Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування інституту, назва факультету)

Кафедра інформаційної безпеки та наноелектроніки
(повна назва кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістра
(ступінь вищої освіти)

на тему Розгортання захищеного корпоративного сервера у хмарній інфраструктурі
(назва теми)

Deploying a secure corporate server in a cloud infrastructure

Виконав: студент 2 курсу, групи БК-814м
Спеціальності 125 Кібербезпека та захист інформації
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Безпека інформаційних і комунікаційних систем
ШТАЦЬКИЙ Г.К.
(ПРІЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент САМОЙЛИК С.С.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій

Кафедра інформаційної безпеки та наноелектроніки

Ступінь вищої освіти магістр

Спеціальність 125 Кібербезпека та захист інформації

(код і найменування)

Освітня програма (спеціалізація) Безпека інформаційних і телекомунікаційних систем

(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІБтаН, к.ф.-м.н., доц.

Андрій КОРОТУН

“_____” _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ШТАЦЬКОГО Гліба Костянтиновича

(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розгортання захищеного корпоративного сервера у хмарній інфраструктурі

Deploying a secure corporate server in a cloud infrastructure

керівник проєкту (роботи) к.т.н., доцент НЕЛАСА Ганна Вікторівна,

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові,)

затверджені наказом закладу вищої освіти від «26» листопада 2025 року № 530

2. Строк подання студентом проєкту (роботи) 01 грудня 2025 року

3. Вихідні дані до проєкту (роботи) хмарна інфраструктура на платформі AWS з потребою налаштування механізмів шифрування KMS, захисту бази даних у приватній мережі та автоматизації резервного копіювання.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): 1. Історія виникнення та розвитку хмарних технологій. 2. Огляд платформи AWS для розгортання корпоративного веб-сервера. 3. Застосування криптографічних методів у хмарній інфраструктурі AWS. 4. Налаштування корпоративного сервера з комплексом засобів кібербезпеки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): Схеми архітектури мережі та налаштувань безпеки, огляд сервісів AWS презентація у Microsoft PowerPoint (15 слайдів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Розділи 1-4	НЕЛАСА Г.В., к.т.н. доцент	05.09.25	23.11.25
Нормоконтроль	КОРОЛЬКОВ Р.Ю., к.т.н. доцент	03.12.25	05.12.25

7. Дата видачі завдання « 05 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів дипломногопроєкту (роботи)	Строк виконання етапів проєкту(роботи)	Примітка
1	Ознайомлення із завданням і підбір технічної літератури	1 тиждень	Виконано
2	Огляд історії розвитку хмарних технологій та вибір платформи AWS	1 тиждень	Виконано
3	Аналіз можливостей платформи AWS для корпоративної інфраструктури.	2 тиждень	Виконано
4	Розробка архітектури захищеного корпоративного веб-сервера в AWS	3-4 тижні	Виконано
5	Налаштування ключових сервісів: EC2, RDS, S3, IAM, VPC	5 тиждень	Виконано
6	Реалізація заходів з інформаційної безпеки та шифрування даних	6 тиждень	Виконано
7	Дослідження криптографічних методів захисту в AWS	7-8 тижні	Виконано
8	Оформлення звіту, підготовка висновків та демонстрація результатів	9 тиждень	

Студент(ка) _____ Гліб ШТАЦЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Ганна НЕЛАСА
(підпис) (Ім'я ПРІЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до дипломного проєкту: 102 с., 55 рис., 22 джерела.

AWS, ВІРТУАЛЬНИЙ СЕРВЕР, ВІРТУАЛЬНІ МЕРЕЖІ, ЗАХИСТ ДАНИХ, ІНФОРМАЦІЙНА БЕЗПЕКА, КІБЕРЗАХИСТ, КРИПТОГРАФІЯ, РЕЗЕРВНЕ КОПІЮВАННЯ, ХМАРНА ІНФРАСТРУКТУРА, ХМАРНІ ОБЧИСЛЕННЯ

Предмет дослідження - хмарна інфраструктура Amazon Web Services (AWS) та засоби забезпечення інформаційної безпеки корпоративного веб-сервера.

Мета роботи - спроєктувати та реалізувати безпечний веб-сервер корпоративного рівня в середовищі AWS, дослідити сучасні методи захисту даних у хмарі, зокрема інструменти шифрування, управління доступом і резервного копіювання.

Наукова новизна роботи полягає в поєднанні сучасної хмарної інфраструктури з комплексною багаторівневою системою кіберзахисту, включаючи впровадження криптографічних механізмів та автоматизованого моніторингу в межах одного рішення.

Практична цінність полягає у можливості застосування реалізованої інфраструктури для побудови безпечних серверів у хмарному середовищі, що може бути адаптовано для реальних потреб організацій. Результати можуть бути використані для навчання фахівців та впроваджені в адміністративну практику для захисту ІТ-систем від кіберзагроз.

ABSTRACT

Explanation Note for the Thesis Project: 102 pp., 55 fig., 22 sources.

AWS, VIRTUAL SERVER, VIRTUAL NETWORKS, DATA PROTECTION, INFORMATION SECURITY, CYBERSECURITY, CRYPTOGRAPHY, BACKUP, CLOUD INFRASTRUCTURE, CLOUD COMPUTING

Subject of the study is the cloud infrastructure of Amazon Web Services (AWS) and the tools for ensuring the information security of a corporate web server.

The goal of the work is to design and implement a secure web server within the corporate AWS environment, investigate modern data protection methods in the cloud, particularly encryption tools, access management, and backup.

The scientific novelty of the work lies in the combination of modern cloud infrastructure with a comprehensive multi-layered cyber defense system, including the implementation of cryptographic mechanisms and automated monitoring of each solution.

The practical value lies in the possibility of applying the implemented infrastructure for building secure servers in a cloud environment, which can be adapted for the actual needs of organizations. The results can be used for training specialists and implemented in administrative practice for protecting IT systems against cyber threats.

ПЕРЕЛІК СКОРОЧЕНЬ

БД - База даних;

ЗД - Зона доступності;

ІТ - Інформаційні технології;

ОС - Операційна система;

ПК - Персональний комп’ютер;

СУБД - Система управління базами даних;

ШІ - Штучний інтелект;

ACM - AWS Certificate Manager - Менеджер сертифікатів AWS;

AES - Advanced Encryption Standard - Покращений стандарт шифрування;

AMI - Amazon Machine Image - Образ машини Amazon;

API - Application Programming Interface - Програмний інтерфейс прикладних програм;

ARN - Amazon Resource Name - Назва ресурсу Amazon;

ASP - Application Service Provider - Постачальник прикладних послуг;

AWS - Amazon Web Services - Вебсервіси Amazon;

AZ - Availability Zone - Зона доступності;

CA - Certificate Authority - Центр сертифікації;

CI/CD - Continuous Integration / Continuous Deployment - Безперервна інтеграція / Безперервне розгортання;

CIDR - Classless Inter-Domain Routing - Безкласова міждоменна маршрутизація;

CLI - Command Line Interface - Інтерфейс командного рядка;

CPU - Central Processing Unit - Центральний процесор;

CRM - Customer Relationship Management - Система управління відносинами з клієнтами;

DNS - Domain Name System - Система доменних імен;

EBS - Elastic Block Store - Еластичне блокове сховище;

EC2 - Elastic Compute Cloud - Еластична хмара обчислень;

FIPS - Federal Information Processing Standards - Федеральні стандарти обробки інформації;

HSM - Hardware Security Module - Апаратний модуль безпеки;

HTTP - HyperText Transfer Protocol - Протокол передачі гіпертексту;

HTTPS - HyperText Transfer Protocol Secure - Захищений протокол передачі гіпертексту;

IaaS - Infrastructure as a Service - Інфраструктура як послуга;

IAM - Identity and Access Management - Управління ідентифікацією та доступом;

IP - Internet Protocol - Протокол інтернету;

JSON - JavaScript Object Notation - Об'єктна нотація JavaScript;

KMS - Key Management Service - Служба управління ключами;

LTS - Long Term Support - Довгострокова підтримка;

MFA - Multi-Factor Authentication - Багатофакторна автентифікація;

PaaS - Platform as a Service - Платформа як послуга;

RAM - Random Access Memory - Оперативна пам'ять;

RDS - Relational Database Service - Служба реляційних баз даних;

RSA - Rivest-Shamir-Adleman - Алгоритм Рівіста-Шаміра-Адлемана;

S3 - Simple Storage Service - Проста служба зберігання;

SaaS - Software as a Service - Програмне забезпечення як послуга;

SNS - Simple Notification Service - Проста служба сповіщень;

SQL - Structured Query Language - Мова структурованих запитів;

SSH - Secure Shell - Захищена оболонка;

SSL - Secure Sockets Layer - Протокол захищених сокетів;

TCP/IP - Transmission Control Protocol / Internet Protocol - Протокол керування передачею / Протокол інтернету;

TLS - Transport Layer Security - Захист транспортного рівня;

TOTP - Time-based One-Time Password - Одноразовий пароль на основі часу;

VPC - Virtual Private Cloud - Віртуальна приватна хмара;

WWW - World Wide Web - Всесвітня мережа.

ЗМІСТ

		С.
	Вступ……..	10
1	Історія виникнення та розвитку хмарних технологій……..…	13
1.1	Передумови появи хмарних обчислень……..	14
1.2	Історичні етапи розвитку хмарних технологій……..	16
2	Огляд платформи AWS для розгортання корпоративного веб-сервера……...	27
2.1	Глобальна інфраструктура AWS та принципи хмарної платформи…	27
2.2	Архітектура рішення та використані сервіси AWS……...	30
3	Застосування криптографічних методів у хмарній інфраструктурі AWS……..…..	43
3.1	SSL/TLS та HTTPS (сертифікат Let's Encrypt)……...	43
3.2	Шифрування дисків EC2 (Amazon EBS Encryption)……..	45
3.3	Шифрування резервних копій у Amazon S3 (SSE-S3 та SSE-KMS)…	47
3.4	AWS KMS – сервіс керування криптографічними ключами……	51
3.5	Багатофакторна автентифікація (MFA) для доступу до AWS……..	52
4	Налаштування корпоративного сервера з комплексом засобів кібербезпеки…….……	57
4.1	Мережева інфраструктура (VPC).....	57
4.2	Запуск сервера (EC2).....	60
4.3	Перетворення "пустої коробки" на Веб-сервер……..…..	65
4.4	Створення Бази Даних (Amazon RDS).....	69
4.5	Створення бакету в Amazon S3.....	75
4.6	Створення адміністратора IAM.....	76

4.7 Створення Ролі IAM для сервера..... 77
4.8 Створення ключа шифрування (AWS KMS)……..... 78
4.9 Тестування зв'язку……..……. 80
4.10 Перевірка зв'язку EC2 з RDS..... 81
4.11 AWS Secret Manager..... 83
4.12 Удосконалення політик безпеки…….. 87
4.13 Автоматизація резервного копіювання……..…... 88
4.14 Зовнішній доступ та HTTPS 90
4.15 Моніторинг та аудит..... 93
Висновки……..... 98
Перелік джерел посилання…… 100

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується стрімким переходом від традиційних локальних обчислювальних систем до хмарних інфраструктур. Хмарні обчислення стали фундаментом цифрової трансформації бізнесу, дозволяючи компаніям будь-якого масштабу отримувати доступ до потужних ресурсів за моделлю «pay-as-you-go», що суттєво знижує капітальні витрати та підвищує гнучкість управління IT-сервісами. Проте, разом із перевагами масштабованості та доступності, використання хмарних середовищ висуває нові виклики у сфері кібербезпеки.

Питання захисту корпоративних даних у хмарі є критично важливим, оскільки традиційні периметрові засоби захисту стають недостатніми в умовах розподіленої інфраструктури. Згідно з моделлю спільної відповідальності (Shared Responsibility Model), хмарний провайдер, такий як Amazon Web Services (AWS), гарантує безпеку фізичного обладнання та інфраструктури, тоді як за захист даних «у хмарі», налаштування серверів, шифрування та управління доступом відповідає безпосередньо клієнт. Саме тому розробка та впровадження комплексних багаторівневих систем захисту для корпоративних серверів є актуальним науково-практичним завданням для фахівців із кібербезпеки.

Популярність платформи AWS, яка є лідером ринку хмарних послуг, зумовлює необхідність детального вивчення її інструментів безпеки, таких як шифрування KMS, ізоляція мереж VPC, управління доступом IAM та автоматизований моніторинг. Необхідність поєднання цих інструментів у єдину захищену екосистему для забезпечення безперервності бізнесу та цілісності інформації визначає вибір теми даної роботи.

Об’єкт дослідження - хмарна інфраструктура Amazon Web Services (AWS) та засоби забезпечення інформаційної безпеки корпоративного веб-сервера.

Предмет дослідження - методи, алгоритми та програмні інструменти для проєктування та реалізації захищеного серверного середовища у хмарі, зокрема механізми шифрування, управління ідентифікацією та автоматизації безпеки.

Мета роботи - спроєктувати та практично реалізувати захищений веб-сервер корпоративного рівня в середовищі AWS, дослідити сучасні методи захисту даних у хмарі та впровадити комплексну систему кіберзахисту, що включає інструменти шифрування, суворого контролю доступу та автоматизованого резервного копіювання.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати історію розвитку хмарних технологій та їхній сучасний стан, виділивши ключові моделі обслуговування (IaaS, PaaS, SaaS).

2. Провести огляд глобальної інфраструктури та сервісів платформи AWS, що використовуються для розгортання корпоративних систем.

3. Дослідити криптографічні методи захисту, які доступні в хмарному середовищі, зокрема протоколи TLS/SSL та сервіси управління ключами.

4. Спроектувати архітектуру захищеної мережі VPC з розділенням на публічні та приватні сегменти.

5. Реалізувати практичне налаштування веб-сервера EC2 та бази даних RDS із застосуванням принципу найменших привілеїв.

6. Впровадити систему автоматизованого моніторингу та аудиту дій у хмарному акаунті для проактивного виявлення загроз.

Наукова новизна роботи полягає у розробці та впровадженні цілісної архітектурної моделі захисту, яка інтегрує сучасну хмарну інфраструктуру з багаторівневою системою кібербезпеки. Це включає одночасне використання ізоляції мережевих рівнів, безпарольної автентифікації між сервісами через IAM-ролі, наскрізного шифрування даних на спочинку та в транзиті, а також інтелектуального моніторингу аномалій.

Практичне значення отриманих результатів. Результати дослідження можуть бути безпосередньо використані організаціями для побудови

безпечних хмарних серверів за готовим шаблоном. Реалізована інфраструктура забезпечує високий рівень захисту від типових кіберзагроз, мінімізує ризики витоку даних завдяки автоматизації безпекових процесів та дозволяє значно знизити витрати на обслуговування IT-систем.

Структура та обсяг роботи. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз історії виникнення та еволюції хмарних обчислень. Розглянуто ключові етапи трансформації від локальних систем до сучасних хмарних інфраструктур, а також детально описано моделі обслуговування IaaS, PaaS та SaaS.

Другий розділ присвячено детальному огляду платформи Amazon Web Services (AWS). Описано глобальну інфраструктуру хмарного провайдера та функціональні можливості основних сервісів, таких як EC2 та RDS, що використовуються для розгортання корпоративних вебсерверів.

У третьому розділі досліджено застосування криптографічних методів у хмарному середовищі. Особливу увагу приділено механізмам шифрування даних, роботі з менеджером ключів AWS KMS, а також використанню протоколів TLS/SSL для захисту каналів зв'язку.

Четвертий розділ містить практичну частину роботи, де описано процес проектування та налаштування корпоративного сервера. Розглянуто побудову захищеної архітектури мережі VPC, впровадження комплексу засобів кібербезпеки, налаштування IAM-ролей для контролю доступу та систем автоматизованого моніторингу для виявлення загроз у реальному часі.

1 ІСТОРІЯ ВИНИКНЕННЯ ТА РОЗВИТКУ ХМАРНИХ ТЕХНОЛОГІЙ

Хмарні технології стали основою сучасної IT-інфраструктури, дозволяючи користувачам отримувати доступ до обчислювальних ресурсів віддалених серверів і даних через мережу Інтернет як до окремої послуги. Такий підхід знімає потребу встановлювати програмне забезпечення локально та утримувати дороговартісні дата-центри – усі ресурси надаються провайдером і доступні з будь-якого пристрою який має підключення до глобальної мережі. Сьогодні хмарні технології проникають в усі сфери: від стримінгових сервісів і онлайн-сховищ до корпоративних бізнес-додатків та наукових проєктів. У цьому розгляді буде розкрито ключові етапи історичного розвитку хмарних обчислень, технічні передумови їх появи, сучасний стан галузі (моделі IaaS, PaaS, SaaS, основні провайдери, Kubernetes, контейнери, CI/CD, питання безпеки та масштабованості), а також прогноз подальшого розвитку (мультихмарність, edge computing, впровадження ШІ. Перехід до хмар забезпечує безперервність бізнесу та гнучке масштабування ресурсів. Це докорінна зміна управління даними, що ставить у пріоритет їхню глобальну доступність, безпеку та ефективність. На рис. 1.1 зображена концепція та типи сервісів хмарних обчислень.

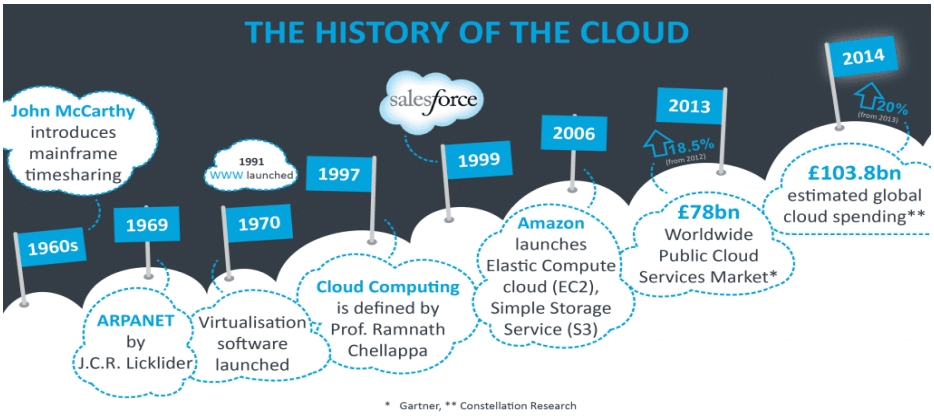


Рисунок 1.1 – Еволюція хмарних технологій [1]

1.1 Передумови появи хмарних обчислень

Для виникнення хмарних технологій склалися певні технічні передумови, що сформували базу сучасної моделі cloud computing. Однією з перших ідей була концепція обчислень як комунальної послуги, висунута ще в 1960-х роках американським науковцем Джоном Маккарті. Він припустив, що в майбутньому комп'ютерні обчислення зможуть надаватися подібно до комунальних сервісів – тобто за потребою та за плату, як електрика чи вода та будуть доступні кожному. Ця думка фактично передбачила модель, за якою програми і обчислювальні потужності доступні користувачам як зовнішня послуга, і стала ідейним підґрунтям для сучасного SaaS (Software as a Service). На рисунку 1.2 зображена схема архітектури та етапів розгортання хмарної інфраструктури.

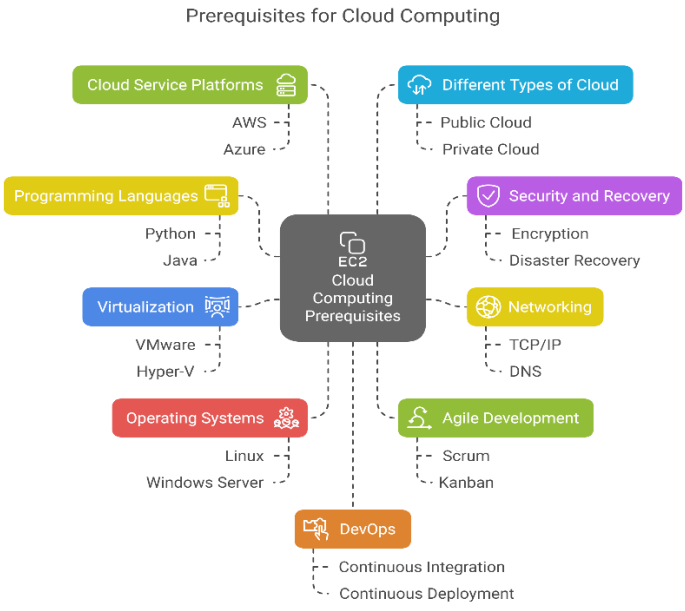


Рисунок 1.2 – Технології які сприяли появі хмарних обчислень [2]

Іншим важливим підґрунтям була поява концепції розподілених обчислень. Ще наприкінці 1960-х Джозеф Ліклайдер, працюючи над

проєктом ARPANET, представив бачення «міжгалактичної комп'ютерної мережі», де користувачі будь-де у світі зможуть бути взаємопов'язані та отримувати доступ до програм і даних з будь-якого вузла мережі. Ідеї глобальних мереж передачі даних втілилися у створенні ARPANET (1969), першої пакетно-комутованої мережі, що продемонструвала можливість спільного використання ресурсів на відстані. Хоча ARPANET безпосередньо не надавала хмарних сервісів, вона стала ключовим кроком у розвитку Інтернету і розподілених систем, показавши, як віддалені користувачі можуть спільно користуватися обчислювальними ресурсами через мережу.

Подальший розвиток мережевих технологій створив необхідну інфраструктуру для майбутньої «хмари». Зокрема, розробка універсальних мережевих протоколів передачі даних. У 1970-х були запропоновані протоколи TCP/IP, які до 1983 року стали стандартом для ARPANET й основою Інтернету. TCP/IP забезпечили надійну міжмережеву взаємодію і універсальний транспорт даних, що заклало фундамент глобальної мережі, здатної підтримувати віддалений доступ до спільних ресурсів. Одночасно з цим розвивалися моделі клієнт-сервер – в 1980-х з появою персональних комп'ютерів і локальних мереж компанії та установи почали розгортати внутрішні сервери для надання спільного доступу до додатків і даних. Така децентралізація обчислень підготувала IT-спільноту до ідеї, що обчислення можуть відбуватися віддалено і спільно, а не лише на локальних пристроях. Нарешті, критичною технічною передумовою стала віртуалізація – технологія, що дозволяє створювати на одному фізичному комп'ютері декілька ізольованих віртуальних машин. Перші кроки тут зробила IBM ще на початку 1970-х, випустивши систему VM/370, яка дала змогу розділити ресурси мейнфрейма між багатьма користувачами одночасно. Віртуалізація продемонструвала принцип, що декілька незалежних середовищ можуть співіснувати на одній фізичній інфраструктурі, динамічно розподіляючи ресурси за потреби. Цей принцип ізоляції та мультиоренди (multi-tenancy) став наріжним каменем хмарних обчислень. Удосконалення віртуалізації

продовжилося в 1990-х: з’явилися програмні рішення для звичайних серверів, зокрема VMware успішно віртуалізувала архітектуру x86 у 1999 році. Таким чином, до початку 2000-х усі необхідні компоненти – глобальний Інтернет, протоколи передавання даних, дешеві комп’ютери, технології віртуалізації та розподіленого зберігання – вже існували, щоб підтримати модель надання обчислювальних ресурсів віддалено на основі моделі "pay-as-you-go" (плати за використане). На рисунку 1.3 зображені основні моделі та переваги хмарних обчислень.

