступу до Інтернет-ресурсів в приватних підмережах. Код для створення NAT-шлюза зображений на рисунку 3.7.

```
resource "aws_nat_gateway" "nat" {
  count         = length(aws_subnet.public_subnet.*.id)
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = element(aws_subnet.public_subnet.*.id, count.index)
  depends_on    = [aws_internet_gateway.igw]

  tags = {
    Name = "nat"
  }
}
```

Рисунок 3.7 – Код для створення NAT-шлюза

5. Шлюз віртуальної приватної мережі – сторона Amazon VPC при VPN підключенні.

6. Пірингове підключення дозволяє маршрутизувати трафік між двома одноранговими VPC за допомогою приватних IP-адрес.

7. Адреси VPC забезпечують приватний доступ до сервісів, розміщених на AWS, з VPC без використання інтернет-шлюзу, VPN, пристроїв для трансляції мережевих адрес (NAT) або проксі-серверів брандмауера.

8. Вихідний інтернет-шлюз – шлюз з фіксацією стану для IPv6-трафіку, що представляє доступ на вихід із хмари VPC в Інтернет.

Для забезпечення безпеки інстансів в Amazon VPC можна використовувати групи безпеки Amazon EC2. Групи безпеки VPC дозволяють вказати як вхідний, так і вихідний мережевий трафік, дозволений для обміну з інстансами Amazon EC2. Трафік, явно не дозволений для обміну з інстансами, автоматично блокується [4].

Крім груп безпеки, за допомогою списків контролю доступу (ACL) до мережі можна дозволити або заборонити мережевий трафік (вхідний та / або вихідний) кожної підмережі.

Групи безпеки VPC визначають трафік, дозволений для обміну з інстансами Amazon EC2. Мережеві ACL працюють на рівні підмережі і виконують оцінку вхідного і вихідного трафіку. Мережеві ACL можна використовувати для створення правил які дозволяють, так і забороняють певний тарфік. Мережеві ACL не фільтрують трафік для інстансів в рамках однієї підмережі. Крім того, мережеві ACL виконує фільтрацію без фіксації стану, а групи безпеки виконують фільтрацію з фіксацією стану [7].

На рисунку 3.8 зображено схему роботи AWS VPC з використанням підмереж в різних зонах доступності в певному регіоні.

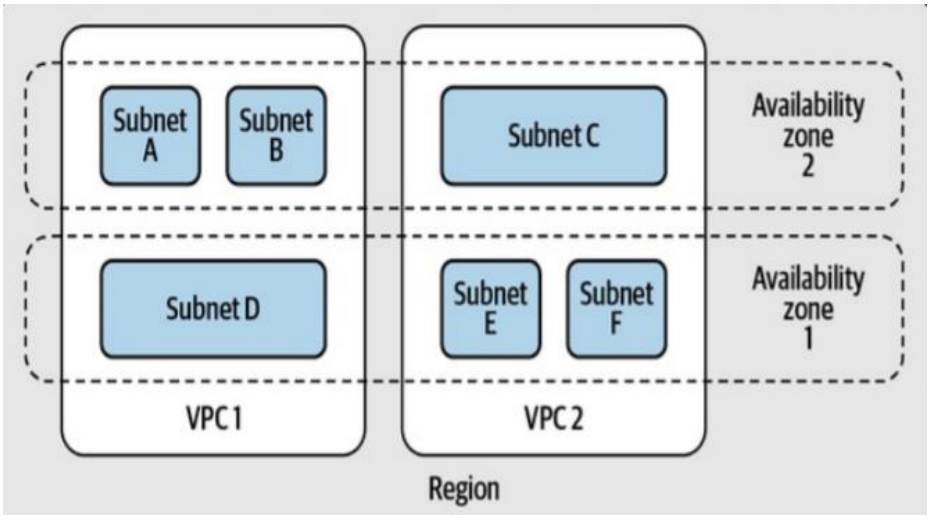


Рисунок 3.8 – Схему роботи AWS VPC з використанням підмереж в різних зонах доступності в певному регіоні

3.3.2 Перетворення мережевих адрес

При проектуванні мереж зазвичай застосовуються приватні IP-адреси 10.0.0.0/8, 172.16.0.0/12 та 192.168.0.0/16. Їх використовують у середині мережі організації для підтримки локального взаємодії між пристроями, та не використовують для маршрутизації до всесвітньої мережі. Щоб пристрій з адресою IPv4 міг звернутися до інших пристроїв або ресурсів через Інтернет, його приватна адреса має бути перетворена на публічну та загальнодоступну. Таке перетворення — це головне, що робить NAT.

Network Address Translation - технологія перетворення приватних IP-адрес на публічні в IPv4. Завдяки цьому процесу машина, яка знаходиться в приватній мережі отримує доступ до Інтернету [22].

Приватні IP-адреси, також звані внутрішніми, внутрішньомережевими, локальними або сірими, будь-яка організація має право використовувати на свій розсуд без будь-якої реєстрації у будь-якої організації. Щоб виходити в Інтернет потрібен білий IP, який «маскуватиме» один або кілька приватних IP-адрес. Механізм NAT саме здійснює заміну (або «маскування») сірих адрес на білі і навпаки.

Таким чином, вся приватна мережа може підключатися до Інтернету через одну публічну IP-адресу (або пул адрес), надану провайдером. В результаті ресурс глобальних адрес витрачається набагато економніше.

Перетворення NAT має важливу особливість з погляду забезпечення безпеки: трансляція приватних IP-адрес в публічні з пулу маршрутизатора, що дозволяє приховати топологію внутрішньої мережі від зовнішніх користувачів, що ускладнює несанкціонований доступ до ресурсів мережі.

Маршрутизатор NAT можна налаштувати з однією або декількома загальнодоступними адресами IPv4, які називають пулом NAT. При відправленні трафіку пристроєм із внутрішньої мережі у зовнішню мережу, маршрутизатор перетворює його внутрішню IPv4-адресу на одну з адрес, що входять до складу пулу. Внаслідок дії такого механізму весь, що виходить із мережі трафік, зовнішні

пристрої «бачать» із загальнодоступною адресою IPv4, яку можна назвати NAT IP адресою.

Розуміючи, як працює NAT, можна виділити комплекс серйозних переваг, які надає цей механізм організації мереж. Розглянемо основні плюси даної технології.

1. Збереження зареєстрованої схеми адресації завдяки дозволу на приватизацію внутрішніх мереж. При використанні PAT можливе використання для зовнішніх з'єднань однієї загальнодоступної IPv4 адреси внутрішніми хостами. Ця схема вимагає малої кількості зовнішніх адрес для підтримки значної кількості внутрішніх хостів.

2. Підвищення гнучкості комунікації з Інтернетом. Забезпечення надійних мережевих підключень досягається завдяки численним пулам адрес, пулам балансування навантаження та резервному копіюванню.

3. Підтримка узгодженої роботи внутрішніх схем мережевої адресації. Якщо мережа не використовує NAT і приватні адреси IPv4, то для зміни загальної схеми адрес доводиться проводити переадресацію всього комплексу хостів. Це може значно підвищувати вартість переадресації. При використанні NAT забезпечується можливість збереження чинної приватної схеми адрес, що дозволяє набагато легше вносити зміни до загальнодоступної схеми адресації. Насправді це означає, наприклад, можливість зміни провайдера компанії без внесення змін у власні внутрішні клієнти (рис. 3.8).

4. Підтримка високого рівня мережі безпеки. При використанні NAT приватні мережі не транслюють свою внутрішню топологію та адреси, що підвищує їхню надійність. При цьому потрібно враховувати, що NAT не є заміною рішень мережевої безпеки, наприклад брандмауера (рис. 3.9).

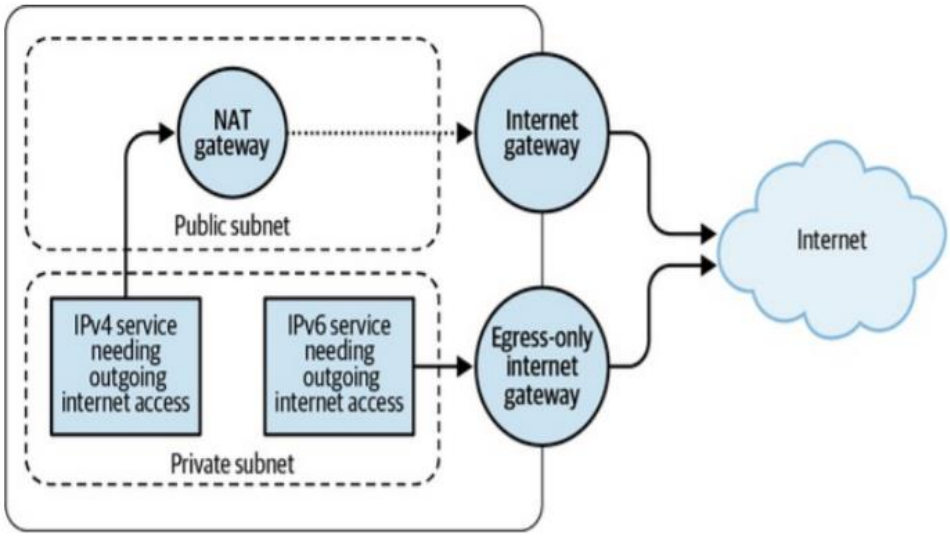


Рисунок 3.9 – Загальна схема технології NAT, реалізована в хмарі провайдера AWS

В ході виконання практичної реалізації – було створено ізольовану віртуальну мережу в хмарі провайдера AWS, яка реалізує в собі технологію мережевого захисту NAT.

3.4 Система авторизації користувачів

3.4.1 AWS Cognito

AWS Cognito – це служба ідентифікації користувачів та синхронізації даних, яка безпечно керує та синхронізує дані користувацьких застосунків на їхніх мобільних пристроях або інтернет-додатках.

Сервіс також надає:

– можливість аутентифікації користувачів через зовнішніх постачальників посвідчень;

– тимчасові дані для доступу до серверних ресурсів, додатків на платформі AWS або до будь-якого сервісу через Amazon API Gateway.

Amazon Cognito працює із зовнішніми постачальниками посвідчень, які підтримують стандарти SAML або OpenID Connect, з соціальними платформами

(такими як Facebook, Twitter, Amazon), а також дозволяє інтегрувати власний постачальник посвідчень [5].

Більш того, Amazon Cognito дозволяє синхронізувати дані на кількох пристроях, щоб користувачі не відчували незручностей при переході між ними або після придбання нового пристрою. Програмне забезпечення може зберігати дані на пристроях користувачів, що дозволить працювати з ним навіть в автономному режимі, а після відновлення підключення до Інтернету проводить автоматичну синхронізацію даних.

Сервіс Cognito надає доступ до власного User Pool, який користувач може налаштувати згідно до своїх вимог. Розглянемо наступі налаштування Cognito User Pool.

1. При створені або налаштуванні можна вказати конкретну політки паролів користувачів, наприклад, вказати рівень надійності паролю або вимоги до типів використовуваних символів та довжини паролю.

2. Використовуючи Cognito Identity, є можливість вказати необхідність перевірки електронних адрес і номерів телефону користувачів, перш ніж надати їм доступ до додатком. Під час реєстрації на мобільний телефон або електронну адресу користувача буде надіслано код для підтвердження, і для завершення реєстрації користувач повинен буде ввести цей код;

3. Дозволити кінцевим користувачам додатка авторизуватися за допомогою багатофакторної аутентифікації на основі SMS. Якщо багатофакторна аутентифікація на основі SMS дозволена, у користувачів буде затребуваний пароль (перший фактор: що їм відомо), і код безпеки, який вони отримають у вигляді SMS на мобільний телефон (другий фактор: чим вони володіють).

4. Вашим користувачам реєструватися або входити за допомогою адреси електронної пошти та пароля або за допомогою номера телефону і пароля за рахунок можливості використання псевдонімів.

В результаті використання пулу користувачів – після вдалої авторизації користувачеві Cognito надає наступні токени, які в майбутньому можуть використовуватися на back-end стороні для аутентифікації користувача:

– Id_token – токен, який містить в собі інформацію про користувача;
– Access_token – токен для ідентифікації користувача;
– Refresh_token – токен для оновлення Id_token та Access_token.

В додаток до User Pool Cognito надає можливість використання пул ідентифікації («Identity Pool»).

Cognito Identity - це повністю керований постачальник посвідчень від сторонніх додатків, що полегшує реалізацію реєстрації і авторизації користувачів мобільних та інтернет-додатків. Дана послуга має вигляд контейнеру для впорядкування федерації посвідчень користувачів додатка. Пул ідентифікації пов'язує федерацію посвідчень від постачальників посвідчень соціальних мереж з унікальними ідентифікаторами користувачів. Пули ідентифікації не зберігають профілів. Пул ідентифікації можна пов'язати з одним або декількома додатками. При використанні двох різних пулів ідентифікації для двох додатків у одного і того ж кінцевого користувача буде окремий унікальний ідентифікатор в кожному з пулів. На рисунку 3.10 зображено приклад роботи Cognito User Pool та Cognito Identity Pool [11].

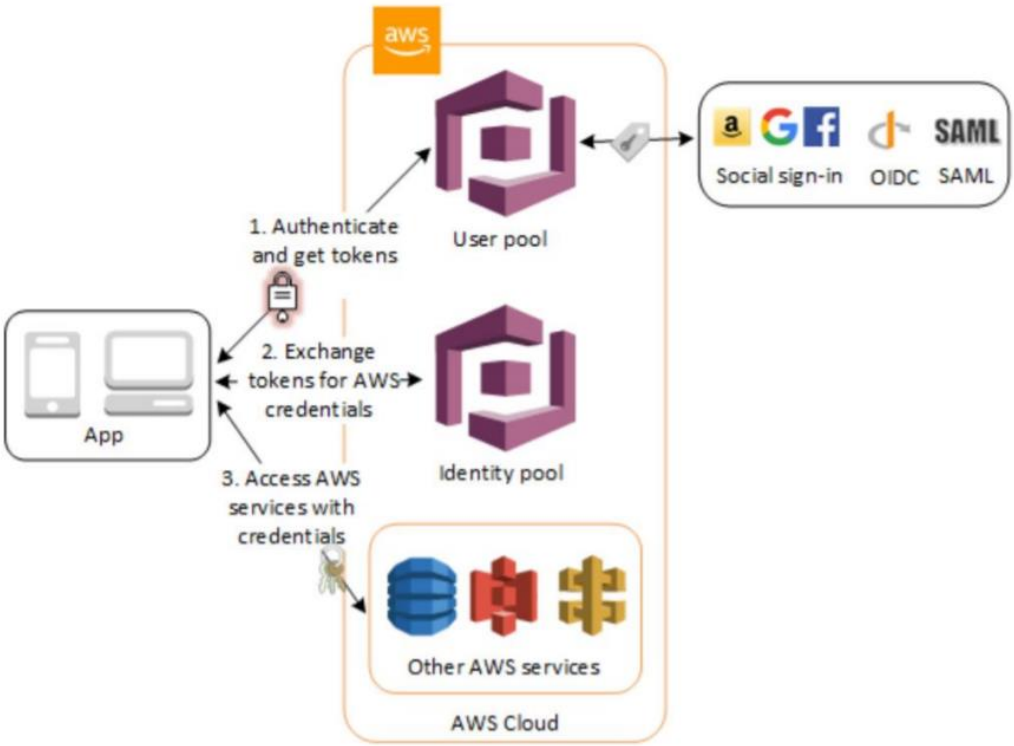


Рисунок 3.10 – Приклад роботи Cognito User Pool та Cognito Identity Pool

3.4.2 JWT

Останнім часом все частіше можна зустріти додатки, які використовують для аутентифікації користувачів механізми JSON Web Tokens. Особливу популярність JWT завоював зі зростанням популярності мікросервісної архітектури: він покладає завдання з обробки автентифікаційних даних на мікросервіси, а отже дозволяє уникнути різних помилок авторизації, збільшити продуктивність і покращити масштабованість програми.