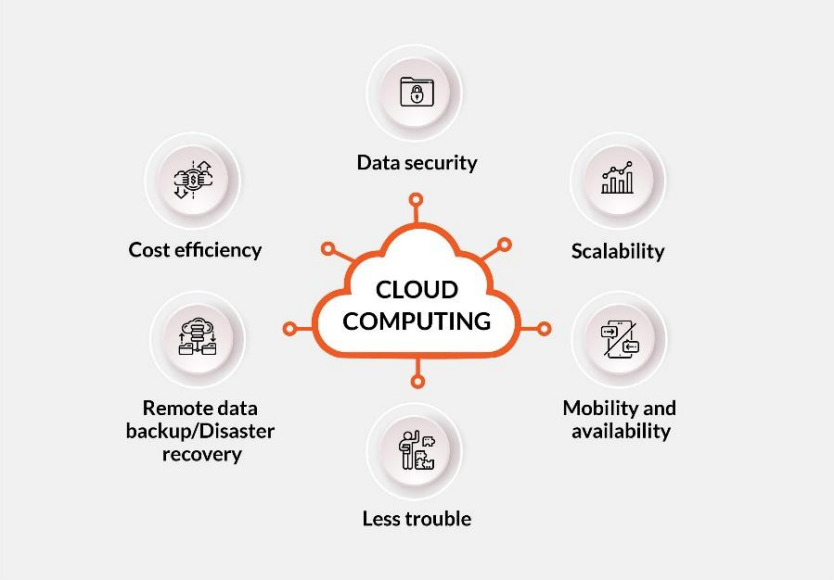


Рисунок 1.3 – Переваги хмарних технологій [3]

1.2 Історичні етапи розвитку хмарних технологій

1950–1960-ті: мейнфрейми, таймшеринґ і зародження ідеї. Перші риси хмарних обчислень проглядалися вже в епоху мейнфреймів. У 1950-х роках компанія IBM почала впроваджувати потужні великі комп’ютери – мейнфрейми, до яких підключалися віддалені термінали. Один мейнфрейм міг обслуговувати відразу кількох користувачів у режимі time-sharing (поділу

часу) – користувачі через «тупі» термінали діставали обмежений доступ до центрального комп’ютера. Це дозволило значно ефективніше використовувати вартісні обчислювальні машини: більшість задач не потребували повної потужності мейнфрейма, тож одночасна робота десятків користувачів зробила обчислення економічно доступнішими. Вже в середині 1960-х з’явилися комерційні сервіси time-sharing, які надавали бізнесу оренду обчислювальних потужностей на віддалених машинах на годинній основі. Цей підхід «обчислення як сервіс» був предтечею хмар: клієнти могли користуватися далеким суперкомп’ютером, не володіючи ним. На рисунку 1.4 зображена еволюція обчислювальної техніки: комп’ютерні системи 1970-х років

Рисунок 1.4 – Суперкомп’ютер – мейнфрейм [4]

Паралельно, у другій половині 60-х зароджувалися ідеї глобальних комп’ютерних мереж. 1969 року за сприяння Агентства перспективних дослідницьких проєктів США було запущено ARPANET – першу мережу з комутацією пакетів, що з’єднала університети й організації. Одним із натхненників ARPANET був Дж. Ліклайдер, який ще в 1963-64 рр. в агенції

DARPA виклав концепцію «Інтергалактичної мережі» – прообразу Інтернету. ARPANET на практиці показала, як мережа може зв’язувати різні комп’ютери і користувачів, дозволяючи передавати дані та віддалено користуватися ресурсами. Хоча в ті часи не йшлося про комерційні хмарні сервіси, ARPANET заклала основу для глобальної мережевої інфраструктури, без якої хмарні обчислення неможливі.

1960-ті дали і важливу ідею утилітарного обчислення: Джон Маккарті висловив припущення, що колись обчислювальні потужності будуть доступні як комунальна послуга на зразок телефонії чи електрики. Його слова фактично передбачили появу бізнес-моделі, де клієнт купує не обладнання, а обчислення як сервіс. Ця ідея залишалася теоретичною понад три десятиліття, але зрештою реалізувалася в моделі хмарних обчислень (особливо SaaS).

1970–1980-ті: мережі, інтернет і віртуалізація. У 1970-х розвиток обчислювальних систем пішов двома взаємопов’язаними шляхами, які готували ґрунт для хмар. Перший – мережеві технології. ARPANET швидко розширювався: вже у 1973 році до нього підключено перші закордонні вузли (у Великобританії та Норвегії), що продемонструвало можливість глобального комп’ютерного сполучення. Для уніфікації мережевого обміну було розроблено стек протоколів TCP/IP (опублікований в 1974 р., стандартизований до 1983 р.), який став основою функціонування Інтернету. 1983 року ARPANET повністю перейшов на TCP/IP, що ознаменувало народження сучасного Інтернету – глобальної мережі, через яку згодом надаватимуться хмарні сервіси. 1989 року з’явилася служба Всесвітньої павутини (WWW), яка суттєво спростила доступ до інформації онлайн. Таким чином, до початку 1990-х інтернет-зв’язність і протоколи були достатньо зрілими, щоб підтримати віддалену роботу з даними і програмами – ключову вимогу для cloud computing. На рисунку 1.4 зображена історія становлення мережі ARPAnet та початкові протоколи передачі даних

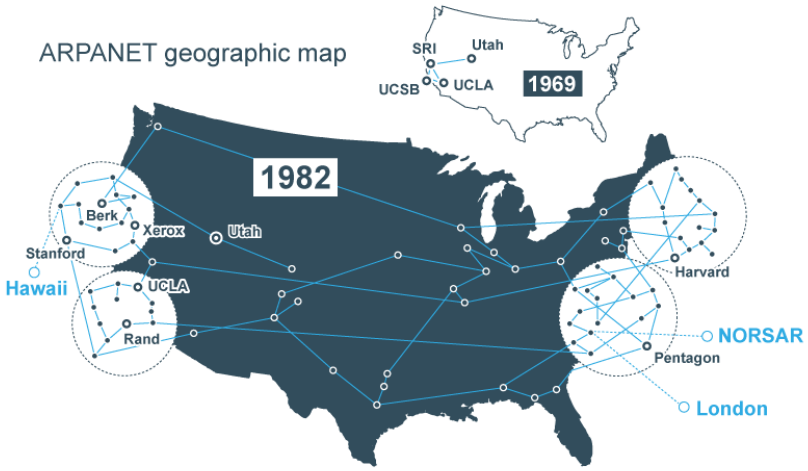


Рисунок 1.5 – Маршрути ARPANET у 1980х роках [5]

Окрім того, у 1980-х стрімко розвивалися локальні мережі (LAN) та клієнт-серверні системи. З появою персональних комп'ютерів багато завдань перейшли з мейнфреймів на сервери меншого масштабу, що обслуговували ПК-клієнти в межах організацій. Це розосередило обчислення, але паралельно сформувало розуміння, що дані можуть зберігатися централізовано (на сервері), а користувачі під'єднуються віддалено. Фактично, клієнт-серверна архітектура стала прообразом хмарної моделі: сервер забезпечує ресурси/сервіси, клієнт споживає їх через мережу. До кінця 1980-х років, з ростом потужності комп'ютерів і мереж, технічно стало можливим реалізувати мрію Маккарті – надавати обчислення як послугу, хоча поки що це не було втілено комерційно.

1990-ті: поява терміну «хмара» і перші інтернет-сервіси. 1990-ті роки – період, коли з'являються безпосередні попередники сучасного хмарного бізнесу. По-перше, це доба широкого розповсюдження Інтернету серед бізнесу і населення. Після винайдення World Wide Web у 1989 р. та появи графічних веб-браузерів (Mosaic у 1993-му) інтернет став масовим середовищем для передачі даних. Це створило плодючий ґрунт для віддалених сервісів: користувачі звикали до того, що інформація і програми можуть знаходитися не на їхньому ПК, а «десь у мережі».

По-друге, в цей період формуються перші онлайн-сервіси та бізнес-моделі, дуже близькі до хмарних. З’являються так звані постачальники прикладних сервісів (ASP, Application Service Providers) – компанії, що пропонують доступ до програм через інтернет на основі підписки. Знаковою подією стало заснування в 1999 році компанії Salesforce.com, яка почала надавати свій програмний продукт (CRM-система для управління взаєминами з клієнтами) повністю через веб-сайт на умовах підписки. Salesforce стала одним із перших успішних прикладів моделі SaaS – коли програмне забезпечення розміщене в провайдера і клієнти користуються ним через інтернет, не встановлюючи локально. Фактично, Salesforce продемонструвала життєздатність концепції “software as a service” у комерційному масштабі, і її успіх підготував ринок до сприйняття хмарних рішень у 2000-х. На рисунку 1.6 зображена концепція програмного забезпечення як послуги (SaaS) та її ключові характеристики.

Рисунок 1.6 – SaaS, як послуга для кожного [6]

Сам термін «хмарні обчислення» також уперше з’явився у 1990-х. Вважається, що слово cloud в IT-контексті почали вживати через схему позначення мережі Інтернет у вигляді хмари на діаграмах. У 1996 році

компанія Compaq використала вираз “cloud computing” у внутрішньому документі, а в 1997 професор Рамнат Челлапа дав одне з перших формальних визначень хмарних обчислень, описавши їх як «обчислювальну парадигму, де межі обчислень визначаються економічною доцільністю, а не технічними можливостями». Тобто ресурси можуть динамічно постачатися і масштабуватися під запит, виходячи з потреб користувача та економічних чинників, що дуже точно відображає сутність сучасного cloud computing.

Ще одним важливим кроком стало вдосконалення технологій віртуалізації та серверної інфраструктури наприкінці 90-х. Вже згаданий VMware (1999) дав можливість ефективно створювати віртуальні машини на звичайних серверах x86. Це відкривало шлях до побудови великих кластерів стандартних серверів, які можуть гнучко розподілятися між різними завданнями та клієнтами. Також наприкінці 90-х були запущені проекти розподілених обчислень, наприклад SETI@home (1999) – який використовував комп’ютери добровольців по всьому світу для аналізу даних, що продемонструвало потенціал глобального “пулу” ресурсів. У сукупності, до 2000 року склалися як термінологічні, так і технологічні передумови для народження комерційних хмарних платформ.

2000-ті: народження сучасних хмарних платформ. Початок 2000-х ознаменувався вибухом інтернет-бізнесів і одночасно лопанням бульбашки dot-com у 2000–2001 роках. Попри загальний спад, саме в цей період з’являються перші справжні хмарні сервіси, які визначили обличчя галузі. Amazon.com, успішний онлайн-рітейлер, став піонером інфраструктурних хмарних послуг. Використовуючи потужну внутрішню IT-інфраструктуру, побудовану для власних потреб, Amazon вирішив монетизувати її надлишкові ресурси, здаючи їх в оренду зовнішнім користувачам. У 2002 році компанія запустила перші веб-служби під брендом Amazon Web Services (AWS), які дозволяли розробникам інтегрувати функції Amazon у свої сайти. А повноцінним стартом хмарної платформи AWS став 2006 рік, коли були офіційно представлені сервіси Amazon S3 (Simple Storage Service) для

зберігання даних і Amazon EC2 (Elastic Compute Cloud) для оренди віртуальних серверів. Запуск EC2 став поворотним моментом: вперше будь-який бажаючий міг за лічені хвилини орендувати через інтернет віддалений сервер потрібної потужності і платити лише за час його роботи. Цей новий підхід до інфраструктури – Infrastructure as a Service (IaaS) – швидко набув популярності і заклав основу індустрії публічних хмарних обчислень.

Успіх Amazon підштовхнув інших технологічних гігантів долучитися до хмарних рішень. У 2008 році Google запустив сервіс Google App Engine, що став одним із перших Platform as a Service (PaaS) продуктів: розробники отримували платформу для запуску своїх веб-додатків без потреби адмініструвати сервери. Google, маючи масштабні дата-центри для власних сервісів, використав їх для надання хмарної платформи стороннім розробникам – цей крок суттєво популяризував ідею PaaS.

Компанія Microsoft спершу придивлялася до нового ринку, але невдовзі також приєдналася: у 2008 році було анонсовано платформу Microsoft Azure (запущена комерційно у 2010-му). Azure починалася як платформа для розробки (.NET-середовище як сервіс), але згодом розрослася до повноцінної хмарної екосистеми, що включає і IaaS (віртуальні машини, сховища, мережі), і PaaS-сервіси, і навіть власні SaaS-продукти. Одночасно з'являлися й інші гравці: у 2009 свою хмарну платформу запустила IBM (IBM SmartCloud), Oracle Cloud стартував на початку 2010-х, значний розвиток отримали хмарні сервіси IBM, Alibaba, VMware, Rackspace та ін. Крім комерційних, виникли open-source проекти для розгортання хмар: перші відкриті платформи Eucalyptus і OpenNebula з'явилися вже у 2008 році, а в 2010 NASA та Rackspace започаткували проект OpenStack – відкритий стек програм для побудови власних хмар (приватних чи публічних). OpenStack став де-факто стандартом для приватних хмар у багатьох компаніях.

Таким чином, у другій половині 2000-х сформувався ринок публічних хмарних платформ з трьома безперечними лідерами – AWS, Google Cloud та Microsoft Azure. Вони запропонували різноманітні послуги на базі потужної

інфраструктури дата-центрів по всьому світу, що працюють за моделлю оплати за використання. На кінець десятиліття бізнес і розробники отримали нову парадигму: замість купівлі серверів і розміщення їх у власних серверних, можна орендувати необхідні ресурси на хмарних «фермах» Amazon, Google, Microsoft тощо, швидко масштабувати їх під свої потреби і позбутися клопоту адміністрування фізичного обладнання. На рисунку 1.7 зображена динаміка зростання частки ринку провайдерів хмарних інфраструктурних послуг.

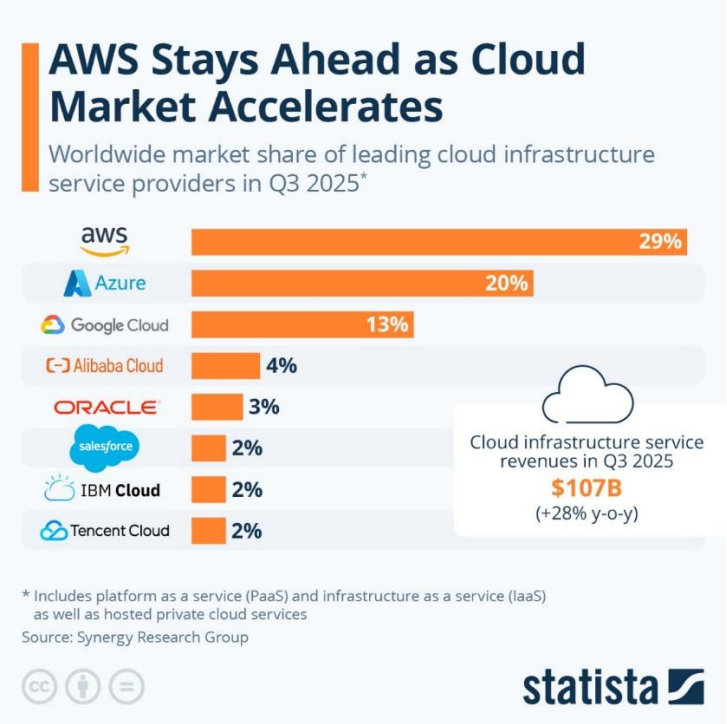


Рисунок 1.7 – Лідери ринку хмарних обчислень [7]

2010-ті: ера масової cloud-адаптації, контейнерів і DevOps. У 2010-х хмарні технології з нішевого рішення перетворилися на мейнстрім і основу IT-стратегії для компаній будь-якого розміру. На початку десятиліття стало зрозуміло, що модель хмари витримала перевірку: за даними IDC, вже у 2009 році обсяг ринку публічних хмар становив ~$17 млрд (близько 5% усього ринку IT), а далі показував вибухове зростання. Провідні гравці продовжили інтенсивний розвиток сервісів і географічне розширення своїх дата-центрів

(регіонів).

Ключовою тенденцією стало впровадження контейнеризації та оркестрації як нової основи хмарної інфраструктури. У 2013 році з’явилася технологія контейнерів Docker, що дозволила пакувати застосунки з усіма залежностями в легкі контейнери, які запускаються будь-де. Контейнери значно спростили розгортання застосунків в хмарі, підвищивши портативність і ефективність використання ресурсів. Наступним кроком було створення системи оркестрації контейнерів Kubernetes (від Google, 2014–2015 рр.), яка стала стандартом для управління контейнерними кластерами. До кінця 2010-х усі провідні хмарні провайдери впровадили підтримку контейнерів і Kubernetes як сервісу. Це дало можливість клієнтам легко розгортати хмарно-нативні мікросервісні архітектури, переносити свої робочі навантаження між різними хмарами або між хмарою і власним дата-центром (що сприяло появі моделей гібридної та мультихмарної інфраструктури). Зокрема, у 2017–2019 рр. з’явилися рішення типу Anthos (Google), Azure Arc (Microsoft), AWS Outposts, які розширюють сервіси публічної хмари на локальні середовища і об’єднують управління різнорідною інфраструктурою.

2020-ті: сьогодення хмарних технологій. У 2020-х роках хмарні обчислення досягли зрілого стану, але продовжують динамічно розвиватися та розширювати сферу застосування. Зараз вже важко знайти сферу IT чи бізнесу, яка б не використовувала хмарні сервіси. Згідно з прогнозами Gartner, глобальні витрати кінцевих споживачів на публічні хмари у 2025 році досягнуть ~$723 млрд, що на 20% більше, ніж у 2024 році. Кілька ключових аспектів сучасного стану хмарних технологій варто підкреслити:

Універсальність і домінування хмари: Для нових проєктів компанії все частіше обирають стратегію "cloud-first" – тобто передусім розгортати системи в хмарі, а не у власному дата-центрі. Політика «no-cloud» (відмова від хмари) нині велика рідкість, подібно до того, як відмова від Інтернету була б дивною для бізнесу 2000-х. Хмари проникли і в життя пересічних

користувачів – від пошти, фото, відео до офісних застосунків – багато чого тепер працює за моделлю SaaS з хмарних серверів.

Різноманітність моделей хмарних сервісів: Сформувалися три основні моделі надання хмарних послуг – IaaS, PaaS, SaaS.

Infrastructure as a Service (IaaS) – інфраструктура як сервіс, що передбачає оренду базових обчислювальних ресурсів: віртуальних машин, дисків, мереж тощо. Користувач отримує максимальний контроль над середовищем (може сам вибирати ОС, встановлювати ПЗ), але адміністрування фізичної інфраструктури виконує провайдер. Приклади IaaS: Amazon EC2, Google Compute Engine, Microsoft Azure Virtual Machines.

Platform as a Service (PaaS) – платформа як сервіс. Провайдер надає готову платформу для розробки та виконання застосунків: середовища виконання, бази даних, сервер застосунків, інструменти розробки. Розробник зосереджується на коді, а не на тому, як налаштувати сервери. Приклади: Google App Engine, Heroku, Azure App Service. PaaS прискорює розробку, але менш гнучкий щодо нестандартних налаштувань.

Software as a Service (SaaS) – програмне забезпечення як сервіс. Повністю готовий застосунок на боці провайдера, яким користуються через інтернет (в браузері чи клієнті) на умовах підписки. Усі оновлення, масштабування і підтримка – на боці постачальника. Приклади: Salesforce (CRM-система), Google Workspace, Microsoft 365, Slack тощо. SaaS знімає з клієнта будь-які питання інфраструктури, натомість надає готовий функціонал «під ключ».

Вибір конкретної моделі залежить від стратегічних цілей підприємства та наявної технічної експертизи команди. Використання IaaS надає найвищий ступінь свободи в архітектурних рішеннях, тоді як PaaS та SaaS дозволяють значно скоротити час виходу продукту на ринок (Time-to-Market) за рахунок делегування рутинних завдань обслуговування хмарному провайдеру. Сучасний підхід до побудови IT-інфраструктури часто передбачає гібридне поєднання цих моделей, що забезпечує оптимальний баланс між вартістю володіння, швидкістю розробки та рівнем безпеки корпоративних даних. На

рисунку 1.8 зображена порівняльна характеристика моделей обслуговування IaaS, PaaS та SaaS.

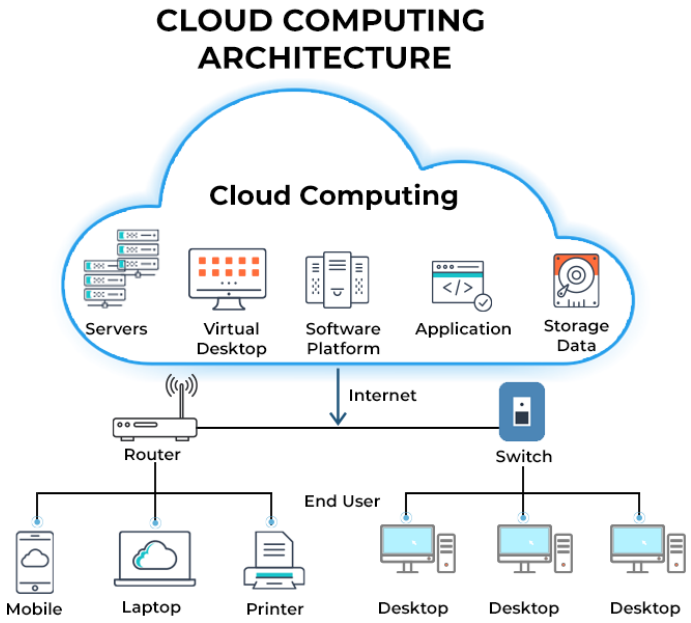


Рисунок 1.8 – Схема роботи Cloud Computing [8]

Отже, ключова відмінність між моделями IaaS, PaaS та SaaS полягає у рівні контролю над обчислювальними ресурсами та розподілі зон відповідальності між провайдером і клієнтом. Якщо IaaS надає доступ до базової інфраструктури з можливістю детального налаштування, то PaaS та SaaS дозволяють зосередитися на розробці або використанні готового ПЗ, мінімізуючи технічне обслуговування. Такий підхід дозволяє бізнесу гнучко обирати інструменти: від повного керування серверами до швидкого впровадження готових сервісів за передплатою.

2 ОГЛЯД ПЛАТФОРМИ AWS ДЛЯ РОЗГОРТАННЯ КОРПОРАТИВНОГО ВЕБ-СЕРВЕРА

2.1 Глобальна інфраструктура AWS та принципи хмарної платформи

Amazon Web Services (AWS) – це провідна хмарна платформа, яка надає більше 200 різноманітних сервісів для побудови IT-інфраструктури будь-якої складності. AWS було обрано для проєкту, оскільки ця платформа дозволяє орендувати обчислювальні ресурси за потреби без капіталовкладень у власні сервери. Таким чином, модель оплати AWS – “pay-as-you-go” – дає змогу платити тільки за фактично використані ресурси і уникнути витрат на простої. Це відповідає принципам гнучкості та масштабованості: надає можливість швидко запускати нові сервери або збільшувати сховище, коли зростають вимоги, і так само швидко згортати ресурси, коли вони вже не потрібні. AWS підтримує різні хмарні моделі (IaaS, PaaS тощо) і надає інструменти для автоматизації, що прискорює розгортання інфраструктури.

Глобальна інфраструктура AWS охоплює дата-центри по всьому світу, об’єднані в регіони та зони доступності (Availability Zones, AZ). Регіон AWS – це географічний регіон (наприклад, Європа (Франкфурт) або Східні США (Вірджинія)), в межах якого розташовано кілька зон доступності. Зона доступності – це одна або кілька ізольованих фізичних дата-центрів із незалежним живленням, охолодженням і мережею. Кожен регіон містить щонайменше три зони доступності (територіально віддалені одна від одної на десятки кілометрів), що дозволяє будувати відмовостійкі рішення. Якщо один дата-центр (AZ) виходить з ладу, інші зони в тому ж регіоні продовжать роботу, забезпечуючи безперервність сервісів. Станом на 2025 рік AWS має 38 регіонів та 120 зон доступності по всьому світу, і постійно розширюється. Така розгалужена інфраструктура гарантує низькі затримки та високу продуктивність – надає можливість розгорнути свій веб-сервер у найближчому до користувачів регіоні, щоб зменшити час відгуку. Крім того,

глобальна мережа AWS із понад 9 млн км волоконно-оптичних ліній зв’язку забезпечує швидку передачу даних між дата-центрами та до кінцевих користувачів. На рисунку 2.1 зображена детальна схема рівнів відповідальності у моделях IaaS, PaaS та SaaS

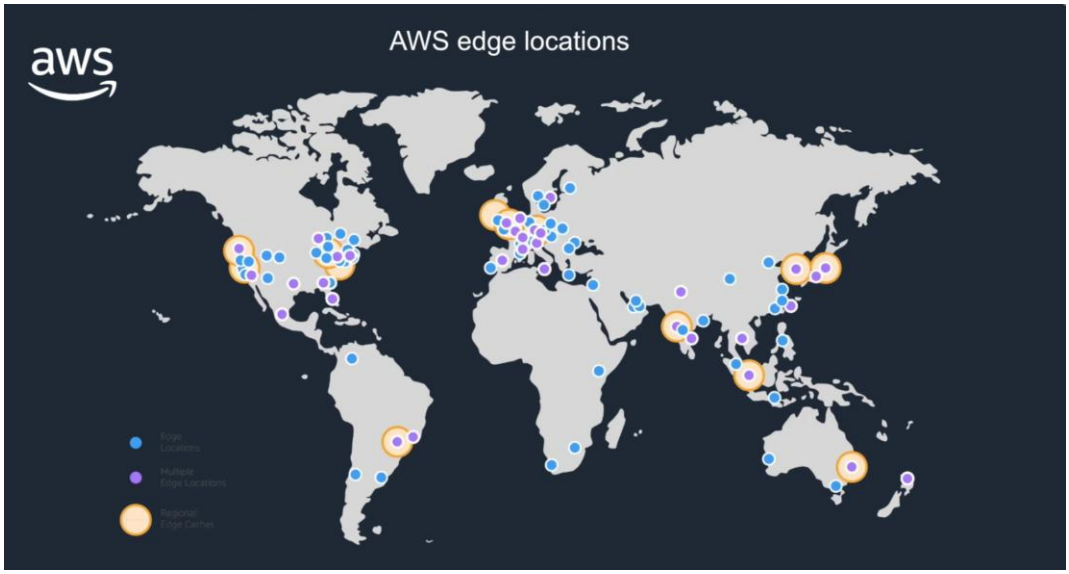


Рисунок 2.1 - Мапа дата центрів Amazon [12]

Надійність та безпека є ключовими принципами AWS. Інфраструктура AWS спроєктована з розрахунком на максимальну відмовостійкість і доступність: кожен сервіс працює на потужностях багатьох дата-центрів, а дані можуть автоматично реплікуватися між зонами. Наприклад, об’єктне сховище Amazon S3 зберігає дані відразу в кількох AZ одного регіону, що дозволяє досягти надвисокої надійності – гарантується майже на 100%. Це означає, що імовірність втрати інформації в S3 практично нульова, а сервіси залишаються доступними майже без перерв. AWS також приділяє особливу увагу безпеці: хмарна інфраструктура відповідає найсуворішим стандартам безпеки та аудиту (PCI DSS, ISO 27001, SOC та інші), дата-центри охороняються фізично і обладнані резервними системами живлення й зв’язку. Водночас AWS реалізує модель розподіленої відповідальності (Shared Responsibility) за безпеку: провайдер відповідає за захист «хмари»

(апаратне забезпечення, мережі, гіпервізори, фізична безпека), а користувач відповідає за безпеку «в хмарі» – налаштування своїх віртуальних серверів, конфігурацію брандмауерів, шифрування даних та керування доступом. Це надає гнучкість у виборі засобів захисту, але і вимагає належного адміністрування. AWS надає для цього численні інструменти: ізольовані мережі (VPC), списки контролю доступу і брандмауери (Security Groups, Network ACL), шифрування даних у транзиті та на дисках, сервіс керування доступом IAM, моніторинг та журналювання тощо. В інфраструктурі було застосовано багато з цих механізмів, щоб гарантувати захищеність веб-сервера (детально про заходи безпеки – далі у цьому та наступному розділах).

Гнучкість і масштабованість платформи AWS дозволяє ефективно адаптувати інфраструктуру під потреби проєкту. По-перше, як уже зазначено, AWS працює за моделлю on-demand: це надає можливость в будь-який момент замовити чи вимкнути ресурс (сервер, базу даних, сховище) через веб-консоль або API, і витрати будуть нараховуватися тільки за час його роботи. Це відрізняє хмару від традиційного дата-центру, де ресурси закуповуються «із запасом» і простоюють у разі зниження навантаження. По-друге, AWS пропонує широкий вибір типів ресурсів – десятки типів EC2-інстансів (з різною кількістю CPU, RAM, прискорювачами GPU тощо), різні класи сховищ даних (від надшвидких SSD до архівних сховищ на стрічці), декілька СУБД на вибір (реляційні, NoSQL, аналітичні) – тому можливо оптимально підібрати конфігурацію під свій веб-сервер. По-третє, AWS підтримує автоматичне масштабування (Auto Scaling): за потреби можна налаштувати автоматичний запуск додаткових серверів при зростанні трафіку і їх вимкнення, коли навантаження спадає. У контексті мого дипломного проєкту великих навантажень не очікується, тому Auto Scaling не застосовано, але можливість масштабування «на виріст» демонструє гнучкість рішення. Нарешті, AWS інтегрується з інструментами інфраструктури як код (IaC) та CI/CD, що спрощує розгортання і підтримку середовищ – у промислових сценаріях це дозволяє швидко впроваджувати

нові версії застосунків і підтримувати стабільність за рахунок відтворюваності конфігурацій.

Завдяки переліченим властивостям – глобальності, надійності, безпеці й гнучкості – платформа AWS стала основою для корпоративного веб-сервера. В AWS як приклад використовується Infrastructure as a Service (IaaS) для розміщення свого застосунку: весь необхідний «залізний» та мережевий фундамент надається провайдером, а адміністратор серверу зосереджується на налаштуванні серверів і сервісів під потреби сайту.

2.2 Архітектура рішення та використані сервіси AWS

У межах дипломного проєкту було реалізовано розгортання корпоративного веб-сайту на хмарній інфраструктурі AWS з дотриманням принципів безпеки. Архітектура рішення виглядає наступним чином: в ізольованій хмарній мережі Amazon VPC розгорнуто EC2-інстанс (віртуальний сервер) з операційною системою Ubuntu Server. На цьому сервері працює веб-сервер Nginx та пов’язані серверні компоненти сайту. Для зберігання даних було використано СУБД MySQL – її було розгорнуто спочатку на тому ж EC2-інстансі для простоти, але з метою надійності була передбачена можливість перенесення бази на керований сервіс Amazon RDS. Дискова підсистема сервера представлена томом Amazon EBS (SSD-накопичувач), з якого регулярно робляться знімки для резервного копіювання. Резервні копії (як файлові дампи бази та архіви сайту, так і знімки EBS) автоматично зберігаються у надійному об’єктному сховищі Amazon S3. Користувацький доступ до веб-сайту здійснюється по захищеному протоколу HTTPS – для цього був налаштован SSL-сертифікат на сервері. Доменне ім’я корпоративного сайту обслуговується через сервіс Amazon Route 53, який виконує роль DNS і спрямовує трафік на мій EC2-інстанс. Вся ця інфраструктура із самого початку спроєктована з акцентом на

безпеку: VPC ізолює сервери на мережевому рівні, Security Groups (вбудовані брандмауери) дозволяють тільки необхідний трафік (HTTP/HTTPS до веб-сервера, SSH тільки для адміністратора серверу, доступ до MySQL лише з веб-сервера), за обліковими записами та правами доступу стежить AWS IAM (включно з багатофакторною автентифікацією), а дії в обліковому записі логуються через AWS CloudTrail. Моніторинг працездатності реалізований за допомогою Amazon CloudWatch, куди надходять метрики серверу і журнали подій. Таким чином, рішення охоплює всі рівні – від обчислень і мережі до зберігання даних і безпеки – використовуючи хмарні сервіси AWS. Нижче надається детальний опис коженого із задіяних сервісів AWS та його роль у проєкті.

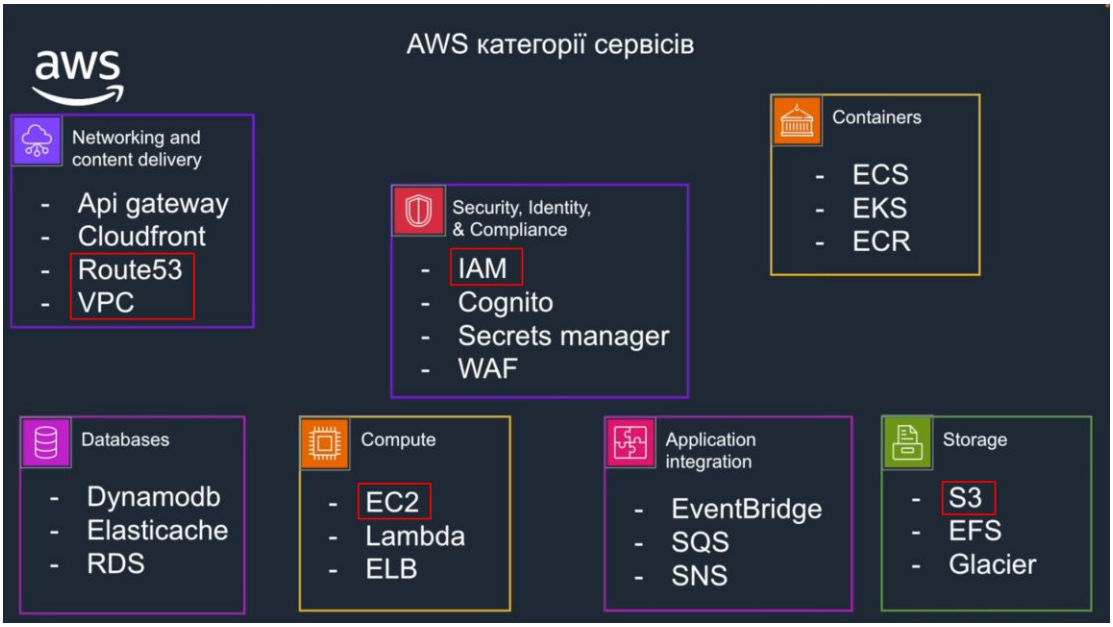


Рисунок 2.2 – Категорії сервісів AWS (у красних боксах ті використані в проєкті)

Amazon EC2 (Elastic Compute Cloud). У своїй інфраструктурі використовується Amazon EC2 як віртуальний сервер для веб-сайту. EC2 надає можливість запускати в хмарі необхідну кількість інстансів (віртуальних машин) із вибраними параметрами. Для було разгорнуто один EC2-інстанс з ОС Ubuntu 20.04, на якому працює веб-сервер Nginx та

прикладне програмне забезпечення сайту. На етапі конфігурації апаратного забезпечення мною було прийнято рішення використати інстанс нового покоління - t3.micro.

На відміну від попереднього покоління (t2), яке базувалося на гіпервізорі Xen, сімейство t3 побудоване на сучасній системі AWS Nitro System. Це забезпечує вищу продуктивність мережі та вводу-виводу.

Обрана конфігурація надає 2 віртуальні процесори (vCPU) та 1 ГБ оперативної пам'яті. Наявність двох ядер дозволяє веб-серверу ефективніше обробляти паралельні запити користувачів, що є критичним для забезпечення стабільної роботи корпоративного сайту. При цьому даний тип інстансу залишається економічно ефективним рішенням для навчальних цілей.

Важлива перевага EC2 – це гнучке керування життєвим циклом серверів: тут можливо запускати, зупиняти чи перезавантажувати віртуальний сервер у кілька кліків через консолі AWS. Більш того, плата за EC2 нараховується погодинно (або щосекундно для новіших поколінь) лише поки інстанс працює. Це означає, що користувач зможе економити кошти, вимикаючи тестові або резервні сервери на час, коли вони не використовуються. Для мого проєкту високої доступності не вимагалося, тому веб-сервер розгорнуто в однині (Single-AZ deployment). Проте, в перспективі, EC2 дає можливість легко побудувати кластер веб-серверів за балансувальником навантаження, якщо зросте кількість відвідувачів. Диски EC2-інстансу представлені томами EBS: системний диск (SSD) містить ОС і веб-додаток, його знімки (snapshots) регулярно копіюються в S3 для відмовостійкості. Таким чином, Amazon EC2 забезпечує надійну і гнучку обчислювальну платформу замість власного фізичного сервера.

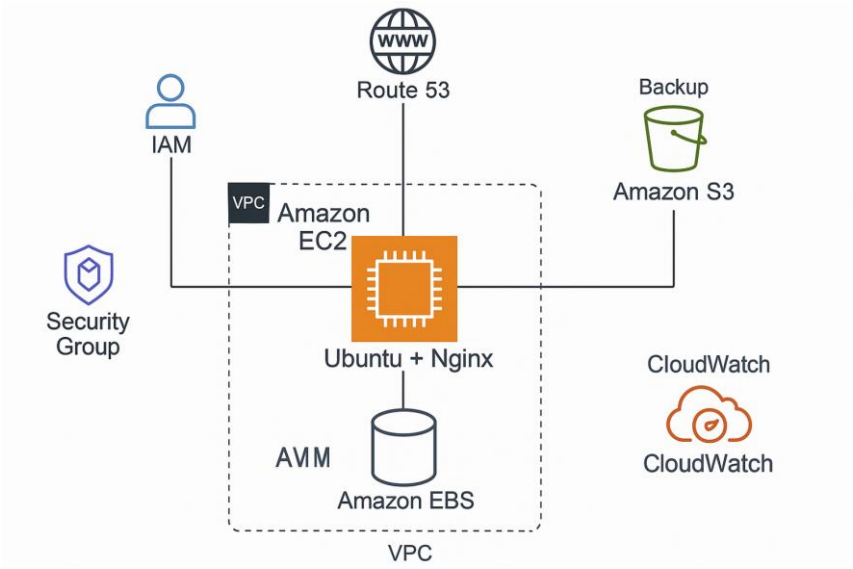


Рисунок 2.3 – Архітектура сервера на базі EC2

Amazon RDS (Relational Database Service). Для зберігання даних MySQL був використаний сервіс Amazon RDS, щоб підвищити надійність і спростити адміністрування бази даних. RDS – це керована реляційна СУБД: AWS самостійно виконує багато рутинних задач, таких як встановлення та оновлення DBMS, резервне копіювання, застосування патчів безпеки і масштабування вертикальне або горизонтальне. У межах безкоштовного річного тарифу AWS можливо розгорнути інстанс MySQL розміру db.t2.micro (або еквівалентний db.t3.micro) з об'ємом сховища до 20 ГБ безкоштовно. Для мого проекту цього достатньо, адже база даних невелика. Була налаштована RDS з параметрами: один інстанс MySQL Community Edition, 1 vCPU, 1 ГБ RAM, зберігання 20 ГБ SSD. У такій конфігурації RDS працює в одній зоні (Single-AZ), але за потреби можна увімкнути режим Multi-AZ – тоді база автоматично реплікуватиметься на іншу зону доступності в тому ж регіоні. Це підвищує відмовостійкість: якщо основний сервер БД недоступний (наприклад, через збій AZ), RDS перемкне підключення на резервну копію. Важливо, що перехід від локальної бази на EC2 до RDS є прозорим для застосунку: достатньо імпортувати дані і змінити строку підключення. Після цього користувач отримує переваги керованого

сервісу – автоматичні бекапи (RDS робить резервні копії бази щоденно і зберігає журнали транзакцій для відновлення), моніторинг продуктивності через CloudWatch, можливість легко збільшити об’єм диска або змінити тип інстансу на потужніший за кілька хвилин. Використання RDS знімає з мене значну частину операційної підтримки БД, дозволяючи зосередитися на логіці додатка. В контексті безпеки для RDS був налаштован парольний користувач з обмеженими правами та помістив БД у приватну підмережу VPC (без прямого доступу з інтернету) – доступ до MySQL можливий тільки з мого веб-сервера в тій же VPC або через VPN адміністратора.

Amazon S3 (Simple Storage Service). Для потреб зберігання файлів і резервних копій використовується Amazon S3 – це об’єктне хмарне сховище, яке відоме своєю надвисокою надійністю та практично необмеженою масштабованістю. S3 підходить для зберігання будь-яких даних (файли, образи дисків, логи і т.д.) і гарантує збереження цих даних навіть у разі відмови цілих дата-центрів. Як згадано, S3 забезпечує майже 100% довговічності даних шляхом багаторазового дублювання об’єктів на декількох пристроях щонайменше у трьох різних AZ одного регіону. У проєкті було створено окремий S3-бакет (сховище) для резервного копіювання. Туди автоматично завантажуються резервні архіви: бекапи бази даних (SQL-дампи) і бекапи файлів сайту. Крім того, AWS налаштовує збереження знімків EBS-томів також у S3 (фактично всі snapshot EC2 зберігаються як об’єкти S3, хоча і не видимі напряму). Таким чином, у випадку збою сервера чи втрати даних можливо або відновити БД з файлу бекапу, або розгорнути новий EC2-інстанс зі знімка диска, що зберігається у S3. Ще одне застосування S3 – розміщення статичного контенту (наприклад, зображень сайту, завантажуваних файлів). Хоча мій корпоративний сайт не є великим файловим сховищем, S3 потенційно можна використати як CDN-вузол для роздачі статичних файлів через мережу CloudFront, щоб зменшити навантаження на веб-сервер. S3 є відносно недорогим сервісом і підтримує версіонування об’єктів, шифрування даних “на льоту” і “на спокої”, а також

гнучкі політики життєвого циклу (наприклад, автоматичне переміщення старих бекапів в дешевший архівний клас Glacier). У нашому випадку було налаштовано політику зберігання бекапів: щоденні резервні копії за останній тиждень зберігаються в S3 Standard, старіші автоматично переміщуються до S3 Glacier для економії коштів. Ці можливості підтверджують гнучкість AWS у сфері зберігання даних.

Amazon VPC (Virtual Private Cloud). Для ізоляції інфраструктури в хмарі було створено власну віртуальну мережу за допомогою Amazon VPC. VPC – це логічно ізольований сегмент мережі всередині AWS, в межах якого повністю контролюється адресний простір, підмережі, маршрутизацію і списки контролю доступу. Іншими словами, VPC аналогічний виділеній корпоративній мережі в хмарі. Було налаштувано VPC з приватним адресним простором IPv4 (вибрав діапазон 10.0.0.0/16) і розбив його на підмережі: публічну підмережу для веб-сервера (вона має вихід в інтернет через інтернет-шлюз) та приватну підмережу для бази даних RDS (без прямого доступу з інтернету). Така сегментація підвищує безпеку: MySQL-сервер взагалі не має глобальної IP-адреси і доступний тільки усередині VPC. У рамках VPC також налаштовано таблиці маршрутизації (щоб трафік з приватної підмережі міг проходити через Network Address Translation до інтернету для оновлень тощо) і ACL (Network ACL) на рівні підмереж для додаткової фільтрації пакетів. Однією з головних переваг VPC є можливість гнучко налаштовувати Security Groups – це брандмауери на рівні інстансів, що визначають, який трафік дозволено до серверів. Було створено окремі Security Groups для веб-сервера і для бази даних. Правила для веб-сервера допускають вхідні з’єднання HTTP (порт 80) та HTTPS (порт 443) від будь-яких адрес (тобто для публічного доступу до сайту), а також SSH (порт 22) – лише від мого статичного IP-адресу адміністратора. Будь-який інший вхідний трафік заблоковано за замовчуванням. Для MySQL-RDS група безпеки налаштована так, що дозволяє підключення на порт 3306 тільки з IP-адреси мого EC2-інстансу (тобто лише веб-додаток може звертатися до бази).

Завдяки Security Groups і VPC, мій веб-сервер захищений від несанкціонованого доступу на мережевому рівні: навіть якщо хтось дізнається його публічний IP, крім портів 80/443 жоден запит не пройде. VPC також надає можливість підключення VPN або AWS Direct Connect для безпечного доступу до хмарної мережі з офісу, але для мого дипломного проєкту достатньо було звичайного доступу через інтернет із захистом SSH-ключами та MFA. На рисунку 2.4 зображена логічна структура та архітектура компонентів Amazon VPC у хмарному середовищі AWS.

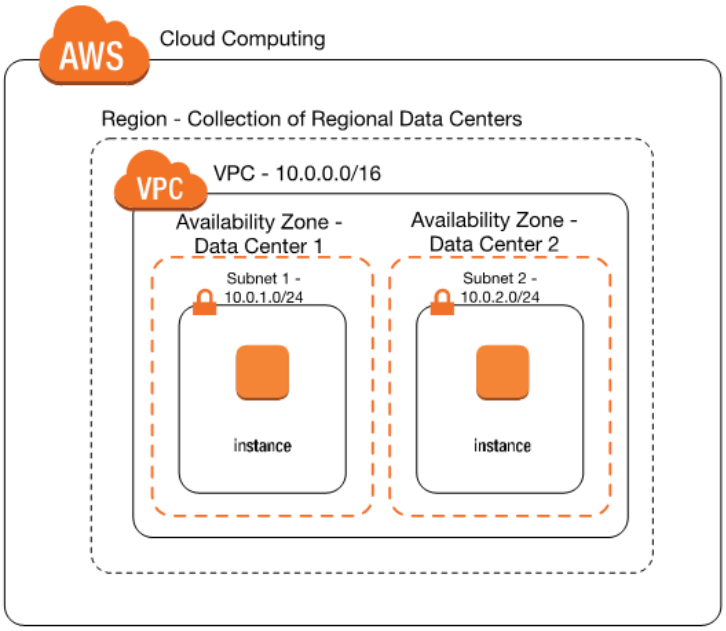


Рисунок 2.4 – Архітектура роботи віртуальних серверів всередині VPC [13]

Підсумовуючи, Amazon VPC виконує роль “приватного дата-центру” в AWS, де розташовано всі мої ресурси, і забезпечує їм мережеву ізоляцію та захист.

Amazon Route 53. Мій корпоративний веб-сайт має власне доменне ім’я, і для керування цим доменом використовується сервіс Route 53. Amazon Route 53 – це хмарний DNS-сервіс, який дозволяє реєструвати домени та керувати DNS-записами. У проєкті був налаштован запис типу A в Route 53, що вказує домен на публічну IP-адресу мого EC2-інстансу. Завдяки цьому

користувачі можуть звертатися до сайту за зручним ім’ям, а не за числовою адресою. Route 53 відомий своєю високою відмовостійкістю – служба розподілена по глобальній мережі DNS-серверів AWS і здатна обробляти мільйони запитів щосекунди, миттєво видаючи IP навіть при великих навантаженнях. Для мого невеликого сайту продуктивність DNS не є вузьким місцем, але надійність критично важлива: Route 53 гарантує, що домен буде правильно резольвитися в будь-який момент, оскільки працює у всіх регіонах AWS одночасно. Також була використана можливість перевірки стану (health check) в Route 53: сервіс може періодично надсилати HTTP-запити до мого сайту і відстежувати, чи він відповідає. Це дозволяє налаштувати сповіщення або автоматичне перемикання DNS на резервний сервер у разі недоступності основного. Хоча в архітектурі немає запасного сервера (одна зона відмови), було протестував функцію health check для ознайомлення. Route 53 також забезпечує гнучкі політики маршрутизації – крім простого відповідника (A-record) можна використати географічну маршрутизацію, балансування між кількома IP, пріоритети, що актуально для масштабних або глобальних застосувань. У підсумку, завдяки Route 53 мій сайт доступний під своїм доменом, а DNS-інфраструктура не потребує окремого адміністрування.