JSON Web Tokens — один із способів представлення даних для передачі між двома або більше сторонами у вигляді об'єкта JSON.

Як правило, структурно JWT складається з трьох частин:

– header – заголовок;
– payload – корисне навантаження;
– signature – підпис.

Заголовок та корисне навантаження – звичайні JSON-об'єкти, які необхідно додатково закодувати за допомогою алгоритму base64url. Закодовані частини з'єднуються один з одним, і на їх основі обчислюється підпис, який також стає частиною токена. Загальний вигляд JWT-токена зображено на рисунку 3.11.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjEzMzciLCJ1c2Vyb
mFtZSI6ImJpem9uZSIsImlhdCI6MTU5NDIwOTYwMCwicm9sZSI6InVy

Рисунок 3.11 – Загальний вигляд JWT-токена

Заголовок - службова частина токена. Він допомагає застосуванню визначити, як слід обробляти отриманий токен. Ця частина, як було згадано раніше, є JSON-об'єктом (рис 3.12).

{
"type": "JWT",
"alg": "HS256"
}

Рисунок 3.12 – Заголовок у вигляді JSON-об'єкта

Заголовок складається з таких полів:

– typ - тип токена, наприклад JWT;

– alg - алгоритм, використаний для генерації підпису.

Значення поля typ часто ігнорується програмами, проте стандарт не рекомендує відмовлятися від нього для забезпечення зворотної сумісності.

Поле alg є обов'язковим для заповнення. У наведеному випадку було застосовано алгоритм HS256 (HMAC-SHA256), в якому для генерації та перевірки підпису використовується єдиний секретний ключ. Для підпису JWT можуть застосовуватись і алгоритми асиметричного шифрування, наприклад RS256 (RSA-SHA256). Стандарт допускає використання інших алгоритмів, включаючи HS512,

RS512, ES256, ES512, none та ін. Використання алгоритму none вказує на те, що токен не було підписано. У подібному токені відсутня частина з підписом, і встановити її справжність неможливо [21].

У корисному навантаженні передається будь-яка інформація, яка допомагає застосуванню так чи інакше ідентифікувати користувача. Додатково можуть передаватися певні службові поля, проте вони не є обов'язковими для заповнення, тому на них зупинятися ми не будемо.

На рисунку 3.12 зображено корисне навантаження.


```
{
"id": "1337",
"username": " frank_dungeon_master@gmail.com",
"iat": 1594209600,
"role": "user"
}
```


Рисунок 3.13 – Приклад корисного навантаження

В прикладі корисного навантаження приведені наступні поля:

– id – унікальний ідентифікатор користувача;
– username - ім'я користувача;
– iat – службове поле, час генерації токена у форматі Unix time;
– role – роль користувача, наприклад admin, user, guest.

Оскільки набір полів у частині корисного навантаження довільний, програма може зберігати в цій частині практично будь-які дані. Наприклад, для прискорення роботи програми в корисному навантаженні можуть зберігатися П. І. Б. користувача, щоб не запитувати ці відомості щоразу з бази даних.

Підпис генерується в такий спосіб. Заголовок та корисне навантаження кодуються за допомогою алгоритму base64url, після чого об'єднуються в єдиний рядок з використанням крапки (".") як роздільник. Генерується підпис, який додається до вихідного рядка так само через крапку (рисунок 3.14).

```
signature = HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  SECRET_KEY
)

JWT = base64UrlEncode(header) + "." + base64UrlEncode(payload) + "." +
base64UrlEncode(signature)
```

Рисунок 3.14 – Псевдокод генерації JWT-токена

Отримавши JWT від користувача, програма самостійно обчислить значення підпису та порівняє його з тим значенням, яке було передано в токені. Якщо ці значення не співпадають, значить, токен був модифікований або згенерований недовіреною стороною, і приймати такий токен і довіряти додаток йому не буде.

Розглянемо переваги використання JWT у порівнянні з класичною схемою автентифікації, яка використовує сесію.

– Підхід з використанням токенів дозволяє не зберігати інформацію про всі видані токені, як при класичній схемі. Коли користувач звертається до програми, він передає йому свій токен. Додатоку залишається лише перевірити підпис і витягти необхідні поля з корисного навантаження.

– Додаткам взагалі не обов'язково займатися видачею та валідацією токенів самостійно, найчастіше для цих цілей використовується окремий сервіс аутентифікації.

– При використанні окремого сервісу аутентифікації стає можливим організувати єдину точку входу в різні сервіси з тими самими обліковими даними. Одного разу пройшовши процедуру аутентифікації, користувач зможе отримати доступ зі своїм токеном до ресурсів, які довіряють цьому сервісу аутентифікації.

– Програма може зберігати в частині корисного навантаження практично будь-які дані, що при грамотній архітектурі програми може суттєво збільшити продуктивність.

Завдяки перерахованим факторам схема аутентифікації з використанням JWT широко використовується у різних корпоративних додатках. Особливо популярна ця схема у додатках, які реалізують парадигми мікросервісної архітектури: за допомогою такого підходу кожен сервіс отримує необхідну йому інформацію про користувача безпосередньо з токена, а не витрачає час отримання цієї інформації з бази даних.

3.5 AWS Load Balancer

Load Balancer - це балансувальник навантаження, який приймає вхідний мережевий трафік від клієнта та, ґрунтуючись на певних умовах цього трафіку, відправляє ці повідомлення на один з декількох бекенд-серверів. При використанні Load Balancing з Auto Scaling можна створювати високодоступні відмовостійкі додатки, які будуть автоматично масштабувати ресурси як в сторону збільшення, так і в бік зменшення в залежності від коливань попиту.

Місце балансувальника - проміжний етап між DNS (Route 53) і серверами (server 1, server 2, server 3). На балансувальник приходить запит (client request, IP: 80), він перенаправляє цей запит на сервер (request from LB), при цьому сервер може використовувати інший порт (IP2: 8080), далі балансувальник приймає відповідь від сервера і передає його клієнтові [15].

Високодоступність та високонагруженість додатку реалізується за рахунок:

– сервер з додатком може вийти з ладу, але до тих пір, поки залишається хоча б один сервер в робочому стані, балансувальник навантаження буде направляти трафік на цю машину;

– два сервера, що працюють за балансувальником навантаження, дозволять обробляти вдвічі більший трафік від клієнтів. Іншими словами, у міру збільшення мережевого трафіку, балансувальник навантаження дозволить легко додавати все більше і більше бекенд-серверів.

Різниця Load Balancer та DNS полягає в:

– клієнт безпосередньо надсилає запит серверу (тобто, якщо раптом сервер зламається, то клієнт буде мати справу зі зламаним сервером – переадресації на інший, працюючий сервер не відбудеться);

– LB завжди взаємодіє з клієнтом - проміжна ланка між клієнтом і сервером (тобто, якщо один сервер не робочий, то LB переадресує його на робочий);

– якщо потрібна можливість попадання конкретного клієнта на конкретний сервер, потрібно включити можливість прив'язування конкретного IP до конкретного сервера (stickness IP).