AWS IAM (Identity and Access Management). Безпека хмарного середовища значною мірою залежить від правильного управління користувачами та правами, тому активно застосується сервіс IAM. AWS IAM забезпечує централізоване керування доступом до всіх ресурсів AWS у моєму обліковому записі. В IAM було створено окремого користувача-адміністратора для, замість використання root-користувача AWS на щоденній основі. Цьому користувачу було призначено індивідуальну політику доступу, що дозволяє виконувати адміністративні дії для проєкту (створення інстансів, зміна налаштувань тощо), але були обмежені права лише необхідними – дотримуючись принципу найменших привілеїв. Root-користувач (головний обліковий запис AWS) тепер не використовується взагалі і захищений додатково. Крім того, для критичних операцій було

налаштовано використання IAM-ролей. Зокрема, моєму EC2-інстансу призначено роль, яка дає йому право записувати резервні копії в S3-бакет. Це означає, що сервер може виконувати операції в S3 без зберігання паролів чи ключів доступу – автентифікація відбувається прозоро через роль, прив'язану до інстансу. Такий підхід підвищує безпеку (не потрібно вшивати ключі AWS у код або конфігурації). Ще одним важливим кроком було ввімкнення Multi-Factor Authentication (MFA). Було підключено MFA для входу як root-користувача AWS, так і для мого адміністративного IAM-користувача. Тепер, при вході в консоль AWS, окрім пароля, запитується одноразовий код з мобільного додатку. Це суттєво знижує ризик компрометації облікового запису, навіть якщо пароль буде викрадено. AWS настійно рекомендує увімкнути MFA для облікового запису root та важливих користувачів, і в моєму проєкті ця рекомендація реалізована. Також були налаштувані політики паролів в IAM (мінімальна довжина, складність, ротація кожні 90 днів) для підвищення рівня безпеки облікових записів. Завдяки IAM адміністратор має чіткий контроль: хто і що може робити в хмарній інфраструктурі. Всі дії користувачів (чи ролей) при цьому фіксуються в журналі CloudTrail, про який йтиметься далі.

Amazon CloudWatch. Для моніторингу стану веб-сервера та логів застосунку використовується Amazon CloudWatch – це інтегрована система збору метрик і логування в AWS. CloudWatch автоматично отримує базові метрики з моїх ресурсів. Наприклад, запущений EC2-інстанс надсилає інформацію про завантаження CPU, обсяг мережевого трафіку та використання диску. За замовчуванням такі метрики збираються кожні 5 хвилин (достатньо для загального спостереження). На сервер був встановлен додатковий агент CloudWatch Agent, який дозволив отримувати детальні метрики – зокрема моніторити використання оперативної пам'яті, завантаженість файлової системи, процесорний час у розрізі процесів тощо, з інтервалом у 1 хвилину. CloudWatch також збирає логи: була налаштована інтеграцію Nginx з CloudWatch Logs, тому журнали веб-сервера (доступи та

помилки) передаються у хмару і зберігаються централізовано. Це зручно для аналізу – AWS надає інтерфейс пошуку по логах, побудови графіків по ним, а також може генерувати сигнали. Було створено кілька CloudWatch Alarm – наприклад, тривогу, якщо CPU utilization на EC2 понад 80% протягом 5 хвилин, або якщо сервер перестав відповідати на HTTP-запити (для цього Alarm підключено до згаданого health check Route 53). У разі спрацьовування тривоги CloudWatch може надіслати повідомлення електронною поштою або в інший канал (інтеграція з Amazon SNS). Таким чином, навіть без постійного ручного нагляду, адміністратор буде оперативно сповіщений про потенційні проблеми – високе навантаження, відсутність доступності сайту тощо. CloudWatch дозволяє будувати персоналізовані дашборди: було створено панель моніторингу, де відображаються графіки завантаження CPU, пам’яті, диску, мережевого трафіку веб-сервера, а також кількість запитів до сайту на хвилину (на основі логів Nginx). Така централізована картина значно полегшує підтримку – адміністратор швидко бачe аномалії або тенденції (наприклад, зростання трафіку з часом). Крім того, зберігання логів у CloudWatch Logs корисне для аудиту безпеки – журнали доступу можна автоматично аналізувати на предмет підозрілих активностей (наприклад, багаторазові спроби входу). В цілому, AWS CloudWatch відіграє роль “оком” інфраструктури, надаючи оперативну інформацію про її стан і інструменти реагування.

AWS CloudTrail. Щоб мати повний контроль над тим, хто і коли вчиняв будь-які дії в моєму хмарному середовищі, було задіяно сервіс CloudTrail. AWS CloudTrail - це система журналювання та аудиту дій в обліковому записі AWS. Вона автоматично записує кожен запит до AWS API або консолі: створення або видалення EC2-інстансу, зміна налаштувань безпеки, додавання користувача IAM, спроби входу в консоль - усі ці події заносяться до немодифікованого журналу.

Окрім базового фіксування подій, використання CloudTrail дозволяє проводити детальний аналіз безпеки та забезпечувати відповідність

нормативним вимогам. Це стає можливим завдяки автоматизації збору логів, які зберігаються у захищеному сховищі S3. Таким чином, адміністратор отримує потужний інструмент для швидкого розслідування інцидентів, виявлення підозрілої активності та оперативного усунення помилок у конфігурації інфраструктури, що значно підвищує загальний рівень кібербезпеки системи. На рисунку 2.5 зображен інтерфейс стандартної інформаційної панелі моніторингу ресурсів у сервісі Amazon CloudWatch.

Рисунок 2.5 – Інтерфейс CloudWatch [14]

У моєму проєкті CloudTrail дозволяє відстежити, хто виконував адміністративні операції. Наприклад, якщо адміністратор змінить правило Security Group або видалю якийсь ресурс, CloudTrail зафіксує його IAM-користувача, час операції, з якої IP-адреси це зроблено та чи була дія успішною. Ці логи дуже цінні для розслідування інцидентів безпеки або аналізу помилок конфігурації – маючи історію подій, можливо з'ясувати, що саме пішло не так і в який момент. CloudTrail за замовчуванням зберігає історію останніх 90 днів, доступну для перегляду через веб-консоль AWS. Було налаштовано Trail – постійний потік логів CloudTrail – щоб вони автоматично зберігалися в мій S3-бакет для архіву. Логи зберігаються у вигляді JSON-файлів, згрупованих по годинах, і можуть довго утримуватися в S3 з мінімальними витратами. Додатково, у нас є можливість підключити

сервіс аналізу (Amazon Athena чи AWS CloudTrail Lake) для виконання запитів по цим логам – наприклад, знайти всі дії, пов'язані з певним ресурсом або всі входи з неавторизованих IP. У нашому випадку, періодично переглядається журнал CloudTrail на предмет підозрілих дій (AWS також може генерувати Alert у разі виявлення аномальних шаблонів, якщо підключити AWS GuardDuty – це поза межами даного проєкту, але варто згадати для повноти). Загалом, CloudTrail забезпечує прозорість і підзвітність всіх операцій в хмарі. Це критично для корпоративного середовища, оскільки дозволяє дотримуватися внутрішніх політик безпеки та вимог відповідності (compliance) – у разі аудиту можна показати журнал всіх змін інфраструктури. У моєму дипломному проєкті CloudTrail виступає важливим компонентом стратегій безпеки та резервного копіювання змін.

AWS Certificate Manager (ACM). Щоб забезпечити шифрування трафіку до мого веб-сайту (HTTPS-з'єднання), необхідно видати SSL/TLS сертифікат для домену і налаштувати його на веб-сервері. AWS пропонує для цього сервіс Certificate Manager, який значно спрощує роботу з сертифікатами. AWS Certificate Manager дозволяє безкоштовно отримувати публічні SSL-сертифікати для підтверджених доменів і автоматично подовжує їх перед закінченням терміну дії. У консолі ACM є можливість подати запит на сертифікат для свого домену (наприклад, corp.example.com), підтвердивши володіння доменом через DNS-запис. Після випуску сертифікат зберігається в AWS і може бути прив'язаний до таких сервісів, як Elastic Load Balancer, CloudFront або API Gateway – вони автоматично розгорнуть сертифікат і увімкнуть HTTPS. У випадку мого проєкту веб-сайт працює напряму на EC2 без балансувальника, тому використання ACM обмежене: ACM не вміє "встановлювати" сертифікати всередині самих EC2-інстансів (це залишається на відповідальності користувача). Щоб не налаштовувати вручну випуск і продовження сертифіката, як альтернативу ACM був застосован безкоштовний сторонній сервіс Let's Encrypt. Була встановлена на сервер утиліта Certbot, яка автоматично отримала дійсний SSL-сертифікат

для мого домену від Let’s Encrypt і налаштувала його в конфігурації Nginx. Certbot також налаштований на автоматичне продовження сертифіката кожні 90 днів, тож підтримка HTTPS не потребує ручного втручання. Таким чином, забезпечується шифрування всього трафіку між клієнтами і сервером, що є обов’язковим вимогою для корпоративного веб-сервера (захист даних і облікових записів користувачів). В розрізі AWS цей аспект інтегрується з іншими сервісами: Route 53 використовувався для перевірки домену при випуску сертифіката, логи доступу HTTPS передаються в CloudWatch, а сам сертифікат та ключі зберігаються на сервері в захищеному вигляді. Питання криптографії і деталі налаштування HTTPS розкриті у наступному розділі, присвяченому безпеці, але на рівні огляду інструментів можна відзначити, що AWS Certificate Manager є зручним рішенням для керування сертифікатами при використанні служб AWS.

3 ЗАСТОСУВАННЯ КРИПТОГРАФІЧНИХ МЕТОДІВ У ХМАРНІЙ ІНФРАСТРУКТУРІ AWS

У сучасній хмарній інфраструктурі криптографічні методи відіграють ключову роль у захисті даних і ресурсів. У межах дипломного проєкту з розгортання корпоративного веб-сервера на AWS передбачено комплекс заходів безпеки, які використовують різні засоби криптографії для забезпечення конфіденційності, цілісності та автентичності даних. Нижче розглянуто основні інструменти криптографічного захисту, застосовані в проєкті: (1) SSL/TLS (HTTPS) шифрування з сертифікатом Let's Encrypt, (2) шифрування дисків EC2 (EBS encryption), (3) шифрування резервних копій у S3 на боці сервера (SSE-S3 або SSE-KMS), (4) сервіс AWS KMS для керування ключами шифрування, та (5) багатофакторна автентифікація (MFA) для доступу до консолі адміністрування. Кожен із цих засобів забезпечує певний аспект криптографічного захисту інфраструктури; розглянемо принцип їх роботи та реалізацію в рамках проєкту, наводячи практичні приклади налаштування.

3.1 SSL/TLS та HTTPS (сертифікат Let's Encrypt)

Протокол TLS/SSL. Одним із фундаментальних криптографічних інструментів для захисту веб-сервера є протокол TLS (Transport Layer Security), що лежить в основі HTTPS. TLS – це криптографічний протокол, призначений для безпечної передачі даних в Інтернеті; він використовує комбінацію асиметричного шифрування та цифрових сертифікатів X.509 для автентифікації і встановлення захищеного каналу. Під час TLS-рукостискання (handshake) клієнт і сервер погоджують версію протоколу і

криптографічні алгоритми, після чого за допомогою асиметричних алгоритмів (з використанням відкритого та приватного ключів) обмінюються даними для встановлення спільного таємного ключа сеансу. Цей спільний симетричний ключ далі використовується для швидкого шифрування всього трафіку між клієнтом і сервером. Таким чином, TLS забезпечує автентифікацію сервера (підтвердження його особи) та конфіденційність і цілісність переданих даних: сторонній спостерігач не може прочитати або змінити трафік HTTPS завдяки сильним криптографічним механізмам.

Окрім захисту від перехоплення даних (sniffing), протокол TLS ефективно протидіє атакам типу «людина посередині» (Man-in-the-Middle), оскільки будь-яка спроба модифікації пакетів призведе до порушення перевірки контрольних сум і розриву з'єднання. Сучасні версії протоколу, зокрема TLS 1.3, також фокусуються на мінімізації затримок при встановленні зв'язку, що дозволяє поєднувати високий рівень безпеки із максимальною продуктивністю веб-ресурсів. Впровадження даного протоколу є обов'язковим стандартом для будь-якої хмарної інфраструктури, що працює з персональними даними користувачів. На рисунку 3.1 зображен інтерфейс налаштування сертифікатів X.509 та параметрів Single Sign-On (SSO) у консолі адміністратора.

Рисунок 3.1 – фрагмент X.509-сертифіката у форматі PEM [19]

HTTPS і сертифікація Let's Encrypt. Для впровадження TLS у веб-застосунку потрібен дійсний цифровий сертифікат. У проєкті використано сертифікат, отриманий через службу Let's Encrypt – це відкритий некомерційний центр сертифікації (Certificate Authority), що надає безкоштовні SSL/TLS-сертифікати й автоматизує процес їх випуску. Let's Encrypt спрощує перехід на HTTPS: сервіс видає довірені браузерами сертифікати після підтвердження контролю власника над доменом (наприклад, шляхом розміщення спеціального файлу на веб-сервері або через DNS-запис). У нашому випадку для доменного імені корпоративного сайту було налаштовано автоматизоване отримання та встановлення сертифіката Let's Encrypt (наприклад, за допомогою утиліти Certbot). Веб-сервер (Apache/Nginx) налаштований на використання цього сертифіката, що дозволило увімкнути захищений протокол HTTPS. Практичним наслідком є те, що всі взаємодії користувачів з сайтом (включно з передачою облікових даних при вході, конфіденційної інформації тощо) відбуваються по шифрованому каналу. Користувачі бачать у браузері значок замка, що вказує на дійсний сертифікат і шифрування з’єднання. Таким чином, SSL/TLS з Let's Encrypt забезпечує шифрування трафіку в транзиті і гарантує автентичність сервера, захищаючи систему від «прослуховування» мережі та атак типу “man-in-the-middle”.

3.2 Шифрування дисків EC2 (Amazon EBS Encryption)

Шифрування томів EBS. Для захисту даних, що зберігаються на віртуальних дисках серверів EC2, використовується шифрування Amazon EBS (Elastic Block Store). EBS надає можливість шифрувати як завантажувальні, так і додаткові томи EC2 інстансу, тобто виконувати прозоре шифрування даних на диску (at-rest encryption). При створенні

шифрованого тому EBS автоматично шифрує всі дані, що записуються на диск, з використанням високонадійного алгоритму AES-256. Зокрема, застосовується режим AES-256-XTS, який оперує двома 256-бітними ключами (еквівалентно 512-бітному ключу) для підвищення стійкості шифрування на рівні блочного пристрою. Ключ шифрування тома (data key) генерується сервісом AWS KMS і зберігається разом із томом у зашифрованому вигляді: перед збереженням цей ключ шифрується майстер-ключем, що знаходиться в KMS. За замовчуванням AWS автоматично створює в кожному регіоні спеціальний керований ключ із псевдонімом aws/ebs для потреб EBS, і він використовується для шифрування томів за замовчуванням. За бажанням замість нього можна використати власний клієнтський ключ KMS – це дає більше контролю (можливість налаштування політик доступу до ключа, його ротації, відключення тощо). На рисунку 3.2 зображена ієрархія ключів шифрування та структура кореневого ключа в сервісі AWS KMS

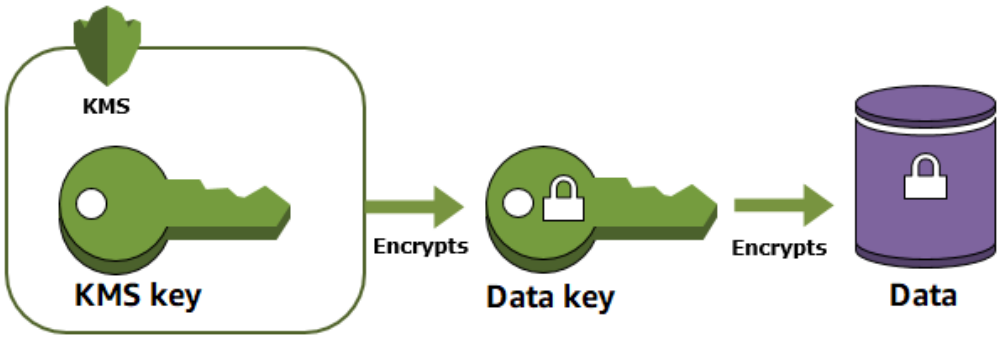


Рисунок 3.2 – принцип роботи AWS KMS (Key Management Service) [20]

Реалізація та ефект шифрування EBS в проєкті. В рамках дипломного проєкту для EC2-інстансу, на якому розгорнуто веб-сервер, було увімкнуто шифрування диска. Це означає, що весь вміст файлової системи на цьому сервері шифрується прозоро для користувача: операційна система і прикладні програми працюють із даними у звичайний спосіб, але на фізичних

носіях в дата-центрі AWS ці дані зберігаються в зашифрованому вигляді. Більш того, шифрування EBS покриває не лише сам том, а й автоматично шифрує всі резервні знімки (snapshots), зроблені від цього тому, та будь-які нові томи, створені з таких знімків. Всі операції вводу-виводу між EC2 і шифрованим томом також шифруються «на льоту» апаратно (через механізми AWS Nitro), тож дані захищені і під час передачі всередині хмарної інфраструктури. Практичний приклад: якщо зловмисник навіть отримає несанкціонований доступ до фізичного диска або знімку EBS (що зберігається у внутрішньому сховищі S3), він не зможе прочитати інформацію без доступу до ключів дешифрування. Для адміністратора AWS ввімкнення EBS Encryption зводиться до вибору опції "Encrypt Volume" при створенні тома (або ввімкнення "Encryption by Default" для облікового запису в певному регіоні). У нашому проєкті ця опція була активована для всіх томів, що містять важливі дані, що гарантує захист даних у разі компрометації інфраструктури зберігання. Важливо відзначити, що шифрування EBS не впливає суттєво на продуктивність завдяки апаратному прискоренню і реалізується повністю на стороні AWS, залишаючись невидимим для кінцевого користувача.

3.3 Шифрування резервних копій у Amazon S3 (SSE-S3 та SSE-KMS)

Server-Side Encryption в S3. Для довгострокового зберігання резервних копій (бекапів) у проєкті використовується служба Amazon S3. Щоб гарантувати безпеку даних, що зберігаються в S3-бакеті, увімкнено шифрування на боці сервера (Server-Side Encryption, SSE). При серверному шифруванні сервіс S3 самостійно шифрує дані під час запису на носії в дата-центрі і автоматично дешифрує при читанні, прозоро для користувача. Якщо

користувач має необхідні права доступу, то процес роботи з зашифрованими об’єктами нічим не відрізняється від звичайних: наприклад, завантаження (GET) зашифрованого файлу через S3 відбувається так само, як і незашифрованого, оскільки S3 повертає вже дешифровані дані за умови успішної автентифікації запиту. Таким чином, увімкнене шифрування S3 забезпечує захист даних на спочинку (at rest) – тобто у сховищі, не впливаючи на зручність доступу для авторизованих користувачів.

Варіанти SSE: SSE-S3 та SSE-KMS. Amazon S3 підтримує кілька механізмів шифрування на боці сервера; у контексті нашого проєкту розглядаються два з них: SSE-S3 та SSE-KMS.

SSE-S3 (з використанням ключів, керованих S3) – це стандартний і тепер уже типовий режим шифрування в S3. Наразі AWS за замовчуванням увімкнув SSE-S3 для всіх нових об’єктів, що завантажуються в S3 (починаючи з 5 січня 2023). При SSE-S3 кожен об’єкт шифрується унікальним випадковим ключем, який генерується і управляється самим сервісом S3. Цей унікальний ключ об’єкта, в свою чергу, додатково шифрується іншим ключем - так званим кореневим ключем (root key), який зберігається в AWS і регулярно ротується для підвищення безпеки. Для шифрування даних використовується один із найсильніших доступних алгоритмів – 256-бітовий блочний шифр AES-256. Важливо, що всі ці криптографічні операції повністю виконуються на стороні AWS: тобто, коли, на приклад, завантажується, файл резервної копії в бакет S3 з увімкненим SSE-S3, він автоматично шифрується без додаткових зусиль або знання ключів зі сторони розробника. Відповідно, при завантаженні цього файлу назад отримуються вже розшифровані дані (якщо маємо права доступу до об’єкта).

SSE-KMS (з використанням ключів AWS KMS) – це режим шифрування, де S3 інтегрується з сервісом AWS Key Management Service для керування ключами. Принципово механізм схожий на SSE-S3 (кожен об’єкт шифрується окремим даним ключем AES-256), проте відмінність у тому, що

майстер-ключ для шифрування об’єктів зберігається не у внутрішньому сервісі S3, а у KMS. За SSE-KMS користувач отримує більший контроль над ключами: можна задавати власні customer-managed ключі, налаштовувати для них політики доступу, відслідковувати використання ключа через AWS CloudTrail, виконувати ротацію ключового матеріалу тощо. Фактично, при завантаженні об’єкта з SSE-KMS S3 запитує у KMS новий ключ даних для шифрування, або використовує вже існуючий KMS-ключ, зашифровує ним дані та зберігає шифротекст у бакеті; метадані об’єкта містять ідентифікатор KMS-ключа, який потрібен при читанні для дешифрування. Будь-який запит на отримання об’єкта з SSE-KMS вимагає не лише прав доступу до самого об’єкта в S3, але й дозволу на використання відповідного KMS-ключа для дешифрування. Отже, SSE-KMS додає другий шар контролю доступу і аудиту: навіть користувач із правами на читання з бакету не зможе розшифрувати файл, якщо йому не дозволено операцію Decrypt відповідним ключем KMS.

Застосування в проєкті. Резервні копії критичних даних (наприклад, дампи баз даних чи конфігурацій) у моєму проєкті зберігаються в приватному бакеті S3 з увімкненим шифруванням. Було прийнято рішення використати режим SSE-KMS для більш гнучкого керування ключами. Для цього створено окремий ключ AWS KMS, призначений для шифрування бекапів, і встановлено політику за замовчуванням (Default Encryption) для бакету: всі нові об’єкти шифруються цим KMS-ключем. Практичний приклад налаштування: при створенні бакету через консоль AWS робиться перехід до розділу Properties → Default encryption і обрали «AWS KMS–Managed Key», вказавши спеціально створений ключ (або можна було використати стандартний ключ aws/s3 від AWS). Після цього будь-яке завантаження файлу в бакет (як вручну через консоль, так і автоматично з застосунку або сервісу резервного копіювання) отримує прапорець «зашифровано» і при зберіганні дані шифруються. Уявімо ситуацію: навіть якщо б хтось отримав несанкціонований доступ до самих фізичних носіїв або зробив копію об’єктів

з бакету, ці файли залишаться зашифрованими набором байтів. Тільки виклики через AWS API з належними правами, які включають доступ до потрібного KMS-ключа, дозволять отримати їх у відкритому вигляді. Зауважимо, що з січня 2023 року навіть без вручну заданої конфігурації шифрування всі об’єкти S3 шифруються при записі за замовчуванням режимом SSE-S3. Однак використання SSE-KMS у проєкті обґрунтоване вимогами до контролю та аудиту: це можна побачити у логах CloudTrail кожен випадок використання ключа (наприклад, розшифрування при читанні резервної копії) і, за потреби, відкликати доступ або відключити ключ, щоб зробити дані недоступними. Цей підхід підвищує рівень безпеки зберігання резервних копій у хмарі до корпоративних стандартів. Крім того, для забезпечення цілісності архітектури було вроваджено механізм S3 Versioning. Це дозволяє захистити резервні копії від випадкового видалення або атаки програмами-вимагачами (ransomware), зберігаючи попередні версії об'єктів. У поєднанні з S3 Object Lock реалізовано модель WORM (Write Once, Read Many), що гарантує незмінність критичних бекапів протягом визначеного періоду часу відповідно до політик безпеки компанії.

Важливим аспектом є також автоматизація життєвого циклу даних через S3 Lifecycle Policies. Оскільки актуальність дампів баз даних з часом знижується, було налаштовано автоматичне переміщення об'єктів до архівних класів зберігання, таких як S3 Glacier Flexible Retrieval, через 30 днів після їх створення. Це дозволяє значно оптимізувати витрати на хмарну інфраструктуру, не жертвуючи при цьому доступністю архівних копій для аудиту.

Для оперативного реагування на інциденти налаштовано сповіщення через Amazon SNS. У разі будь-якої невдалої спроби доступу до KMS-ключа або видалення об'єкта з бакету, команда безпеки миттєво отримує алерт. Таким чином, обрана стратегія використання SSE-KMS разом із додатковими інструментами AWS створює багаторівневу систему захисту, яка відповідає найсуворішим вимогам до збереження корпоративних даних та забезпечує

швидке відновлення бізнес-процесів у разі потреби. Окрему увагу в проєкті приділено стратегії Cross-Region Replication (CRR). Для захисту від регіональних збоїв у роботі хмарної інфраструктури було налаштоване автоматичне дублювання критичних бекапів у інший географічний регіон AWS. При цьому важливо, що копіювання відбувається із перешифруванням даних «на льоту» за допомогою KMS-ключа цільового регіону, що забезпечує безперервність захисту на кожному етапі транзиту.