LB працює з Amazon VPC, надаючи різні можливості щодо забезпечення безпеки – це інтегроване управління сертифікатами, аутентифікація користувачів, розшифровка SSL / TLS. Все разом забезпечує централізоване і гнучке управління налаштуваннями TLS. Будь-який балансувальник навантаження можна налаштувати для роботи з виходом в Інтернет або створити балансувальник навантаження без публічної IP-адреси для роботи в якості внутрішнього балансувальника навантаження (без виходу в Інтернет) [13].

На рисунку 3.15 зображено створення балансувальника навантаження за допомогою Terraform коду.

```
resource "aws_alb" "load_balancer" {
  name = "${var.project_name}-alb"
  security_groups = [
    aws_security_group.load_balances.id
  ]
  subnets = data.terraform_remote_state.vpc.outputs.subnet_public_id

  depends_on = [
    aws_security_group.load_balances
  ]
```

Рисунок 3.15 – Створення балансувальника навантаження

Проте налаштування захищеного підключення до веб-додатку, з використанням протоколів SSL/TLS, за допомогою програмного коду потребує

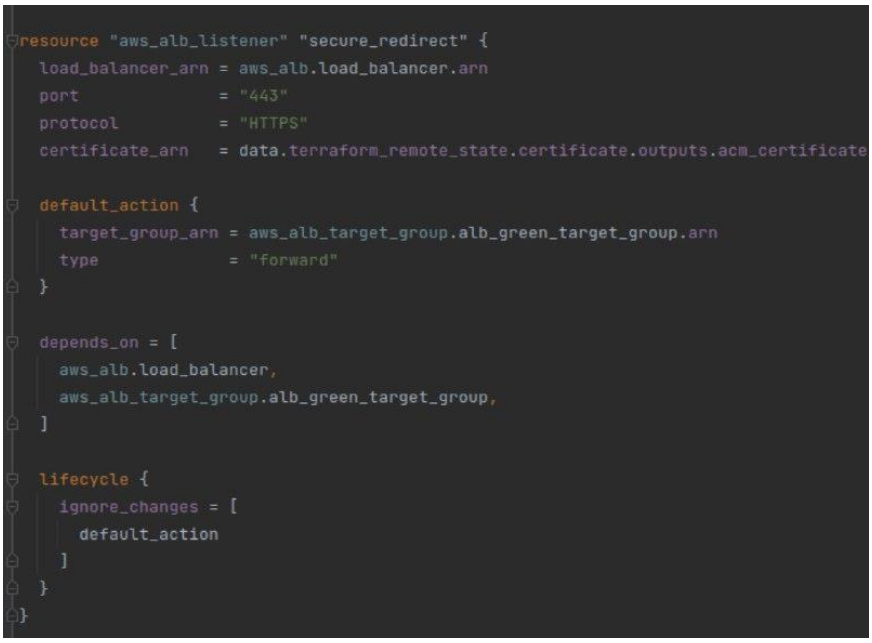
впевнених знать протоколу HTTP/HTPPS та практичних навичок з можливостей перенаправлення та відхилення різних типів запитів. В даній роботі було створено сертифікати, які дозволяють використовувати безпечне підключення, а також розроблено такий балансувальник навантаження, який забороняє різні види підключення, окрім HTTPS (443 порт).

В результаті проектування створено два лісенери (aws_alb_listener), які відповідають за прийняття виключно того трафіку, що надходить на 443 порт. На рисунку 3.16 зображено лісенер, який зчитує будь-який трафік за порту 80 (HTPP) та виконує перенаправлення на порт 443 (HTPPS).

```
resource "aws_alb_listener" "http_listener" {
  load_balancer_arn = aws_alb.load_balancer.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type = "redirect"

    redirect {
      port        = "443"
      protocol    = "HTTPS"
      status_code = "HTTP_301"
    }
  }

  depends_on = [
    aws_alb.load_balancer
  ]
}
```

Рисунок 3.16 – Лістенер, що виконує перенаправлення з 80 порту на 443

Також було створено інший лісенер, що виконує перевірку захищеності трафіку і в разі успішності перевірки перенаправляє трафік на цільову групу (рис. 3.17).



```
resource "aws_alb_listener" "secure_redirect" {
  load_balancer_arn = aws_alb.load_balancer.arn
  port              = "443"
  protocol          = "HTTPS"
  certificate_arn   = data.terraform_remote_state.certificate.outputs.acm_certificate

  default_action {
    target_group_arn = aws_alb_target_group.alb_green_target_group.arn
    type             = "forward"
  }

  depends_on = [
    aws_alb.load_balancer,
    aws_alb_target_group.alb_green_target_group,
  ]

  lifecycle {
    ignore_changes = [
      default_action
    ]
  }
}
```

Рисунок 3.17 – Лісенер для HTTPS протоколу

3.6 Сертифікати

AWS Certificate Manager - це сервіс, що дозволяє легко надавати і розгортати публічні та приватні сертифікати Secure Sockets Layer / Transport Layer Security (SSL/TLS) для використання разом з сервісами AWS або внутрішніми підключеними ресурсами, а також допомагає керувати цими сертифікатами.

Сертифікати SSL/TLS використовуються для захисту мережевих підключень і встановлення автентичності веб-сайтів в Інтернеті, а також ресурсів в приватних мережах. AWS Certificate Manager дозволяє не витрачати час на придбання, завантаження і оновлення сертифікатів вручну. Завдяки даному сервісу можна швидко запросити сертифікат, виконати його розгортання за допомогою таких ресурсів AWS, як балансувальник навантаження сервісу Elastic Load Balancing, бази роздачі Amazon CloudFront або API в Amazon API Gateway, а також дозволити сервісу AWS Certificate Manager виконувати оновлення сертифікатів [7].

Даний сервіс дозволяє також створювати приватні сертифікати для внутрішніх ресурсів і централізовано керувати їх життєвим циклом. Публічні та приватні сертифікати SSL/TLS, які надаються за допомогою AWS Certificate

Manager і використовуються тільки для інтегрованих з ACM сервісів (наприклад, Elastic Load Balancing, Amazon CloudFront, Amazon API Gateway), є безкоштовними.

Однією з передових методик перевірки сертифікатів є – перевірка за допомогою запису DNS. Така перевірка дозволяє легко підтвердити, що ви володієте або керуєте доменом, для отримання сертифіката SSL/TLS. При перевірці за допомогою запису DNS, щоб підтвердити можливість управління своїм доменним ім'ям, досить внести в конфігурацію DNS запис CNAME. Для спрощення процесу перевірки за допомогою запису DNS консоль управління ACM може автоматично налаштувати записи DNS, якщо для управління ними використовується сервіс Amazon Route 53.

3.7 Створення та контроль секретних змінних

Сервіс AWS Key Management Service (KMS) забезпечує централізований контроль над ключами шифрування, що використовуються для захисту даних. AWS KMS інтегрований з сервісами AWS, що спрощує шифрування даних, що зберігаються в цих сервісах, і управління доступом до використовуваних ключів шифрування. Завдяки інтеграції AWS KMS з AWS CloudTrail можна перевіряти, хто використовував ключі, які саме, коли і для яких ресурсів. AWS KMS також дозволяє розробникам без зайвих зусиль додавати функціонал шифрування в код додатків безпосередньо через API сервісу для шифрування і дешифрування або шляхом інтеграції з пакетом AWS Encryption SDK [9].

AWS KMS влаштований таким чином, що ніхто, у тому числі працівникам AWS, не може витягти з сервісу незашифровані ключі. Сервіс використовує перевірені на відповідність FIPS 140-2 апаратні модулі безпеки (HSM) для захисту конфіденційності та цілісності всіх ключів - як створених KMS від імені користувача та створених в кластері AWS CloudHSM, так і імпортованих в сервіс. Незашифровані ключі ніколи не записуються на диск і використовуються тільки в енергозалежній пам'яті HSM протягом часу, необхідного для виконання

запитаної криптографічного операції. Ключі, створені KMS, можуть використовуватися тільки в тому регіоні AWS, де вони були створені, і ніколи не передаються за його межі.

Ця послуга відіграє важливу роль у захисті даних, що зберігаються програмним забезпеченням.

4 РЕАЛІЗАЦІЯ БЕЗПЕЧНОГО ВЕБ-ДОДАТКУ ЗА ДОПОМОГОЮ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ TERRAFORM ТА DJANGO

4.1 Terraform код

В ході виконання роботи було досліджено та використано наступні сервіси клаудпровайдера AWS для створення та налаштування безпечного веб-додатку:

- AWS Cognito;
- AWS Database Instance (With Postgres engine);
- AWS EC2;
- AWS ECS;
- AWS KMS;
- AWS VPC;
- AWS Load Balancing;
- AWS CloudWatch;
- AWS CodeBuild;
- AWS CodePipelines;
- AWS CodeDeploy;
- AWS SNS;
- AWS Secret Manager;
- AWS ECR;
- AWS IAM.

Бекенд сторона була розроблена на мові програмування Python з використанням веб-фреймворку Django та Django Rest framework. За допомогою Python коду було створено власну систему автентифікації користувачів на основі JWT ID-токенів від AWS Cognito. Створений пул використовує сервіс AWS SNS для реалізації двухфакторної автентифікації користувачів за допомогою відправки повідомлень на мобільний телефон.

На рисунку 4.1 зображено створення пулу користувачів, який відповідає за реєстрацію, авторизації та зберігання даних про користувачів.

```
resource "aws_cognito_user_pool" "user_pool" {
  name = "${var.developer}-users"

  username_attributes      = ["phone_number"]
  auto_verified_attributes = ["phone_number"]

  password_policy {
    minimum_length                   = 8
    require_symbols                  = true
    require_numbers                  = true
    require_uppercase                = true
    require_lowercase                = true
    temporary_password_validity_days = 1
  }

  username_configuration {
    case_sensitive = false
  }

  account_recovery_setting {
    recovery_mechanism {
      name     = "verified_phone_number"
      priority = 1
    }
  }

  admin_create_user_config {
    allow_admin_create_user_only = false
  }

  device_configuration {
    challenge_required_on_new_device      = true
    device_only_remembered_on_user_prompt = false
  }

  sms_configuration {
    external_id    = "diploma_external_id"
    sns_caller_arn = aws_iam_role.sns_caller.arn
```

Рисунок 4.1 – Код створення Cognito User Pool

На даному рисунку зображено код, який визначає вимоги для користувацьких паролів, а саме:

– найменша допустима довжина паролю – 8 символів;

– обов’язкове використання символів;

– обов’язкове використання чисел;

– обов’язкове використання символів верхнього регістру;
– обов’язкове використання символів нижнього регістру;
– код для підтвердження аккаунту валідний лише 1 день.

Також при налаштуванні було прийнято рішення, що в ситуації, коли користувач використовує сервіс з нового пристрою (телефон, комп’ютер) – запитувати тимчасовий пароль, для проведення двохфакторної автентифікації. При цьому найвищий пріоритет для відновлення доступу до існуючого акаунту був вибраний спосіб за допомогою номеру телефону.

Крім цього було створено клієнт, який надає можливість налаштовувати властивості пулу для розробників. На рисунку 4.2 зображено код для налаштування:

```
resource "aws_cognito_user_pool_client" "user_pool_client" {
  generate_secret               = false
  name                          = "${var.developer}-user-pool-client"
  user_pool_id                  = aws_cognito_user_pool.user_pool.id
  access_token_validity         = var.tokens_validity
  refresh_token_validity        = 365
  id_token_validity             = var.tokens_validity
  prevent_user_existence_errors = "LEGACY"
  enable_token_revocation       = true
  callback_urls                 = [var.default_url]
  default_redirect_uri          = var.default_url
  explicit_auth_flows           = [
    "ALLOW_ADMIN_USER_PASSWORD_AUTH",
    "ALLOW_CUSTOM_AUTH",
    "ALLOW_REFRESH_TOKEN_AUTH",
    "ALLOW_USER_PASSWORD_AUTH",
    "ALLOW_USER_SRP_AUTH"
  ]

  token_validity_units {
    access_token  = "minutes"
    id_token      = "minutes"
    refresh_token = "minutes"
  }
}
```

Рисунок 4.2 – Створення клієнту Cognito пулу

В даній роботі було обрано, що користувацький токен авторизації, токен авторизації дійсні виключно 30 хвилин, а токен для відновлення ACCESS та ID

токенів дійсний 365 днів. Після того, як токени стануть не дійсними – потрібно виконувати дії для отримання нових. Також були визначені всі можливі алгоритми автентифікації токенів.

В ході роботи також було створено Database Instance з використанням системи керуванням база даних – PostgreSQL. На рисунку 4.3 зображено код для створення бази даних.

```
resource "aws_db_instance" "postgresql_db_instance" {
  identifier                = "${var.developer}-db-instance"
  final_snapshot_identifier = "${var.developer}-db-instance-snapshot"
  name                      = var.database_name
  allocated_storage         = var.allocated_storage
  max_allocated_storage     = var.max_allocated_storage
  storage_type              = var.storage_type
  engine                    = var.database_engine
  engine_version            = var.database_engine_version
  instance_class            = var.database_type
  username                  = var.database_user
  password                  = var.database_password
  port                      = var.database_port

  db_subnet_group_name = aws_db_subnet_group.database_subnet_group.name

  vpc_security_group_ids = [
    aws_security_group.database_sg.id
  ]

  depends_on = [
    aws_security_group.database_sg,
    aws_db_subnet_group.database_subnet_group,
  ]
}
```

Рисунок 4.3 - код створення бази даних в хмарному провайдері AWS

Даний код визначає такі важливі моменти, як:

– пароль для доступу до БД;
– ім’я бази даних;
– ім’я користувача, який має доступ до БД;
– тип СКБД та її версію;
– максимальний розмір БД.

Також важливими моментами при створені бази даних являються:

– додавання її до створеної VPC, а саме до приватної підгрупи;

– надання доступу до неї винятково з машин EC2, на яких буде розгорнуто наш додаток.

Пункт номер два являється критичним, адже він надає можливість підключатися до БД виключно з серверів, які підконтрольні нам. У випадку, якщо дані, що зберігаються в базі даних будуть змінені або скомпрометовані, то це буде свідчити про те, що наші сервери, EC2 машини, також підверглись несанкціонованому доступу.