3.4 AWS KMS – сервіс керування криптографічними ключами

Огляд AWS KMS. AWS Key Management Service (KMS) – це централізований хмарний сервіс для створення, зберігання і керування криптографічними ключами, які використовуються для шифрування даних або цифрового підпису. KMS спрощує управління ключами шифрування: адміністратор може генерувати ключі (симетричні або асиметричні), налаштовувати політики доступу до них і викликати криптографічні операції (шифрування, дешифрування, генерування підпису тощо) через стандартизований інтерфейс AWS API. Безпека реалізації KMS гарантується тим, що власне матеріал ключів верхнього рівня (master keys) зберігається і використовується у спеціальних апаратних модулях безпеки (HSM), сертифікованих за високим стандартом FIPS 140-3 (рівень 3). Ці кореневі ключі ніколи не покидають межі сервісу KMS у відкритому вигляді – будь-яке використання ключа (наприклад, для дешифрування) виконується всередині HSM, а ззовні можна отримати лише результат операції. Таким чином, навіть внутрішнім компонентам AWS недоступні нешифровані значення ключів. KMS побудований на принципі ієрархії ключів (envelope encryption): замість шифрування великих обсягів даних безпосередньо майстер-ключем, сервіс генерує окремі ключі даних (data keys) для кожного

набору даних чи ресурсу. Ці ключі даних шифруються майстер-ключем і можуть передаватися іншим сервісам AWS для локального використання. Подібний підхід вже зазначався вище: при шифруванні тома EBS чи об’єкта S3 задіяно майстер-ключ KMS, який шифрує унікальний ключ даних, а вже ним виконано шифрування великого масиву інформації.

3.5 Багатофакторна автентифікація (MFA) для доступу до AWS

Концепція MFA. Традиційний доступ до облікового запису або системи захищений лише паролем, проте такий захист вразливий, якщо пароль буде розгадано або викрадено. Багатофакторна автентифікація (Multi-Factor Authentication, MFA) значно підвищує рівень безпеки, вимагаючи при вході додаткового перевірочного фактора окрім пароля. AWS визначає MFA як кращу практику для захисту облікових записів: при ввімкненій MFA користувач під час входу в консоль AWS повинен ввести не лише свій пароль (фактор знання – something you know), але й одноразовий код з свого MFA-пристрою (фактор володіння – something you have). У деяких випадках другим фактором може бути біометрія (наприклад, відбиток пальця чи розпізнавання обличчя на пристрої) – це фактор властивості користувача (something you are). Лише поєднання правильного пароля і дійсного коду з другого фактору дозволяє успішно автентифікуватися; таким чином, компрометація лише пароля не дасть змоги зловмиснику отримати доступ до облікового запису. З огляду на високу відповідальність облікового запису AWS (особливо root user або адміністраторського IAM-користувача), MFA є критично важливим засобом криптографічного захисту на рівні автентифікації.

Використання MFA дозволяє нівелювати ризики, пов'язані з методами соціальної інженерії, брутфорс-атаками та використанням викрадених

облікових даних. Окрім апаратних токенів та віртуальних пристроїв MFA, сучасні хмарні платформи підтримують інтеграцію зі стандартами FIDO2/WebAuthn, що забезпечує ще вищий рівень захисту від фішингу. Впровадження обов'язкової багатофакторної автентифікації для всіх привілейованих користувачів є фундаментальним кроком у побудові стратегії нульової довіри (Zero Trust Architecture), де кожна спроба доступу підлягає ретельній та багатовекторній перевірці. На рисунку 3.3 зображена структура та ключові компоненти сервісу AWS CloudTrail для моніторингу активності.

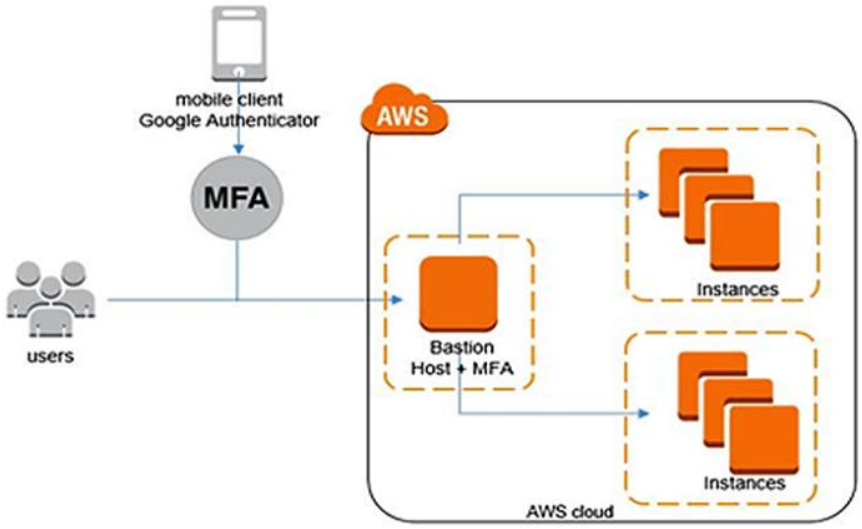


Рисунок 3.3 – Архітектура роботи MFA [21]

Реалізація MFA в AWS. Amazon AWS підтримує різні типи MFA-пристроїв. Найбільш розповсюджений та зручний підхід – це використання віртуального автентикатора (наприклад, мобільного додатку на смартфоні: Google Authenticator, AWS MFA, Authy тощо), який генерує одноразові коди за стандартом TOTP (Time-based One-Time Password). TOTP-пристрій має спільний секретний ключ, зареєстрований при активації MFA (представлений у вигляді QR-коду, який сканує користувач). Після прив'язки такого пристрою обліковий запис AWS очікує, що при кожному вході буде надано

6-значний код, згенерований додатком. Код дійсний лише протягом короткого проміжку часу (30 секунд), після чого генерується новий, що унеможливлює його багаторазове використання чи підбір. З криптографічної точки значення TOTP-алгоритм базується на хешуванні поточного часу і секретного ключа; перевірка на стороні AWS відбувається шляхом обчислення очікуваного коду з того ж секрету і часу та порівняння з введеним користувачем. Окрім програмних токенів, AWS також підтримує апаратні MFA-токени – це можуть бути класичні ключі, що генерують коди (наприклад, брелоки від третьої сторони, сертифіковані AWS), або сучасні FIDO2/U2F-сумісні апаратні ключі безпеки (наприклад, YubiKey). FIDO-пристрої реалізують багатофакторну автентифікацію на основі відкритого ключа (асиметричної криптографії) – при прив'язці ключа до облікового запису генерується пара ключів, причому приватний ключ зберігається на самому пристрої і ніколи не покидає його, а для входу потрібне криптографічне підтвердження операції (наприклад, натискання кнопки на USB-ключі) замість введення коду. Такий метод є стійким до фішингу, адже підписує саме запит автентифікації на справжньому сайті AWS.

MFA в дипломному проєкті. У контексті розгортання корпоративної інфраструктури на AWS було впроваджено MFA для облікових записів адміністраторів (передусім для облікового запису root user AWS). При першому налаштуванні облікового запису власника (root) в консолі безпеки AWS було додано віртуальний MFA-пристрій: використано додаток Google Authenticator на смартфоні відповідального адміністратора. Після прив'язки, кожен вхід до AWS Management Console під цим обліковим записом вимагає вводу коду MFA. Наприклад, адміністратор відкриває консоль AWS, вводить свій e-mail та пароль, а система додатково запитує шестизначний код з Google Authenticator; лише після введення правильного коду вхід завершується успішно. Уявімо загрозу: навіть якщо пароль від консолі випадково став відомий зловмиснику, без фізичного доступу до смартфона з додатком MFA він не зможе пройти другий фактор автентифікації. Таким чином, ризик

несанкціонованого доступу значно знижується. AWS дозволяє реєструвати кілька MFA-пристроїв на одного користувача (до 8 на кожного IAM-користувача або root), що корисно для відмовостійкості – було додано дві різні програми на різних пристроях для одного з облікових записів, аби у разі втрати телефону був резервний метод входу. Також в політиках доступу AWS можна налаштувати обов'язковість MFA: наприклад, для дій, пов'язаних з критичною конфігурацією, встановлено додаткову умову (Condition), яка вимагає, щоб сесія була автентифікована з MFA. Це гарантує, що навіть компрометація токенів доступу (Access Key) не дозволить виконати чутливі дії без MFA.

На завершення, усі вищезазначені заходи криптографічного захисту працюють комплексно, забезпечуючи різні рівні безпеки в нашому AWS-рішенні. Шифрування TLS (HTTPS) захищає дані під час передавання між користувачами і сервером, шифрування EBS і S3 гарантує збереження даних у хмарі у закритому вигляді, керування ключами через AWS KMS дає централізований контроль над усіма криптографічними матеріалами, а багатофакторна автентифікація запобігає несанкціонованому доступу до самих хмарних ресурсів. Такий багатошаровий підхід відповідає найкращим практикам інформаційної безпеки і демонструє прикладове використання криптографії для побудови захищеної корпоративної інфраструктури на AWS.

Таким чином, у сучасному світі хмарні технології відіграють критично важливу роль у цифровій трансформації бізнесу, освіти, державного управління та повсякденного життя. Завдяки моделі "інфраструктура як послуга" (IaaS) компанії можуть розгортати сервери, бази даних, сховища та мережі без витрат на фізичне обладнання. Це дозволяє запускати продукти швидко, масштабувати навантаження у разі потреби та скорочувати витрати.

В межах дипломного проєкту були використані інструменти AWS, щоб створити гнучку та безпечну серверну інфраструктуру корпоративного рівня. Реалізовано EC2 для обчислень, RDS для бази даних, S3 і EBS для зберігання

та резервного копіювання, а також VPC для мережевої ізоляції. Важливою частиною проєкту стали заходи безпеки: було впроваджено багатофакторну автентифікацію, шифрування даних у сховищах, шифрування трафіку через HTTPS і централізоване управління ключами через KMS.

Таким чином, хмарні сервіси стали основою ефективного, масштабованого та безпечного рішення, що відповідає сучасним вимогам до цифрової інфраструктури.

4 НАЛАШТУВАННЯ КОРПОРАТИВНОГО СЕРВЕРА З КОМПЛЕКСОМ ЗАСОБІВ КІБЕРБЕЗПЕКИ

4.1 Мережева інфраструктура (VPC)

Мережа (VPC): Це фундамент. Створюється ізольована мережа, для того щоб сервер не «гуляв» по Інтернету. Це дуже важливий етап для створення гідного рівня безпеки, а також дозволяє розділити архітектуру на публічну і приватну. В AWS зараз є зручний майстер, який створює все одразу (і VPC, і підмережі, і шлюзи).

Крок 1. Ізоляція від небажаних «гостей» Перехід до VPC Dashboard та натискання кнопки Create VPC. Відбувається потрапляння в VPC settings.

Крок 2. Побудова надійних «стін»: Обирається опція VPC and more (Ця опція малює карту мережі і створює підмережі автоматично).

Name tag auto-generation: Обирається назва проєкту (у даному випадку - SHK-Diploma). IPv4 CIDR block: Залишається 10.0.0.0/16 (це стандартний приватний діапазон).

Availability Zones (Зони доступності): Обирається 1 (Робиться Single-AZ deployment для економії ресурсів, хоча архітектура дозволяє розширення, у разі необхідності у майбутньому).

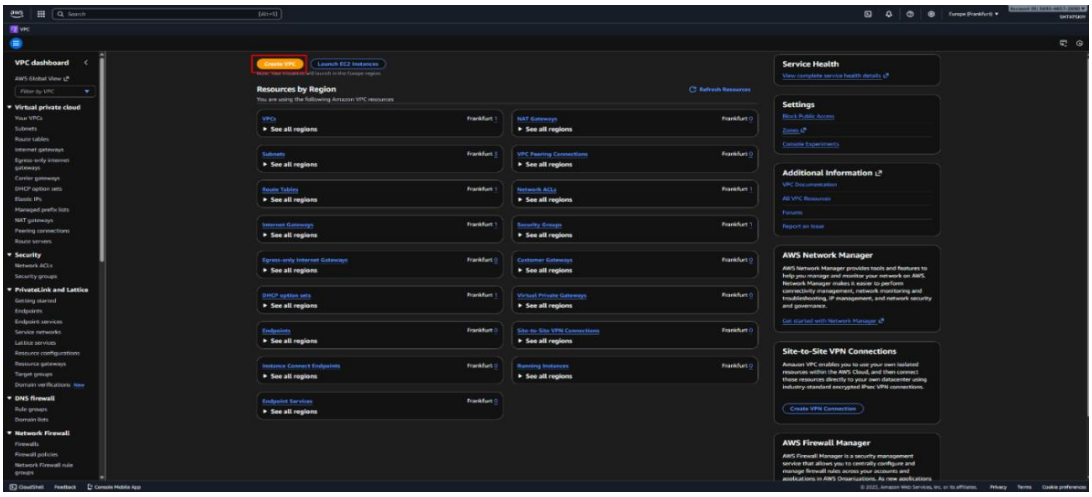


Рисунок 4.1 – Dashboard VPC

Number of public subnets (Публічні підмережі):

Обирається 1. (Тут буде розміщено веб-сервер, щоб його бачили з інтернету).

Number of private subnets (Приватні підмережі):

Обирається 1. (Тут буде розміщено база даних, схована від інтернету).

NAT gateways:

Обирається None. (Задля поєднання балансу ціни та якості вимикається NAT через її відносно високу ціну ~$30/міс. Для диплома оновлення серверів відбуватиметься через тимчасові методи або Public IP, щоб не витрачати кошти).

VPC endpoints:

Обирається None. Це платні "тунелі" до S3. На даному етапі вони не потрібні, веб-сервер дістанеться до S3 через звичайний інтернет.

DNS (Імена серверів):

Enable DNS hostnames - Вмикається

Enable DNS resolution - Вмикається

Це дозволяє серверам мати зрозумілі імена, а не тільки сухі цифри IP-адрес.

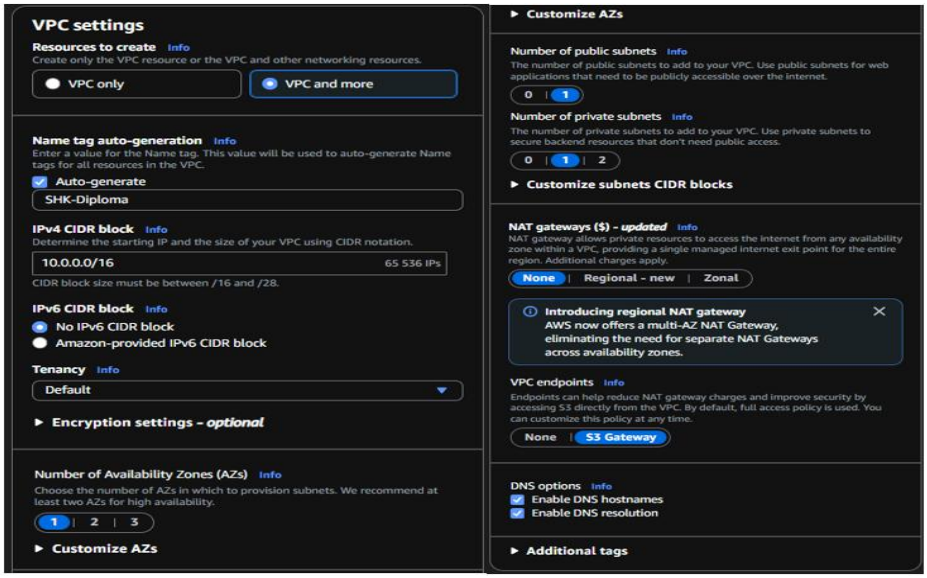


Рисунок 4.2 – Конфігурація параметрів VPC: IPv4 CIDR та налаштування DNS.

Натискається кнопка Create VPC, після чого очікується, поки всі відмітки стануть зеленими.

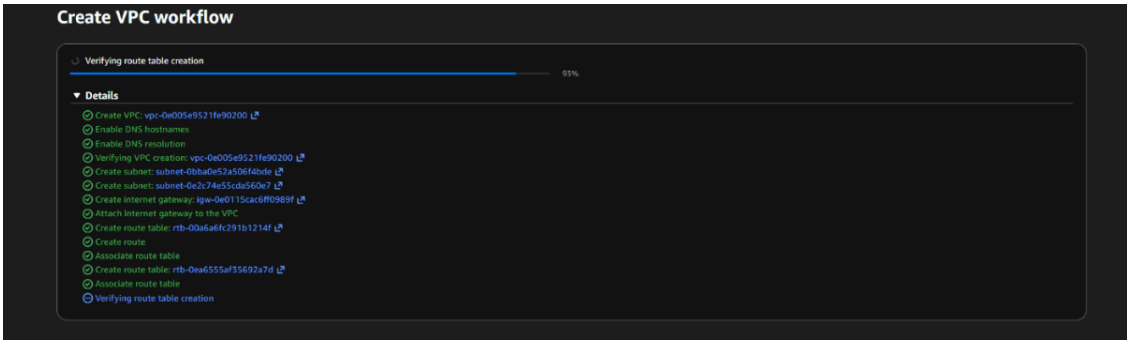


Рисунок 4.3 – Процес створення VPC

Що було зроблено: Побудовано "будівлю офісу" (VPC 10.0.0.0/16), створено "хол для гостей" (Public Subnet) і "кабінет директора" (Private Subnet), а також підведено інтернет-кабель до холу (Internet Gateway).

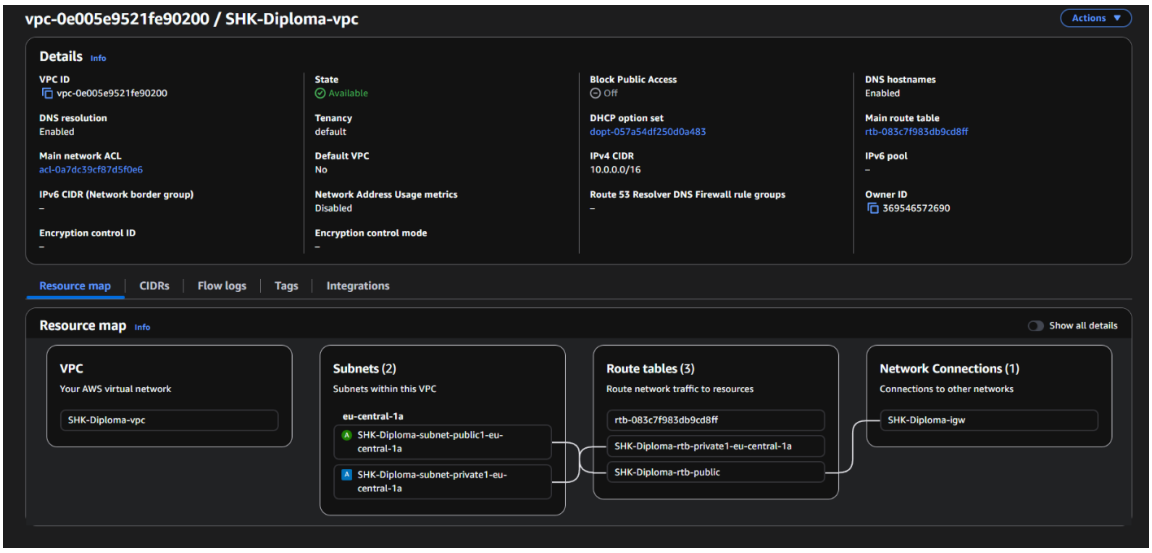


Рисунок 4.4 – Топологія створеної мережі VPC (Resource Map).

4.2 Запуск сервера (EC2)

Amazon EC2 (Elastic Compute Cloud) - це веб-сервіс, який надає обчислювальні потужності в хмарі. Простими словами, це віртуальний сервер, який орендується у Amazon замість того, щоб купувати власне "залізо".

Це класичний приклад моделі IaaS (Infrastructure as a Service). Отримується "чистий" сервер, де самостійно обирається операційна система та налаштовується все необхідне програмне забезпечення для власних потреб та виходячи з бюджету.

Крок 1. Вхід у "майстерню" серверів.

Перехід до EC2 Dashboard (панель керування) та ініціалізація створення серверу натисканням кнопки Launch instance. Відбувається перехід до етапу налаштувань.

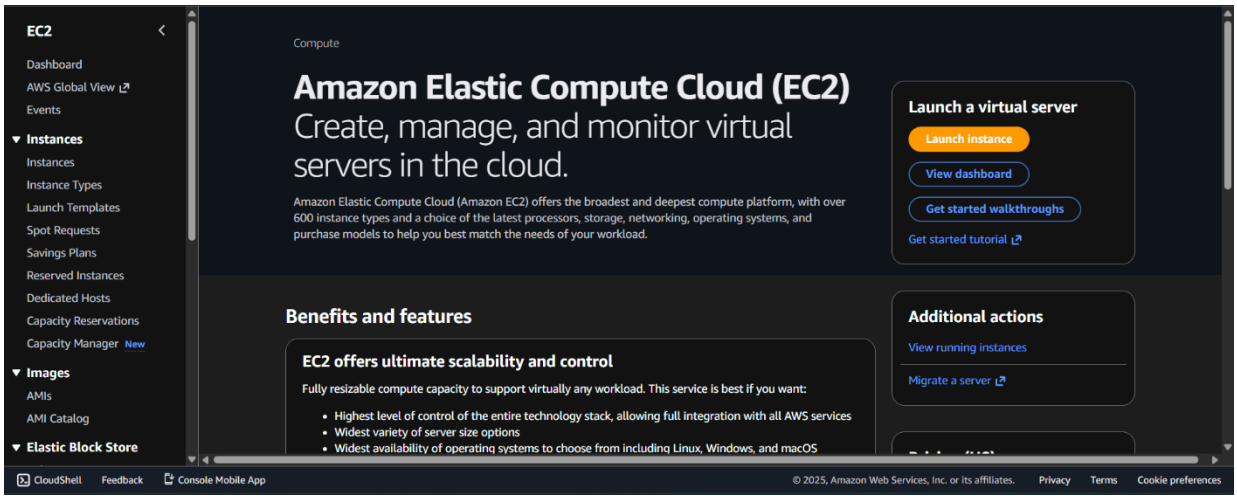


Рисунок 4.5 – Панель керування EC2

Крок 2. Ім'я сервера (OS):

Name: Обирається назва (у даному випадку - Corporate-Web-Server)

Application and OS Images (AMI): Обирається іконка Ubuntu.

Amazon Machine Image (AMI): У випадаючому списку обирається Ubuntu Server 24.04 LTS. Це найсвіжіша версія (LTS означає Long Term Support - довгострокова підтримка). Вона безкоштовна для використання

Крок 3. "М'язи" сервера (Instance Type):

Instance type: Обирається t3.micro.

t2.micro дає 2 vCPU (віртуальне ядро процесора) і 1 GiB оперативної пам'яті.

Крок 4. Ключі від дверей (Key pair):

Це найважливіший момент для доступу.

Знаходиться поле Key pair (login).

Натискається посилання праворуч Create new key pair.

У вікні налаштувань:

Key pair name: Вказується diploma-key

Key pair type: Залишається RSA (це стандарт шифрування).

Private key file format: Обирається .pem (це універсальний формат для OpenSSH).

Натискається Create key pair.

Браузером завантажується файл diploma-key.pem. Передбачено його зберігання в надійній папці на комп'ютері.

Що це таке: AWS не використовує паролі для входу адміністратора (root), бо їх легко підібрати. Замість цього використовується асиметрична криптографія.

Публічний ключ AWS покладається на сервер при запуску.

Приватний ключ (.pem) - зберігається в надійному місці.

Сервером надається доступ тільки за наявності цього файлу. Без нього доступ неможливий.