Наступним важливим пунктом створення безпечного веб-додатка – це налаштування AWS VPC та NAT для забезпечення мережевої безпеки. На рисунках 4.4 та 4.5 зображено створення та конфігурацію приватної мережі в хмарі. Було реалізовано 2 підгрупи: публічну та приватну (рисунок 4.6), а також NAT і призначено йому публічну IP адресу. Окрім цього було також створено інтернет шлюз, та шлюз для технології NAT.

```
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = "true"
  tags                 = {
    Name = "Default vpc. Developer - ${var.developer}."
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.developer}-igw"
  }
}

resource "aws_eip" "nat_eip" {
  vpc        = true
  depends_on = [aws_internet_gateway.igw]
}

resource "aws_nat_gateway" "nat" {
  count         = length(aws_subnet.public_subnet.*.id)
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = element(aws_subnet.public_subnet.*.id, count.index)
  depends_on    = [aws_internet_gateway.igw]

  tags = {
    Name = "nat"
  }
}
```

Рисунок 4.4 – Створення AWS VPC та NAT для забезпечення мережевої безпеки

```
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

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

resource "aws_main_route_table_association" "public" {
  vpc_id         = aws_vpc.main.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat.*.id[count.index]
  }

  count = length(aws_subnet.private_subnet)
}

resource "aws_route_table_association" "private" {
  subnet_id      = aws_subnet.private_subnet.*.id[count.index]
  route_table_id = aws_route_table.private.*.id[count.index]
  count          = length(aws_subnet.private_subnet)
}
```

Рисунок 4.5 – Налаштування технології NAT

```
resource "aws_db_subnet_group" "database_subnet_group" {
  name       = "${var.developer}-default_database_subnet_group"
  subnet_ids = aws_subnet.private_subnet.*.id
}

resource "aws_subnet" "public_subnet" {
  count                   = length(var.availability_zones)
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.public_cidr_block
  map_public_ip_on_launch = true
  availability_zone       = "${var.aws_region}${var.availability_zones[count.index]}"

  tags = {
    Name = "Default public subnet. Developer - ${var.developer}."
  }
}

resource "aws_subnet" "private_subnet" {
  count                   = length(var.availability_zones)
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.private_cidr_block
  map_public_ip_on_launch = false
  availability_zone       = "${var.aws_region}${var.availability_zones[count.index]}"

  tags = {
    Name = "Default private subnet. Developer - ${var.developer}."
  }
}
```

Рисунок 4.6 – Створення підгруп для технології NAT та бази даних.

Кожен додаток використовує в своїй роботі секретні дані, які повинні надійно зберігатися та коректно оброблятися та передаватися кожного разу, коли виконується запуск програмного забезпечення на сервері. Для цього було обрано AWS Secrets Manager – для зберігання даних та сервіси для налаштування Secure Continuous Integration, Continuous delivery – AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline. На рисунках 4.6, 4.7 та 4.8 зображено код для налаштування вище перерахованих сервісів.

Рисунок 4.7 – Створення ресурсів в AWS SM для зберігання секретних даних

```
resource "aws_codedeploy_deployment_group" "deployment_group" {
  app_name              = aws_codedeploy_app.deploy.name
  deployment_group_name = "${var.developer}-deployment-group"
  service_role_arn      = aws_iam_role.codedeploy.arn
  deployment_config_name = aws_codedeploy_deployment_config.deployment_config.id

  auto_rollback_configuration {
    enabled = true
    events  = ["DEPLOYMENT_FAILURE"]
  }

  blue_green_deployment_config {
    deployment_ready_option {
      action_on_timeout = "CONTINUE_DEPLOYMENT"
    }

    terminate_blue_instances_on_deployment_success {
      action                           = "TERMINATE"
      termination_wait_time_in_minutes = 2
    }
  }

  deployment_style {
    deployment_option = "WITH_TRAFFIC_CONTROL"
    deployment_type   = "BLUE_GREEN"
  }

  ecs_service {
    cluster_name = data.terraform_remote_state.ecs_cluster.outputs.ecs_cluster_name
    service_name = aws_ecs_service.ecs_service.name
  }

  load_balancer_info {
    target_group_pair_info {
      prod_traffic_route {
        listener_arns = [aws_lb_listener.http_listener.arn]
      }
```

Рисунок 4.8 – Налаштування Continuous deployment

В ході роботи було використано сервіс AWS CodeDeploy, який дозволяє реалізувати методологію Continuous deployment (рисунок 4.8 та рис. 4.9). Найголовнішим в налаштуванні – є використання стилю deploymenta. В ході роботи було використано blue-green deployment style, що дає можливість уникнути великого часу простою, коли команда інженерів буде оновлювати веб-додаток.

Такий вид розгортування додатків спочатку запускає новий сервер на машині EC2, де виконується повний деплой додатку, що розробляється, далі за

допомогою балансувальника навантаження або певних пунктів налаштування сервісу ECS та Autoscaling Group перевіряється чи відповідає розгорнутий додаток на новому сервері на запити. Якщо все пройшло без помилок і новий додаток відповідає на перевірки, то балансувальник навантаження перенаправляє всі нові запити на нього. Після того, як всі сесії та запити, що оброблялися старим сервером завершилися, балансувальник більше не відправляє на нього запити і з деяким часом сервіс ECS знищує даний сервер.

За допомогою даної технології реалізується методологія Secure Software Development Lifecycle, яка забезпечує доступність додатку в будь який момент часу.

```
stage {
  name = "Deploy"

  action {
    name            = "Deploy"
    category        = "Deploy"
    owner           = "AWS"
    provider        = "CodeDeployToECS"
    version         = "1"

    input_artifacts = [
      "image_output",
      "source_output",
    ]

    configuration   = {
      AppSpecTemplateArtifact : "source_output",
      AppSpecTemplatePath : "appspec-dev.yaml",
      TaskDefinitionTemplateArtifact : "source_output",
      TaskDefinitionTemplatePath : "taskdef-dev.json",
      ApplicationName : aws_codedeploy_app.deploy.name,
      DeploymentGroupName : aws_codedeploy_deployment_group.deployment_group.deployment_group_name
      Image1ArtifactName : "image_output",
      Image1ContainerName : "IMAGE"
    }
  }
}
```

Рисунок 4.9 – Визначення всіх потрібних ресурсів, які потрібні для коректного розгортання додатка

Для того, що уникнути можливості компрометації секретних даних в роботі було прийнято рішення використовувати Task Definition (рис. 4.10). Це надає нам можливість замість секретних даних передавати їх AWS ARN, які ми отримали після створення ресурсів в сервісі AWS Secrets Manager. Проте для коректної роботи потрібно надати права Task Definition на отримання секретних даних від менеджера секретів (рисунок 4.11).

AWS дотримується правил найменших привілеїв як для користувачів, так і для ролей та ресурсів. Цей функціонал надає нульові права будь-чому, то ж при створенні всіх ресурсів, що використовувалися для реалізації безпечного веб–додатка були також створені ролі та надані їм коректні права. Створення ролей та видача привілеїв користувачам або ж сервісам виконується за допомогою сервісу корування доступом AWS IAM. Для кожного із ресурсів повинна бути згенерована довірча роль, а також указані ARN-и тих користувачів або сервісів, що мають право на співпрацю. Повний лістинг програми, включно з кодом створення ролей та надання прав доступу, буде приведено в додатку А.

Рисунок 4.10 – Файл в проекті, що визначає всі змінні для Task Definition

```
name = "${var.developer}-ecs-task"

assume_role_policy = jsonencode({
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : "sts:AssumeRole",
      "Principal" : {
        "Service" : "ecs-tasks.amazonaws.com"
      },
      "Effect" : "Allow"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "ecr:*",
        "logs:*"
      ],
      "Resource" : "*"
    },
    {
      "Effect" : "Allow",
      "Action" : [
        "secretsmanager:GetSecretValue"
      ],
      "Resource" : [
        data.aws_secretsmanager_secret.database_name.arn,
        data.aws_secretsmanager_secret.database_host.arn,
        data.aws_secretsmanager_secret.database_port.arn,
        data.aws_secretsmanager_secret.database_password.arn,
        data.aws_secretsmanager_secret.database_username.arn,

        data.aws_secretsmanager_secret.cognito_user_pool.arn,
        data.aws_secretsmanager_secret.cognito_client_ids.arn,

        data.aws_secretsmanager_secret.app_secret_key.arn

      ]
```

Рисунок 4.11 – Надання прав доступу до ресурсів AWS SM для Task Definition

Після того як було створено всі необхідні ресурси в клаудпровайдері AWS, а також настроєно тригери та веб-хуки з репозиторієм, де зберігається робочий код веб-додатку – запускається процес розгортання додатку на машинах EC2 за допомогою AWS CodePipelines та AWS ECS.

Як тільки новий Docker-контейнер відповідає вимогам робото здатності, які ми вказували при створені сервісу AWS ECS, то виконується перенаправлення HTTP трафіку, тобто відбувається розгортання додатку за допомогою системи blue-green. За рахунок цього наш додаток відповідає запланованій бізнес моделі, тобто не буде часу простою, при якому наш сервіс являється недоступним. Це являється критично важливою складовою, якщо наш веб-сервер обробляє конфіденційні дані, наприклад, сервер для онлайн банкінгу.

Для перевірки коректності всіх сконфігурованих сервісів, що використовуються для створення безпечних додатків було також написано власний сервер, який автенифікує винятково користувачів, які являються учасниками створеного власноруч AWS Cognito user pool (рис. 4.12).

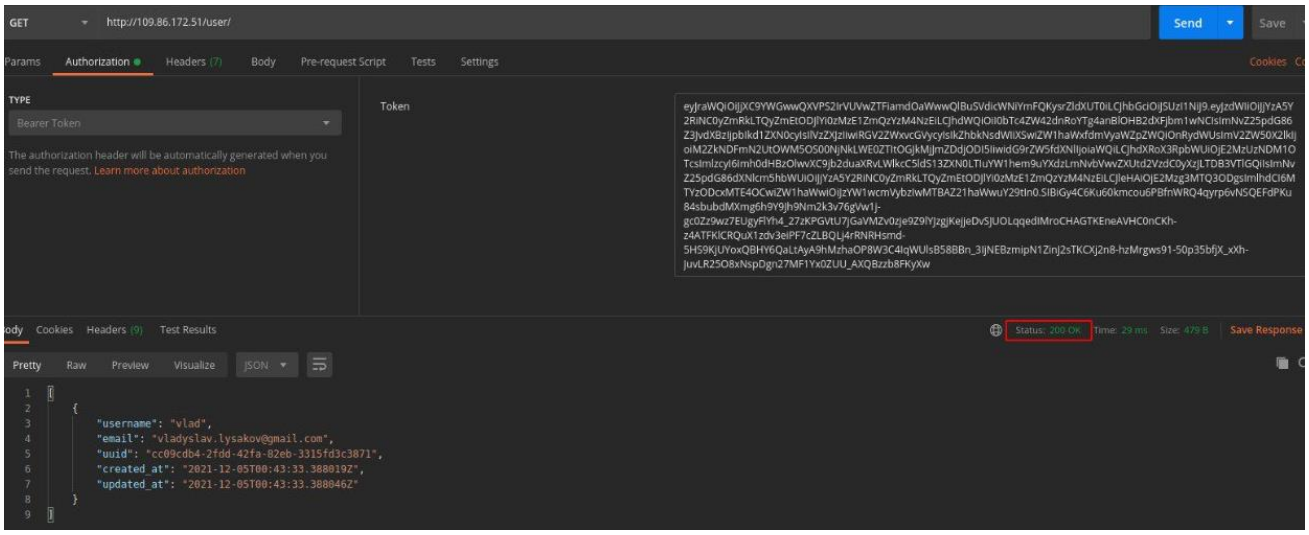


Рисунок 4.12 – Автентифікація користувача на власноруч розгорненому веб-сервері за допомогою AWS Cognito

В даній роботі також було проведено тестування коректності авторизаційної системи власного веб-серверу. Програмне забезпечення було написано за допомогою мови програмування Python та веб-фреймворків Django/Django rest-framework. Авторизація безпосередньо виконувалась сторонньою бібліотекою djangorestframework_simplejwt, яка, на момент написання кваліфікаційної роботи, являється однією з найрозповсюдженіших. Для використання в своїй програмі

необхідно було виконати імпорт програмного модуля, а також його реєстрацію. Так як ми використовуємо для авторизації користувачів сервіс клаудпровайдера, то налаштування програмного забезпечення djangorestframework_simplejwt були перевизначені, для коректної роботи (рис. 4.13).

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': ('rest_framework_simplejwt.authentication.JWTAuthentication',),
    'DEFAULT_PERMISSION_CLASSES': 'pet_project.security.permissions.DenyAny',
}

SIMPLE_JWT = {
    'UPDATE_LAST_LOGIN': True,
    'VERIFYING_KEY': CognitoConfig.keys,
    'ALGORITHM': CognitoConfig.algorithm,
    'AUDIENCE': CognitoConfig.app_client_ids.split(),
    'JWK_URL': CognitoConfig.keys_url,
    'ISSUER': CognitoConfig.issuer,
    'JTI_CLAIM': 'event_id',
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=30),
    'TOKEN_TYPE_CLAIM': 'token_use',
    'USER_ID_CLAIM': 'sub',
    'USER_ID_FIELD': 'uuid',
    'AUTH_TOKEN_CLASSES': ('pet_project.security.jwt.CognitoToken',),
}