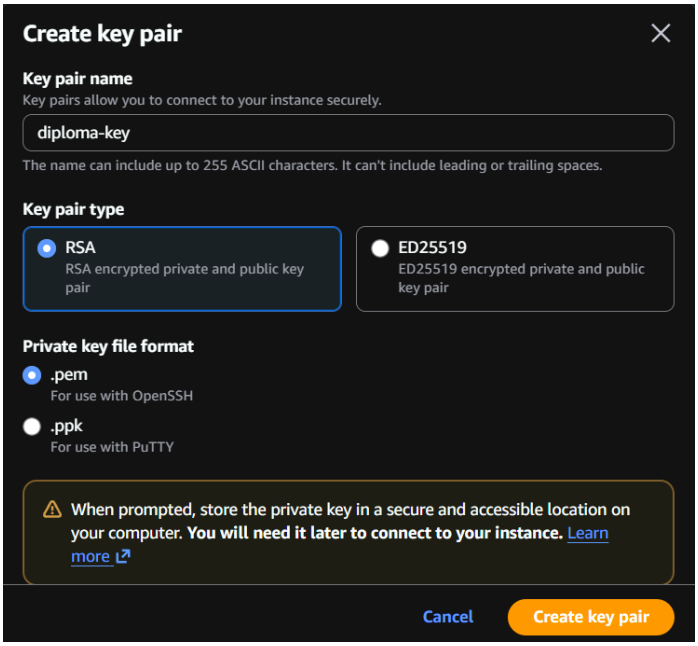


Рисунок 4.6 – Вікно створення ключів

Крок 5. Мережа (Network settings):

Це фундамент, тут сервер розміщується у "офіс зі стінами", який було створено на Етапі 1.

У блоці "Network settings" натискається кнопка Edit (Редагувати) справа зверху блоку.

VPC: Відкривається список і обирається власна мережа: SHK-Diploma-vpc. (Цим дією сервер ізолюється у власній хмарі).

Subnet: Обирається підмережа, у назві якої є слово public (у даному випадку - SHK-Diploma-vpc-subnet-public1). (Сервер розміщується у "публічну зону" для забезпечення виходу в інтернет).

Рисунок 4.7 – Мережеві налаштування

Auto-assign public IP: Змінюється на Enable. (Це критично важливо! Якщо це не увімкнути, сервер не отримає публічної адреси, і підключитися до нього з дому буде неможливо. Він буде "німим" для світу).

Крок 6. Охоронець (Security Group):

Створюються правила: кому можна "стукати" у двері сервера.

Виконується перевірка, що вибрано Create security group.

Security group name: Текст змінюється на Web-Server-SG

Description: Вказується Allow SSH, HTTP, HTTPS

Нижче наведено правила (Inbound security group rules). Налаштування виконується наступним чином:

Правило 1:

Type: SSH.

Source type: Змінюється "Anywhere" на My IP. (Це відкриває порт 22 (керування) тільки для власного комп'ютера).

Правило 2: Натискається Add security group rule.

Type: HTTP. Source type: Anywhere (0.0.0.0/0). (Порт 80. Дозволяється всім бачити сайт).

Правило 3: Type: HTTPS. Source type: Anywhere (0.0.0.0/0). (Порт 443. Для майбутнього захищеного з'єднання).

Крок 7. Диск (Storage):

Configure storage: Встановлюється 10 GiB, gp3. (Це віртуальний жорсткий диск (EBS Volume). 10 ГБ цілком вистачить для Linux і веб-сервера).

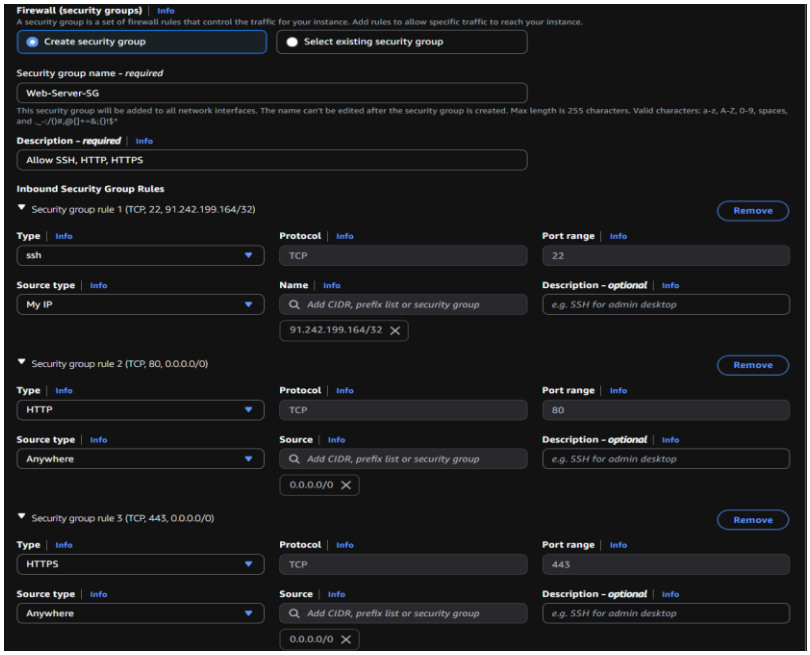


Рисунок 4.8 – Налаштування Security Group

Крок 8. Запуск

Натискається помаранчева кнопка Launch instance.

Після цього: AWS бере налаштування, шукає вільне місце на стійці у Франкфурті, запускає віртуалізацію, монтує мережу VPC, прописує ключ безпеки і стартує Ubuntu.

Подальші дії:

Відображається зелене повідомлення "Success".

Натискається на ID інстансу.

Здійснюється перехід у список серверів. Статус буде Pending (жовтий).

Очікується 30-60 секунд, поки він не стане Running (зелений).

4.3 Перетворення "пустої коробки" на Веб-сервер

На цьому етапі встановлюється Nginx на сервер. Nginx – це програмне забезпечення веб-сервіра.

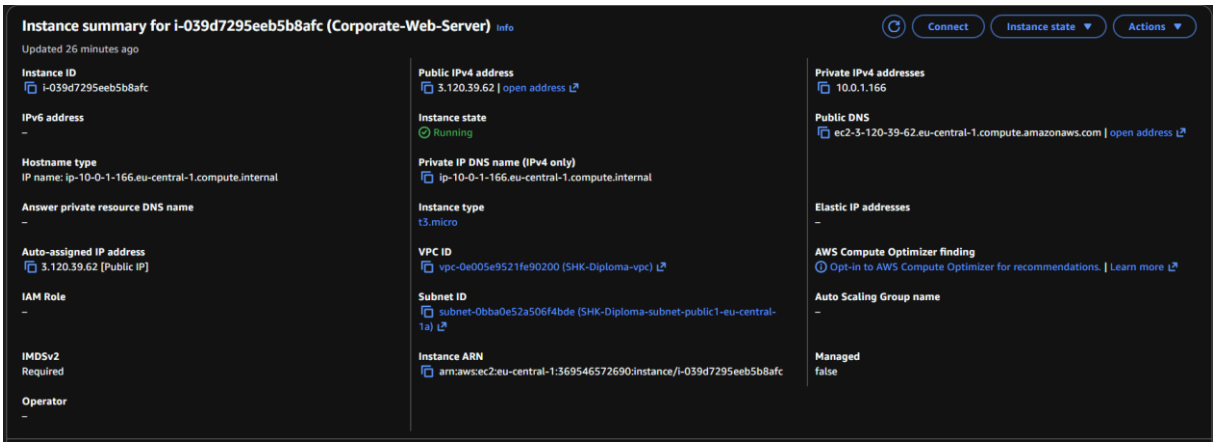


Рисунок 4.9 – Панель керування EC2: верифікація успішного запуску інстансу в статусі Running

Як він працює:

1. Nginx постійно "слухає" вхідні двері (Порт 80).
2. Як тільки надходить запит (клієнтом введено IP-адресу), Nginx приймає його.
3. Відбувається перехід у папку з файлами (/var/www/html) та обирається потрібна сторінка (index.html).

4. Сторінка віддається клієнту.

Крок 1. Підготовка PuTTY (SSH Client)

Додавання створеного ключа до способів авторизації.

Попередньо виконується його переформатування в .ppk за допомогою PuTTYgen.

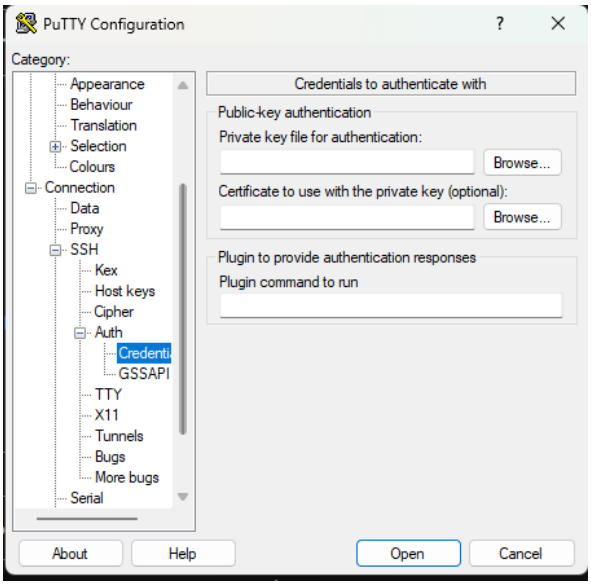


Рисунок 4.10 – Вікно посилання на ключ PuTTy

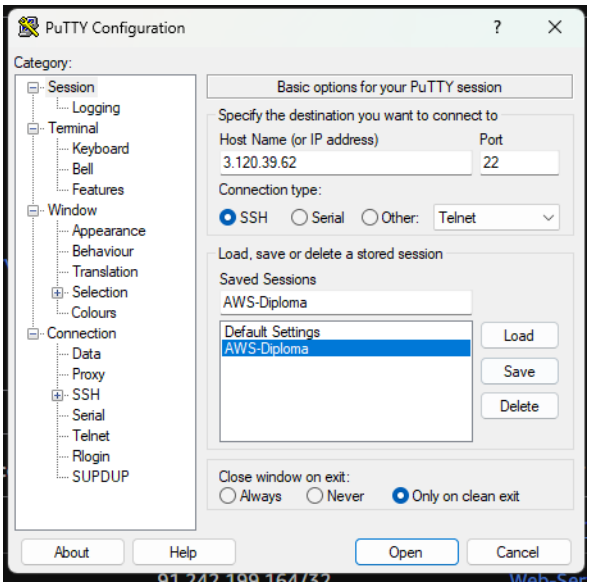


Рисунок 4.11 – Меню підключення до серверу PuTTy

Після цього здійснюється вхід на сервер під логіном ubuntu.

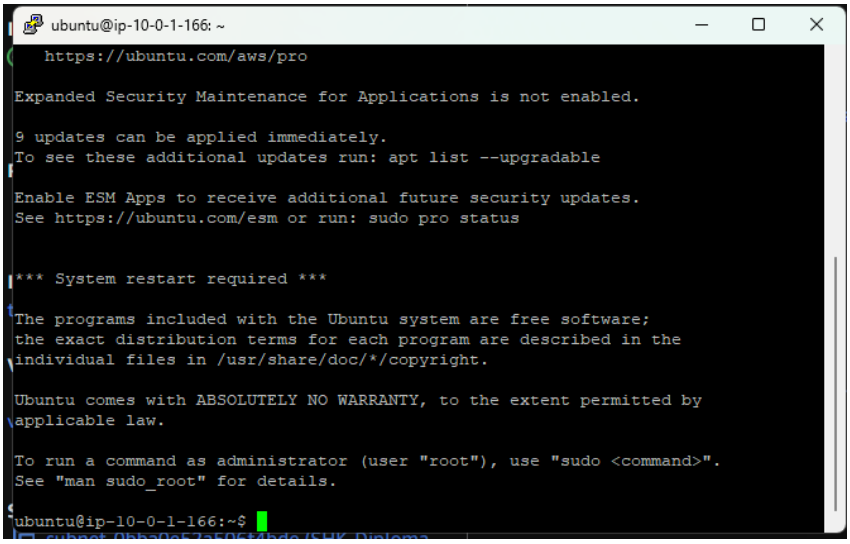


Рисунок 4.12 – Успішна авторизація на сервері через SSH

Крок 2. Оновлення каталогу

Перш ніж щось встановлювати, необхідно оновити базу даних пакетів Ubuntu.

```
ubuntu@ip-10-0-1-166:~$ sudo apt update
```

Крок 2. Встановлення Nginx

```
ubuntu@ip-10-0-1-166:~$ sudo apt install nginx -y
```

Крок 3. Перевірка статусу

```
ubuntu@ip-10-0-1-166:~$ sudo systemctl status nginx
```

```
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Sat 2025-11-29 15:21:48 UTC; 37s ago
       Docs: man:nginx(8)
    Process: 18783 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, st
    Process: 18785 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SU
   Main PID: 18815 (nginx)
      Tasks: 3 (limit: 1008)
     Memory: 2.4M (peak: 5.3M)
        CPU: 25ms
     CGroup: /system.slice/nginx.service
             ├─18815 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             ├─18817 "nginx: worker process"
             └─18818 "nginx: worker process"
```

Рисунок 4.13 – Успішне встановлення Nginx

Крок 4. Створення "Корпоративної" сторінки

За замовчуванням Nginx показує сторінку "Welcome to Nginx". Проте передбачено заміну стандартного коду файлу index.html на власний:

```
ubuntu@ip-10-0-1-166:~$ sudo tee /var/www/html/index.html
```

Тепер головну сторінку корпоративного сервера можна відкрити в браузері. Проте наразі це можливо тільки через http://

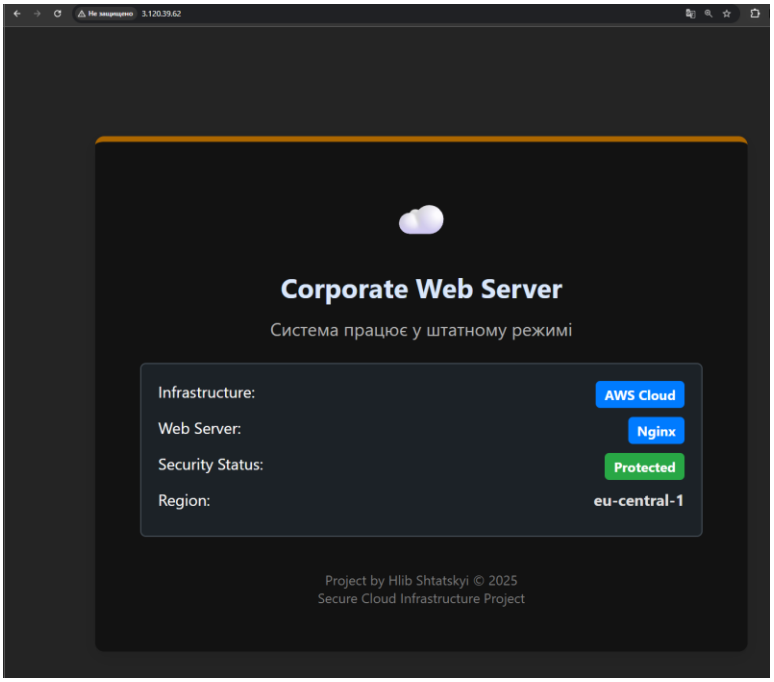


Рисунок 4.13 –Веб-сторінка нашого сервера через http підключеня

4.4 Створення Бази Даних (Amazon RDS)

Чому RDS, а не просто MySQL на сервері?

Встановлення MySQL можливе безпосередньо на той самий комп'ютер, де розміщено сайт (EC2). Це дешево, але небезпечно:

1. У разі зламу сайту зловмисники отримають доступ і до бази даних.

2. У разі відмови ("падіння") сервера припиняється робота всієї системи.

Amazon RDS (Relational Database Service) - це надання AWS окремого, налаштованого сервера спеціально під базу даних.

Передбачено автоматичне створення бекапів.

Оновлення відбувається автоматично.

База даних розміщується в приватній підмережі, куди доступ з інтернету повністю відсутній.

Крок 1. Створення RDS в AWS.

Здійснюється перехід до консолі RDS, а саме до розділу створення Database. Натискається Create Database та обирається

Крок 2. Налаштування.

1. Метод налаштування обирається - Full configuration (саме цей вибір дозволить зробити налаштування під власні потреби).

2. Engine option (рушій, який буде керувати збереженням інформації в Database)

Обирається MySQL - це стандарт для веб-розробки (особливо в зв'язці з Nginx/Apache). MySQL є найпопулярнішим рішенням у світі. Версія залишається за замовчуванням.

3. Templates (Шаблони) обирається Dev/Test задля подальших гнучких налаштувань.

4. DB instance identifier: Змінюється на зрозумілу назву, наприклад: diploma-db. Це буде внутрішня назва бази в консолі AWS.

5. Credentials Settings (Пароль)

Master username: Залишається admin.

Credentials management: Залишається Self managed (це є правильним, оскільки керування паролем здійснюється самостійно).

Master password: Генерується за допомогою Google Pass Manager.

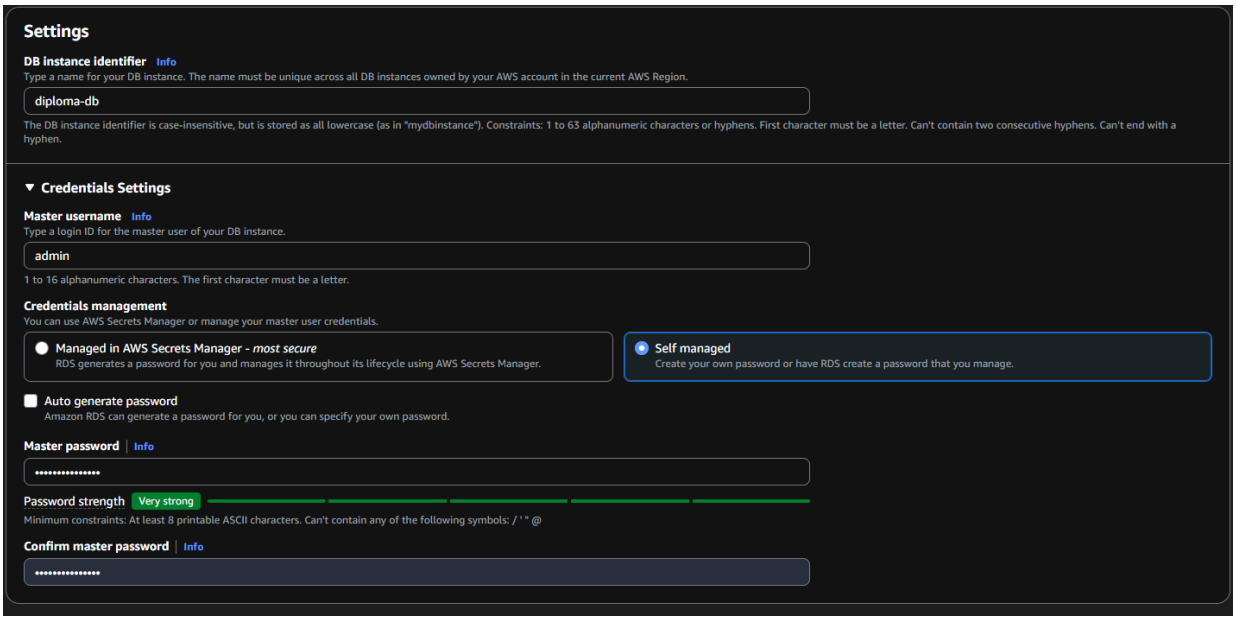


Рисунок 4.14 – Налаштування RDS

6. Instance configuration. Обирається Burstable -> db.t3.micro. Це найбільш оптимальні характеристики для даного сервера.

7. Налаштування Storage.

Storage Type (Тип сховища) - це, простими словами, технологія жорсткого диска, на якому фізично розміщуватиметься база даних у дата-центрі Amazon.

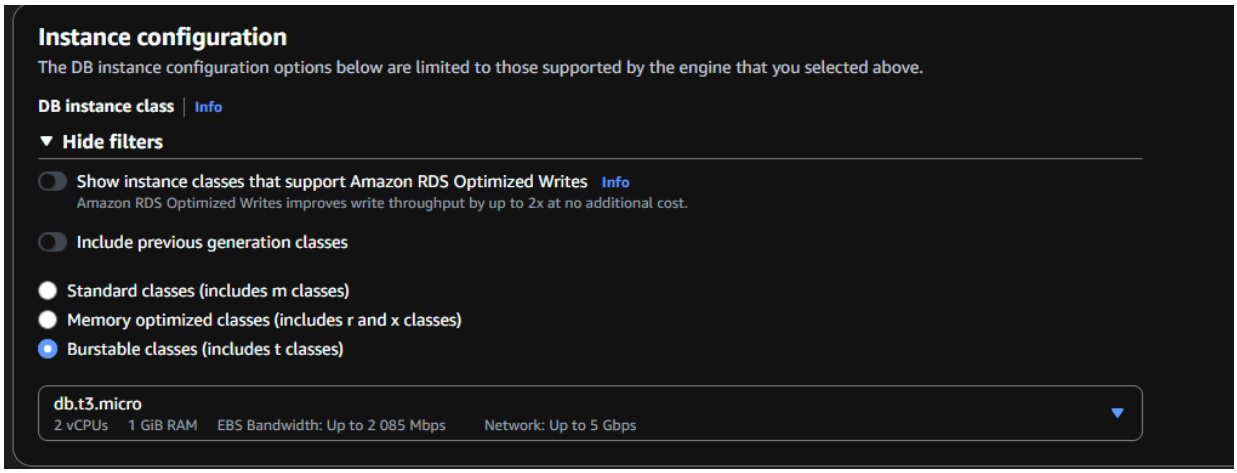


Рисунок 4.15 – Налаштування Instance для RDS

Вибір:

General Purpose SSD (gp2 або gp3) - "Золота середина".

Це SSD-диски загального призначення. Вони швидкі, надійні та відносно недорогі. Це стандартний вибір для 90% веб-сайтів.

8. Connectivity (Підключення).

Це критично важливий етап. Тут визначається, де розміщується база даних і хто має до неї доступ.

У разі помилки з VPC, сайт (EC2) не зможе виявити базу даних.

Налаштування:

Virtual private cloud (VPC): Обирається SHK-Diploma-vpc, оскільки саме тут знаходиться сервер EC2.

Public access (Публічний доступ): Обирається - NO. Це найважливіше налаштування безпеки. Відмовляється у прямому доступі з Інтернету. До бази можна дістатися тільки зсередини VPC.

VPC security group (firewall): Створюється нова група. Називається SHK-Diploma-DB-SG. Створюється персональний "охоронець" (брандмауер) для бази даних, де пізніше буде дозволено вхід тільки від веб-сервера.

Availability Zone (Зона доступності): Обирається - No preference. Оскільки фізичне розміщення сервера в конкретному дата-центрі (a, b або c)

не має значення, головною умовою є перебування в межах обраного регіону. AWS автоматично підбирає оптимальний варіант.

RDS Proxy & Certificate Authority. RDS Proxy - вимикається (прапорець не встановлюється), CA - залишається за замовчуванням. Використання RDS Proxy передбачено для дуже великих навантажень і потребує додаткових витрат. Сертифікат залишається стандартним для шифрування з'єднання.