AUTH_USER_MODEL = DatabaseConfig.user_model
```

Рисунок 4.13 – Перевизначення параметрів для авторизаційної системи

Для підтримки авторизації за допомогою токенів доступу, що були згенеровані та управляються AWS Cognito, були превизначенні такі параметри системи розмежування доступу:

– Verifying key – ключ або набір ключів, що отримуються за запитом від AWS Cognito. Для отримання ключів потрібно згенерувати URL-адресу, яка посилається на створений пул користувачів в хмарі.

– Algorithm – алгоритм за допомогою якого шифруються та розшифровуються веб-токени. Вході роботи використовувався «RS256».

– Audience – це значення, які були згенеровані AWS Cognito для перевірки токенів на стороні веб-серверу. Дане значення було передано в додаток за допомогою Environment Variables в Task Definition, проте не як значення, а як ARN ресурсу, що уникнення компроментації.

– JWK URL – URL-адреса, яка посилається на створений пул користувачів в хмарі. Приклад – «https://cognito-idp.eu-west-2.amazonaws.com/{user_pool_id}/.well-known/jwks.json».

– Issuer – це ідентифікатор ресурсу, який видав даний токен. Використовується для перевірки довіреного емітенту.

– User ID claim – значення, що зберігається в середині корисного навантаження JWT токену. Amazon Web Services рекомендує використовувати для авторизації користувачів ID токен, тому для даного параметру налаштування було використано значення – «sub».

– User ID field – назва поля в базі даних, яка зберігає унікальний ідентифікатор користувача, який має співпадати з тим, що був переданий в JWT токені. В ході проектування програмного забезпечення було прийнято рішення, що для запобігання можливих проблем, а також колізій, при створені унікальних значень для користувачів, цю задачу перенести на сторону хмарного провайдера, коли веб-сервер буде тільки зберігати це значення в базі даних та використовувати його для авторизації.

– Auth token classes – написаний власноруч клас, що виконує перевірку та валідацію веб-токену.

Тестування авторизаційної системи, що використовувала AWS Cognito для перевірки користувачів показала високу швидкість та надійну верифікацію користувацьких токенів. Окрім цього, було також дотримано рекомендацій загальновідомих архітектурних стилів взаємодії компонентів програмного забезпечення (рис. 4.14 та рис 4.15).

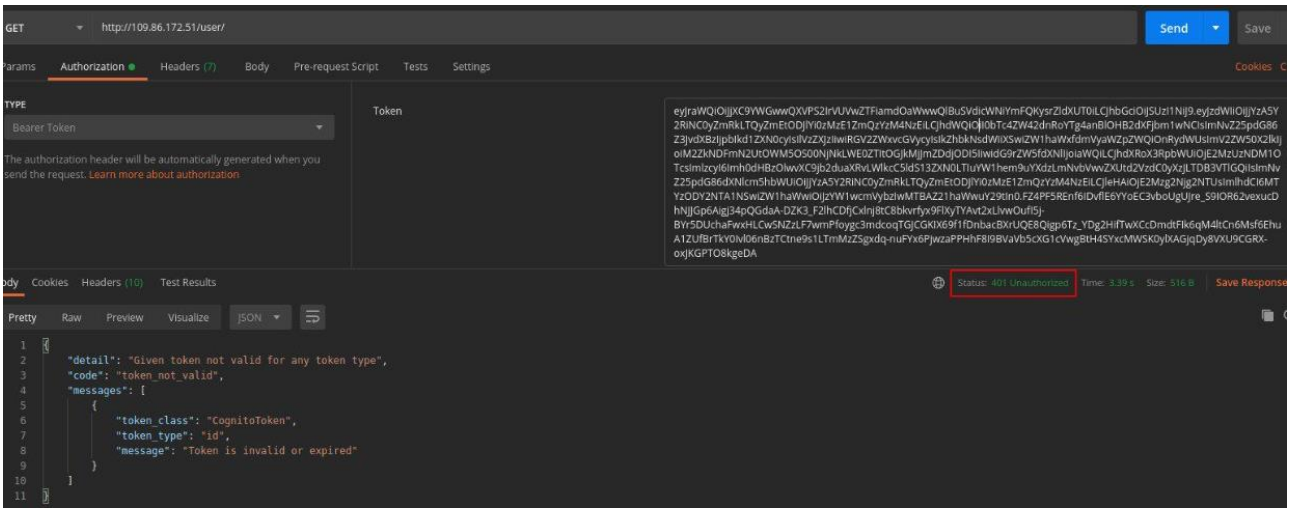


Рисунок 4.14 – Помилка авторизації, як відповідь на запит з не валідним токеном доступу

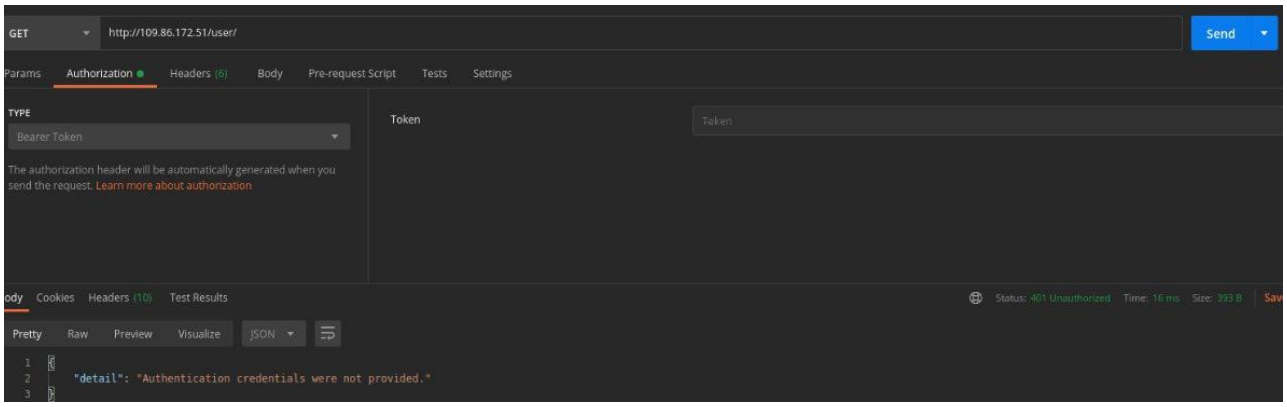


Рисунок 4.15 – Помилка авторизації, як відповідь на запит без авторизаційного токену.

На приведених вище скріншотах можна побачити, що сервер має інтерфейс, що відповідає світовим стандартам та рекомендаціям в створенні веб-додатків.

ВИСНОВКИ

При детальному розгляді клаудпровайдера AWS були досліджені та проаналізовані сервіси, які надають гарантії безпеки при розробці власного продукту.

Проте для використання потрібне глибоке його розуміння для подальшої можливої взаємодії та конфігурації. Так як хмарний провайдер надає винятково окремі сервіси, які потенційно можна компонувати, вся практична реалізація залежить виключно на уяві та технічному розумінні архітектора проекту. Саме тому, на даний момент перспективні IT-компанії дуже цінують розробників, архітекторів та secure DevOps-інженерів з твердими знаннями та навичками використання сервісів клаудпровайдера AWS.

В першому розділі даної роботи були розглянуті загальні характеристики безпеки інформації в хмарних обчисленнях. Детально були описані моделі порушника та загроз, виявлені потенційні вразливості. Були розроблені вимоги до безпеки, а також методи захисту від загроз при використанні хмарного обчислення.

В другому розділі мова йдеться про клаудпровайдер AWS та його можливості. Окремо в розділах розглядаються загальні характеристики безпеки хмарного провайдера та його обов’язки щодо безпеки даних, які він оброблює.

Третій розділ присвячений розгляду сервісів, що реалізують різні послуги безпеки для веб додатку, розгорненого в хмарі клаудпровайдеру AWS. Були детально досліджені такі сервіси, як Cognito, VPC, Load Balancer, IAM та інші, які в були використані в практичній реалізації кваліфікаційної роботи за допомогою програмного інструменту оркестрації хмарних технологій Terraform&Terragrunt.

В четвертому розділі було детально розглянуто схеми створення кожного окремого ресурсу за допомогою коду на мові Terraform&Terragrunt.

В ході виконання роботи було досліджено декілька сервісів клаудпровайдера AWS, що надають можливості створення безпечного веб-

додатку, а також дозволяють спроектувати та побудувати безпечний життєвий цикл програмного забезпечення. В проектування програмного забезпечення було придумано реалізовано схему взаємодії всіх можливих сервісів, як одне ціле, що в результаті надають високо стійкий та високо навантажений веб-додаток, з послугою надійної автентифікації користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com.

2. AWS documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/index.html?nc2=h_ql_doc_do_v.

3. Uchit Vyas. Mastering AWS Development, 2015.

4. Young M. Implementing Cloud Design Patterns for AWS, 2015.

5. Wadia Y. AWS Administration – The Definitive Guide, 2016.

6. Amazon Elastic Compute Cloud Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

7. Amazon Elastic Container Service Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-dg.pdf.

8. Amazon Cognito Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-dg.pdf#cognito-user-identity-pools.

9. Amazon Virtual Private Cloud Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-ug.pdf.

10. Elastic Load Balancing Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/elasticloadbalancing/?id=docs_gateway.

11. Key Management Service Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/kms/latest/developerguide/kms-dg.pdf.

12. Certificate Manger Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/acm/latest/userguide/acm-ug.pdf.

13. AWS Identity and Access Management Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/IAM/latest/UserGuide/iam-ug.pdf.

14. Guidelines on Security and Privacy in Public Cloud Computing , NIST SP800-144, 2011.

15. Security Guidance for Critical Areas of Focus in Cloud Computing, Version 3.0. Technical report, Cloud Security Alliance, 2011. [Електронний ресурс] – Режим доступу: http://www.cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf.

16. Ліпкан В.А. Національна безпека України: Навч. посіб. – 2-ге вид. – К., 2009.

17. Про прийняття за основу проекту Закону України «Про Концепцію державної інформаційної»

18. The NIST Definition of Cloud Computing, NIST Special Publication 800-145, 2011.

19. Django web framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com

20. Django Rest Framework [Електронний ресурс] – Режим доступу до ресурсу: https://www.django-rest-framework.org

21. JWT [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/JSON_Web_Token

22. NAT [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/NAT

23. OWASP [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/

24. Terraform&Terragrunt [Електронний ресурс] – Режим доступу до ресурсу: https://www.terraform.io