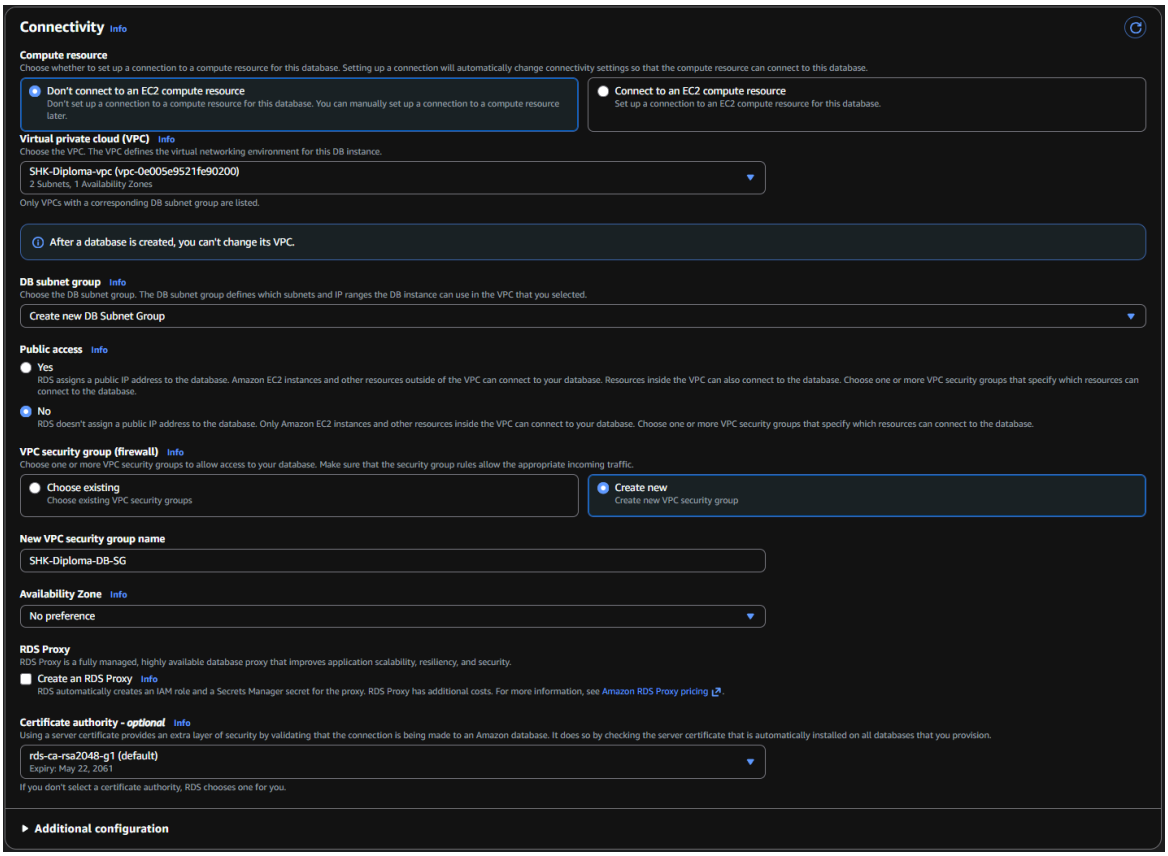


Рисунок 4.16 – Налаштування мережевих параметрів Amazon RDS

Database authentication (Автентифікація в базі даних).

Залишається Password authentication. Це класичний спосіб входу за логіном (admin) та паролем, який було створено раніше. Інші методи (IAM або Kerberos) є занадто складними для цього етапу і вимагають зміни коду сайту.

9. Tags (Теги). Задля подальшої зручності маркується приналежність БД.

Key: Project Value: Diploma-Web-Server

10. Monitoring. Вимикається Enhanced monitoring. Log exports: усі квадратики залишаються порожніми, оскільки експорт логів передбачає додаткові витрати. Таким чином обирається стандартний рівень моніторингу (Standard Monitoring), який автоматично збирає ключові метрики продуктивності: завантаження процесора (CPU Utilization), використання вільної пам'яті (Freeable Memory) та обсяг вільного дискового простору. Метрики збираються з інтервалом у 5 хвилин, що є достатнім для аналізу трендів навантаження без додаткових витрат на посилений моніторинг (Enhanced Monitoring).

11. Additional configuration. Додається наступне: Initial database name (назва corporatedb) - це назва конкретної БД всередині цього окремого сервера для зберігання баз даних. Сервер MySQL може містити всередині себе багато різних баз даних (одна для сайту, одна для форуму, одна для бухгалтерії). Ця опція створює базу даних одразу всередині сервера RDS після його створення. Натискається Create та виконується наступний крок.

12. Налаштування створеної Security Group. Для цього здійснюється перехід в EC2 – Security Group - SHK-Diploma-DB-SG. Натискається Edit та виконуються наступні налаштування: Type: MYSQL/Aurora Protocol: TCP Port range: 3306 Source: Custom - Web-Server-SG Створюється.

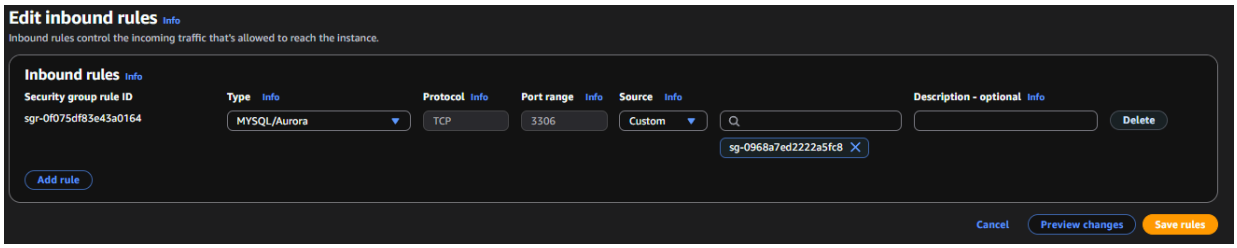


Рисунок 4.17 – Налаштування Inbound Rules (Вхідних правил) для бази даних RDS: реалізація довіри між Security Groups

Використаний механізм Security Group Referencing забезпечує високий рівень безпеки та гнучкості інфраструктури. На відміну від фільтрації за статичними IP-адресами (білі списки), цей підхід дозволяє динамічно керувати доступом на основі логічного групування ресурсів.

Це означає, що доступ до порту 3306 бази даних автоматично надається будь-якому інстансу, якому призначено групу безпеки веб-сервера (Web-Server-SG). Таке рішення усуває проблему зміни IP-адрес при перезапуску інстансів та значно спрощує масштабування системи, оскільки нові сервери веб-кластера отримують необхідні доступи автоматично без втручання адміністратора.

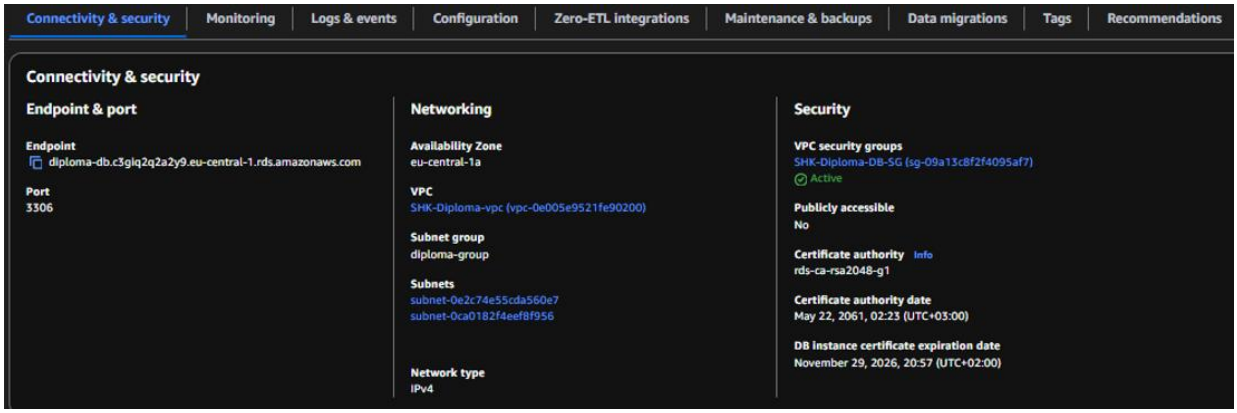


Рисунок 4.18 –Налаштування успішно встановленного RDS

4.5 Створення бакету в Amazon S3

Amazon S3 (Simple Storage Service) - це об'єктне сховище. Його можна уявити як бездонну флешку в хмарі.

У проєкті передбачено використання S3 для зберігання бекапів бази даних та файлів сайту.

Чому це важливо: навіть у разі критичного пошкодження дата-центру з сервером EC2, дані в S3 залишаться неушкодженими. Така надійність досягається архітектурно: будь-який об'єкт, завантажений у бакет S3, автоматично реплікується щонайменше у три фізично ізольовані зони доступності (Availability Zones) в межах регіону.

Безпека: Налаштовується шифрування та версіонування (захист від випадкового видалення або вірусів-шифрувальників).

Створення S3 Бакету:

1. Здійснюється вхід у сервіс. Через пошук виконується перехід до сервісу S3 та натискається Create Bucket.

2. Name and Region (Назва та Регіон). Bucket name: Назва бакету має бути унікальною на весь світ (як адреса Gmail). Створюється унікальна назва, пов'язана з проєктом - diploma-backup-shtatskyi-2025. AWS Region: Обирається Frankfurt (eu-central-1). Бекапи мають зберігатися поруч із сервером.

3. Object Ownership (Налаштування власності). Обирається ACLs disabled (Recommended). Це сучасний стандарт безпеки, за якого керування доступом здійснюється через політики. Це налаштування вимикає застарілі списки контролю доступу (ACL) та передає керування дозволами виключно політикам бакета (Bucket Policies). Такий підхід гарантує, що всі об'єкти, завантажені в бакет (наприклад, резервні копії), будуть захищені.

4. Public Access (Публічний доступ). Будь-яка можливість доступу із публічного простору вимикається: встановлюється прапорець - Block all

public access. Оскільки бекапи є конфіденційною інформацією, доступ до них з інтернету забороняється. Доступ надається лише серверу та адміністратору.

5. Bucket Versioning (Версіонування). Обирається - Enable. У разі випадкового перезапису файлу бекапу пошкодженим файлом, S3 збереже обидві версії. Це забезпечує можливість "відкоту" назад і є елементом відмовостійкості.

6. Default encryption (Шифрування). Encryption type: Обирається Server-side encryption with Amazon S3 managed keys (SSE-S3). Bucket Key: Enable. Це зменшує кількість запитів до сервісу KMS за рахунок використання проміжних ключів, що знижує навантаження на систему керування ключами без зниження рівня безпеки даних.

Вмикається базове шифрування "з коробки". Пізніше, у розділі "Безпека", передбачено зміну на SSE-KMS (власні ключі), що надасть додатковий рівень захисту: для читання файлу зловмиснику знадобляться права не тільки на S3, а й на KMS.

4.6 Створення адміністратора IAM

1. Створення User.

Здійснюється вхід у панель керування IAM. Натискається User, далі - Create User.

2. Specify user details.

User name: Вказується - Admin.

Provide user access to the AWS Management Console: Встановлюється прапорець. Це означає можливість входу користувача в консоль через графічний інтерфейс, а не тільки через програмний код. Пароль користувача створюється за допомогою Google Pass Manager.

3. Set permissions.

Обирається Attach policies directly (Приєднати політики напряму). Після цього відкривається список "Permissions policies" і рядок пошуку. У пошуку вказується: AdministratorAccess. Це надає користувачу повні права адміністратора в акаунті.

4. В консолі керування користувачем додається MFA за допомогою Google Authenticator.

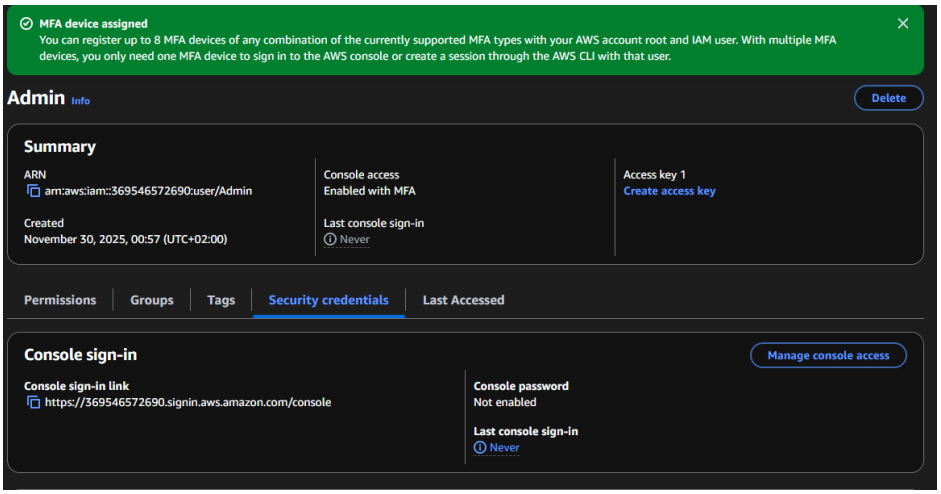


Рисунок 4.19 – Налаштування багатофакторної автентифікації (MFA)

4.7 Створення Ролі IAM для сервера

1. Створення ролі.

Здійснюється перехід до консолі керування IAM – Roles – Create role.

2. Trusted entity type (Найважливіший момент!)

Тип сутності: обирається AWS service.

Service or use case: у випадаючому списку обирається EC2.

Потім нижче обирається радіо-кнопка EC2. Саме цей крок робить роль видимою для серверів. Якщо обрати щось інше, сервер її не побачить.

3. Add permissions (Права). У пошуку здійснюється вибір двох політик: AmazonS3FullAccess (для забезпечення можливості запису сервером у бакет). AWSKeyManagementServicePowerUser (для забезпечення можливості використання сервером ключа шифрування).

4. Name. Роль називається: EC2-S3-KMS-Role

5. Додавання цього IAM до EC2 – Instances – Actions – Security – Modify IAM role.

6. Додавання до KMS – Key users

4.8 Створення ключа шифрування (AWS KMS)

1. Початок створення. Здійснюється перехід до меню налаштувань KMS та натискається Customer managed keys. Натискається помаранчева кнопка Create key.

2. Configure key (Тип ключа). Key type: обирається Symmetric (Симетричний). Цей вибір обумовлений використанням алгоритму AES-256 (Advanced Encryption Standard), який є галузевим стандартом для ефективного шифрування великих обсягів даних у спокої (Data at Rest). Key usage: Encrypt and decrypt. Це призначення ключа.

3. Add labels (Назва). Обирається назва та опис для ключа. Alias: diploma-s3-key Description: Master key for S3 backups

4. Define key administrators (Адміністратори).

Обирається створений користувач - Admin. Це означає, що користувач Admin має право керувати цим ключем.

5. Define key usage permissions.

Обирається створений користувач - Admin. Це дозволить (як адміністратору) переглядати зашифровані файли, якщо знадобиться їх відновити вручну.

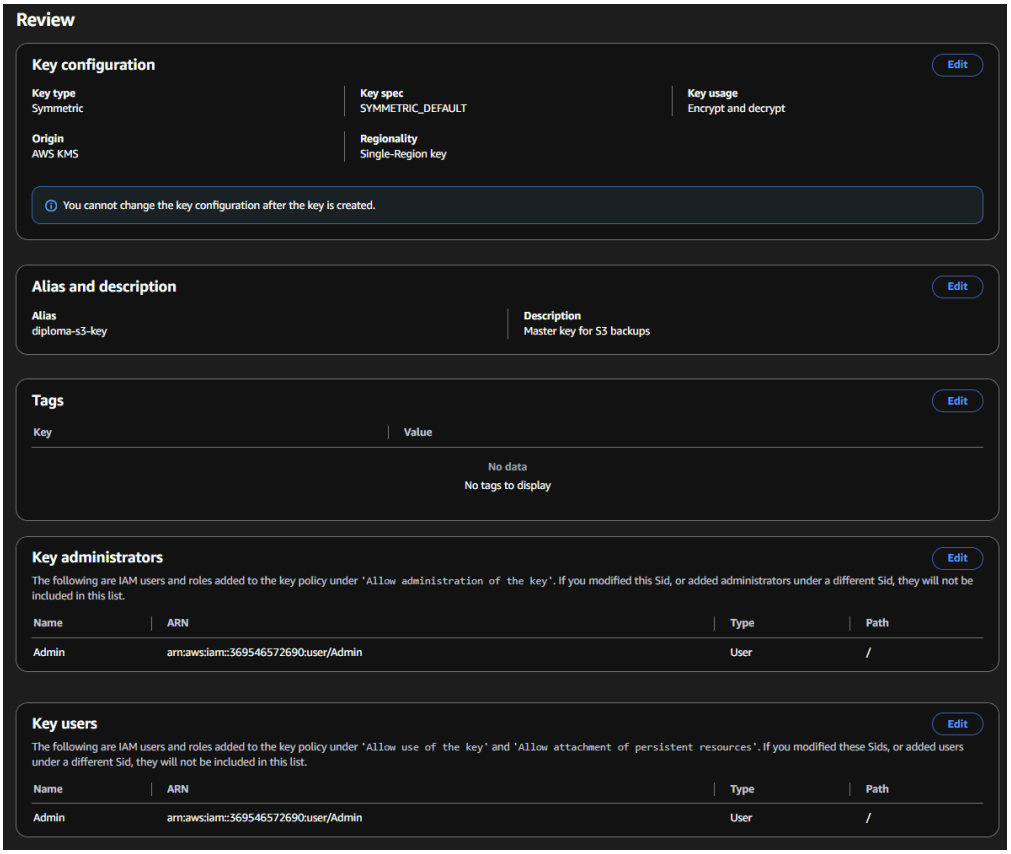


Рисунок 4.20 – Фінальна конфігурація KMS-ключа: призначення прав адміністратора та користувача

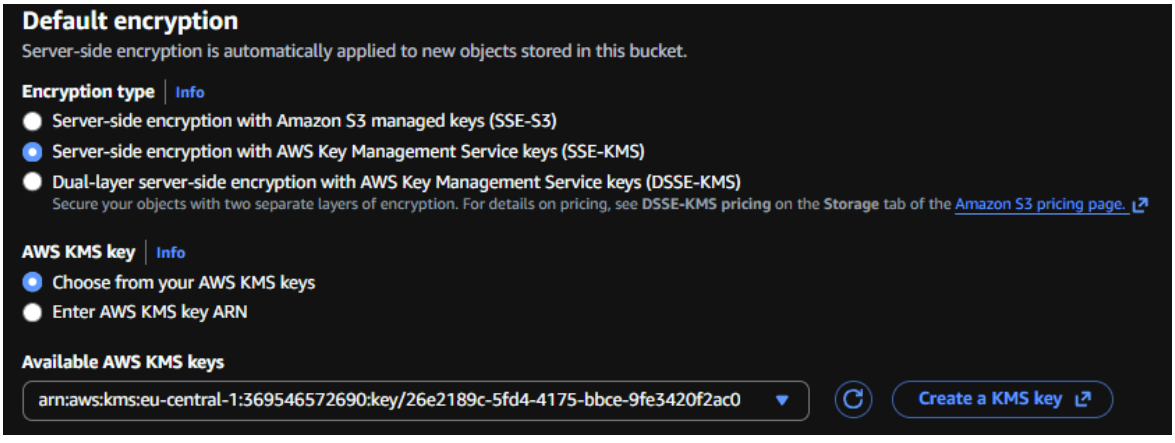


Рисунок 4.21 – Налаштування ключа шифрування для Bucket

6. Здійснюється повернення в налаштування вже створеного Bucket - виконується перехід у налаштування Default encryption та обирається створений ключ.

Це працює наступним чином: при спробі завантаження об'єкта в S3, сервіс сховища звертається до сервісу керування ключами (KMS) для генерації унікального ключа даних (Data Key).

Доступ до операцій шифрування та дешифрування надається виключно авторизованим сутностям (адміністратору та веб-серверу через IAM-роль), що унеможливлює несанкціонований доступ до даних навіть у випадку компрометації фізичних носіїв у дата-центрі провайдера.

4.9 Тестування зв'язку

Необхідно переконатися у працездатності налаштованої системи. Передбачено вхід на сервер та спробу завантаження файлу в бакет. Успішне виконання операції без введення логінів і паролів підтвердить коректність роботи IAM Role та KMS.

1. Встановлення інструментів AWS (для забезпечення взаємодії сервера з хмарою). Команди:

sudo apt update sudo apt install awscli -y

2. Створення тестового файлу (імітація важливого документа).

echo "Це перевірка шифрування KMS для диплома" > diploma_test.txt

3. Відправлення файлу у Сховище (S3).

aws s3 cp diploma_test.txt s3://diploma-backup-shtatskyi-2025/

Відображення повідомлення "upload: diploma_test.txt to s3://..." свідчить про успіх. Сервер виконав запис файлу, використовуючи IAM-роль та KMS-ключ.

4. Робота в Браузері (Перевірка результату).

Здійснюється перехід у S3 панель керування в AWS Console.

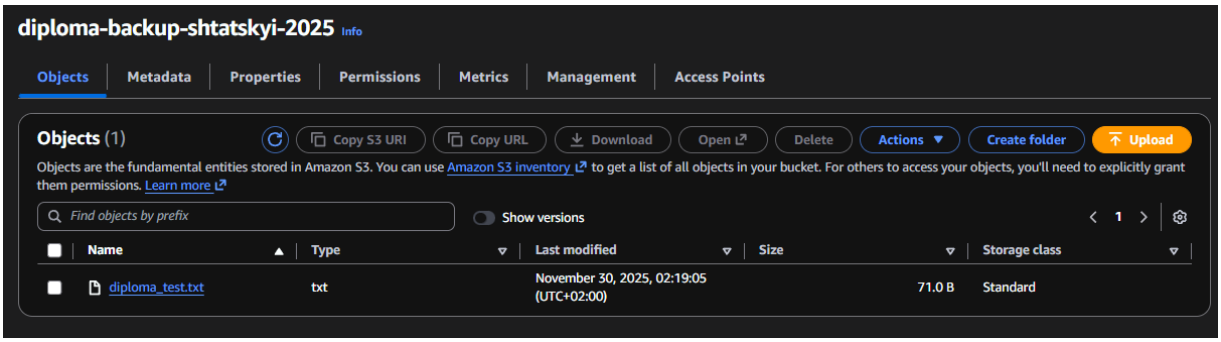


Рисунок 4.22 – Верифікація успішного запису тестового об'єкта в цільовий S3-бакет для резервного копіювання

На практиці реалізовано архітектуру нульової довіри ("Zero Trust Architecture"):

1. Відсутність паролів у коді сервера.
2. Використання сервером унікального "бейджа" (IAM Role).
3. Проходження перевірки у "охоронця ключів" (KMS).
4. Успішне зашифрування та розміщення файлу у "сейф" (S3).

При переході у властивості завантаженого файлу можна переконатися, що він зашифрований саме створеним ключем.

4.10 Перевірка зв'язку EC2 з RDS

1. Встановлення відносин довіри для Security groups.

Дуже важливий крок! Здійснюється перехід у консоль керування RDS та натискається на створений RDS. У розділі Connectivity & security натискається на Action у підрозділі Connected compute resources та виконується перехід у Set up EC2 connection. Натискається Set up.

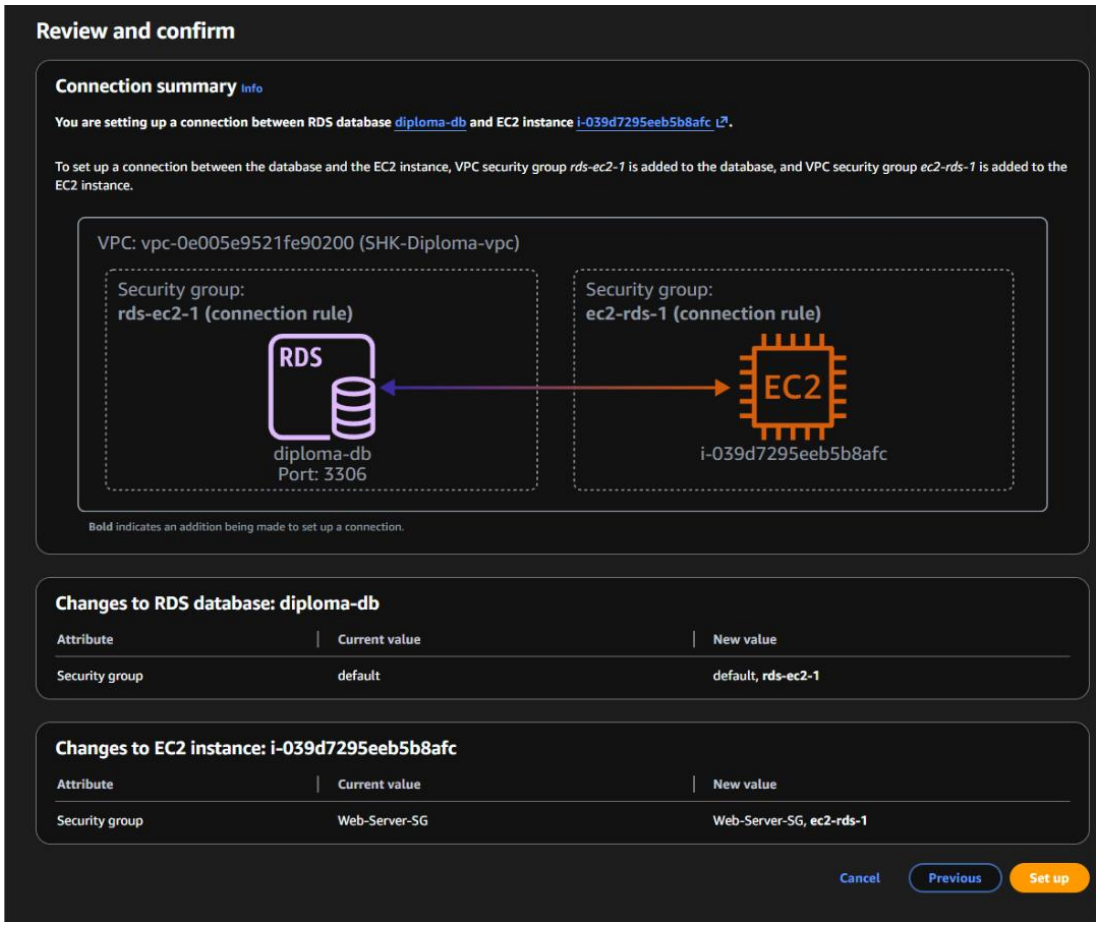


Рисунок 4.23 – Конфігурація взаємної довіри: автоматичне налаштування Security Group для зв'язку між RDS та EC2

Таким чином налагоджується взаємодія між двома групами security group RDS та EC2 шляхом додавання правила до групи RDS щодо довіри до EC2. Без цього кроку сервер не зможе з'єднатися з базою даних.

2. Перевірка зв'язку.

Для встановлення зв'язку з базою даних MySQL на сервері EC2 необхідно мати встановлений клієнт MySQL CLI. Передбачено його встановлення:

```
sudo apt install mysql-client-core-8.0 -y
```

Здійснюється підключення до бази даних.

Хост (Endpoint): https://www.google.com/search?q=diploma-db.c3giq2q2a2y9.eu-central-1.rds.amazonaws.com

Користувач: admin

За допомогою команди:

```
mysql -h diploma-db.c3giq2q2a2y9.eu-central-1.rds.amazonaws.com -u admin -p
```

Після цієї команди запитується пароль від RDS. Пароль було створено за допомогою Google Pass Manager, тому він копіюється зі сховища паролів.

Успішне підключення до бази даних підтверджується появою командного рядка mysql> та інформацією про версію сервера.

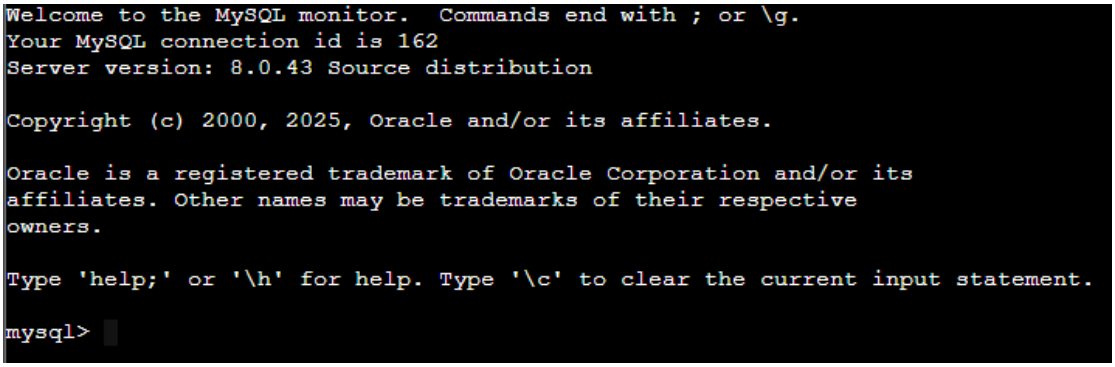


Рисунок 4.24 – Результат верифікації: успішна автентифікація та встановлення захищеного з'єднання EC2 з базою даних RDS

З'єднання встановлено, що підтверджує коректність роботи AWS Security Group (фаєрвол) щодо допуску вхідного трафіку на Port 3306 від Security Group EC2-сервера.

4.11 AWS Secret Manager.

Наступним кроком передбачено створення скрипту для автоматизації резервного копіювання. У цьому скрипті необхідно використовувати пароль від RDS. Для забезпечення безпеки та уникнення зберігання пароля у

відкритому вигляді (не залишати пароль просто в файлі скрипта), налаштовується AWS Secrets Manager.

1. Створення секрету (AWS Secrets Manager). Здійснюється перехід до консолі Secrets Manager та створюється новий секрет. Обирається Credentials for RDS database. Вводяться облікові дані, які зберігаються в Google Pass Manager.

Encryption key (Ключ шифрування): Обирається aws/secretsmanager. Це стандартний ключ KMS, який AWS використовує для шифрування секрету. Це повністю відповідає вимогам безпеки.

Database (База даних): обирається інстанс diploma-db.

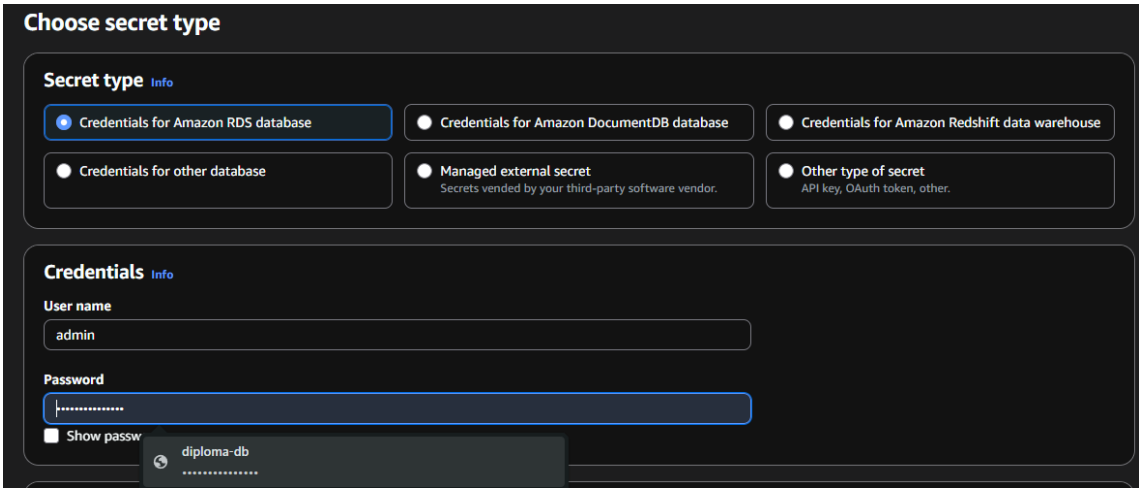


Рисунок 4.25 – Налаштування AWS Secrets Manager

Створюється назва та опис для секрету.

Configure rotation - пропускається, оскільки ця функція не є необхідною. Це механізм, який дозволяє Secrets Manager періодично і автоматично змінювати облікові дані (пароль, ключ API) на цільовому ресурсі (у цьому випадку - на базі даних RDS) без необхідності втручання людини. Для цього проєкту це є зайвим, оскільки вже встановлені налаштування повністю покривають вимоги до безпеки.

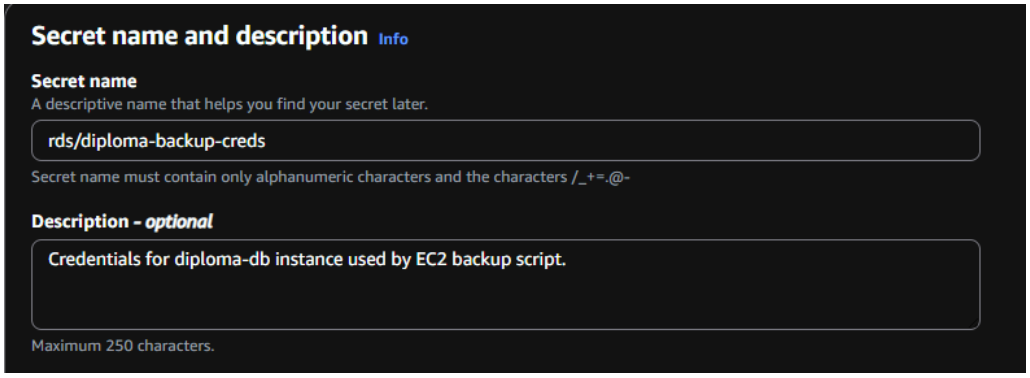


Рисунок 4.26 – Налаштування назві та опису AWS Secret Manager

Здійснюється перехід на етап Review та натискається Store.

Відображається створений секрет, який зберігається в зашифрованому вигляді.

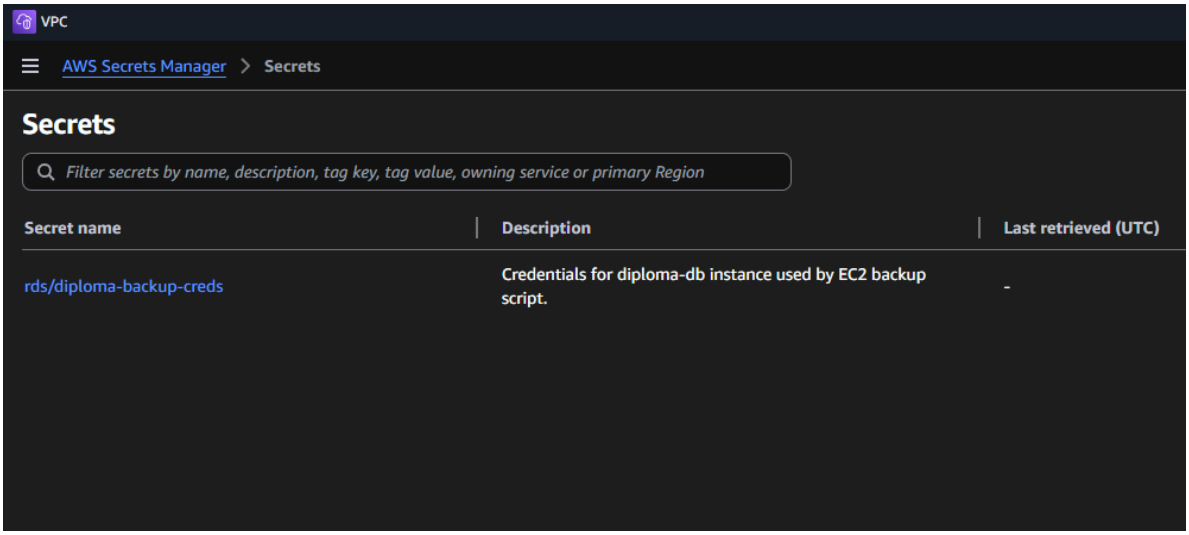


Рисунок 4.27 – Панель керування AWS Secrets Manager: зберігання облікових даних бази даних для автоматизованого сценарію

2. Налаштування дозволів (IAM Policy).

Передбачено надання серверу (EC2) прав на читання цього секрету. У консолі AWS здійснюється перехід в IAM – Roles. Виконується перехід до ролі EC2-S3-KMS-Role, яку було створено раніше.

Створення політики: У розділі "Permissions" натискається Add permissions – Create inline policy.

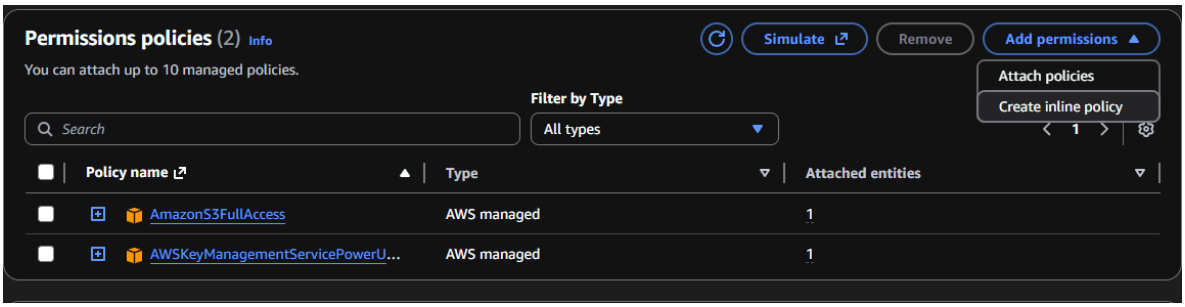


Рисунок 4.28 – Початкова конфігурація IAM політик: надання повного доступу до S3 та KMS для ролі сервера

У підрозділі Read обирається - GetSecretValue.

Налаштовується Specify ARNs.

Здійснюється копіювання Secret ARN з консолі керування секретом та вставлення у поле Resource ARN. Інші дані підтягуються автоматично.

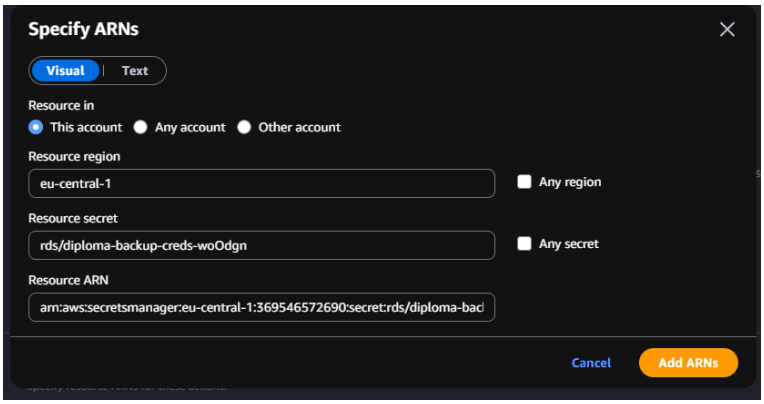


Рисунок 4.29 – Конфігурація політики найменших привілеїв: обмеження доступу EC2

Натискається Add ARNs та натискається Next.

В вікні Review додається назва - Policy-Get-DiplomaDB-Secret.

Та натискається Create Policy.

4.12 Удосконалення політик безпеки

Оскільки при створенні інфраструктури для початкових тестів надавались максимальні права для EC2-S3-KMS-Role в роботі з сервісами S3 та KMS, наразі впроваджуються обмеження, які застосовуються для справжньої робочої інфраструктури. Для цього передбачено видалення старих політик S3 та KMS та створення нових через Create Policy Line.

1. Для S3. Read - обирається тільки GetBucketLocation. Write - обирається тільки PutObject. У Resources в bucket додається ARN arn:aws:s3:::diploma-backup-shtatskyi-2025 У Resources в object додається ARN arn:aws:s3:::diploma-backup-shtatskyi-2025/* Політиці надається назва Policy-Minimal-S3-Upload та зберігається.

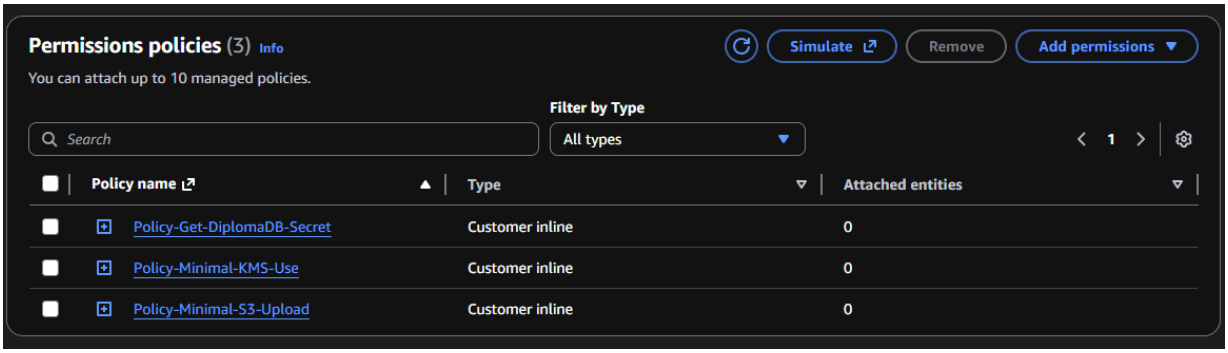


Рисунок 4.30 – Реалізація принципу найменших привілеїв: фінальні Inline Policies для доступу до S3, KMS та Secrets Manager

2. Для KMS. У розділі Write ставиться галочка лише навпроти дії: GenerateDataKey. Ця дія дозволяє AWS S3 попросити KMS згенерувати унікальний ключ для шифрування цього конкретного файлу під час завантаження. Це мінімум, необхідний для роботи S3 SSE-KMS. Resources: натискається Add ARNs та вставляється ARN, який береться з консолі керування створеного KMS. У цьому випадку це: arn:aws:kms:eu-central-

1:369546572690:key/5f64-4175-bbce-9fe3420f2ac0 Додається ім'я Policy-Minimal-KMS-Use та створюється.

Таким чином впроваджено комплексний механізм безпеки для виконання щоденного бекапу бази даних. Усі надлишкові політики видалено та замінено трьома Inline Policies, які суворо дотримуються Принципу найменших привілеїв.

Це рішення демонструє: (1) Безпарольну автентифікацію для Secrets Manager, (2) Обмеження доступу (Upload Only) для S3 та (3) Керування ключами (GenerateDataKey) для KMS. Інфраструктура є функціональною та високозахищеною.

4.13 Автоматизація резервного копіювання

У цьому розділі передбачено створення скрипту на Bash, який буде виконувати повний зашифрований цикл резервного копіювання з використанням підтверджених облікових даних.

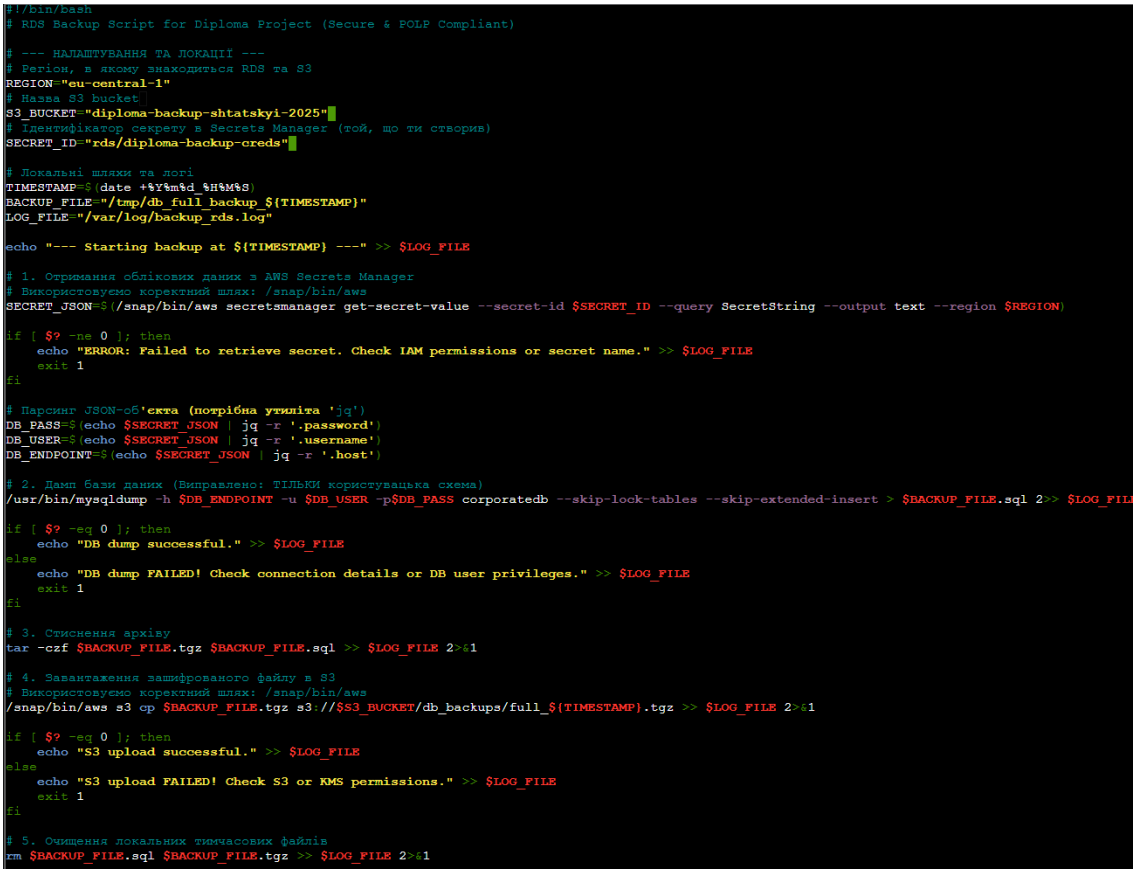
Створення та конфігурація скрипту бекапу: Сценарій буде використовувати інструмент mysqldump для копіювання бази даних та AWS CLI для завантаження в S3 (де дані будуть автоматично зашифровані ключем KMS).

Завдяки інтеграції з AWS Secrets Manager, скрипт зможе динамічно отримувати необхідні параметри доступу, не зберігаючи їх у відкритому вигляді. Це забезпечує високий рівень автоматизації та безпеки, що є критично важливим для стабільної роботи корпоративної інфраструктури. Такий підхід мінімізує ризики, пов'язані з людським фактором та можливим витоком конфіденційних даних.

1. Створення файлу скрипту (в терміналі EC2):

```
sudo nano /usr/local/bin/backup_rds.sh
```

2. Вставлення коду скрипту.



```
#!/bin/bash
# RDS Backup Script for Diploma Project (Secure & POLP Compliant)

# --- НАЛАШТУВАННЯ ТА ЛОКАЦІЇ ---
# Регіон, в якому знаходиться RDS та S3
REGION="eu-central-1"
# Назва S3 bucket
S3_BUCKET="diploma-backup-shtatskyi-2025"
# Ідентифікатор секрету в Secrets Manager (той, що ти створив)
SECRET_ID="rds/diploma-backup-creds"

# Локальні шляхи та логі
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="/tmp/db_full_backup_${TIMESTAMP}"
LOG_FILE="/var/log/backup_rds.log"

echo "--- Starting backup at ${TIMESTAMP} ---" >> $LOG_FILE

# 1. Отримання облікових даних з AWS Secrets Manager
# Використовуємо коректний шлях: /snap/bin/aws
SECRET_JSON=$(/snap/bin/aws secretsmanager get-secret-value --secret-id $SECRET_ID --query SecretString --output text --region $REGION)

if [ $? -ne 0 ]; then
    echo "ERROR: Failed to retrieve secret. Check IAM permissions or secret name." >> $LOG_FILE
    exit 1
fi

# Парсинг JSON-об'єкта (потрібна утиліта 'jq')
DB_PASS=$(echo $SECRET_JSON | jq -r '.password')
DB_USER=$(echo $SECRET_JSON | jq -r '.username')
DB_ENDPOINT=$(echo $SECRET_JSON | jq -r '.host')

# 2. Дамп бази даних (Виправлено: ТІЛЬКИ користувацька схема)
/usr/bin/mysqldump -h $DB_ENDPOINT -u $DB_USER -p$DB_PASS corporatedb --skip-lock-tables --skip-extended-insert > $BACKUP_FILE.sql 2>> $LOG_FILE

if [ $? -eq 0 ]; then
    echo "DB dump successful." >> $LOG_FILE
else
    echo "DB dump FAILED! Check connection details or DB user privileges." >> $LOG_FILE
    exit 1
fi

# 3. Стиснення архіву
tar -czf $BACKUP_FILE.tgz $BACKUP_FILE.sql >> $LOG_FILE 2>&1

# 4. Завантаження зашифрованого файлу в S3
# Використовуємо коректний шлях: /snap/bin/aws
/snap/bin/aws s3 cp $BACKUP_FILE.tgz s3://$S3_BUCKET/db_backups/full_${TIMESTAMP}.tgz >> $LOG_FILE 2>&1

if [ $? -eq 0 ]; then
    echo "S3 upload successful." >> $LOG_FILE
else
    echo "S3 upload FAILED! Check S3 or KMS permissions." >> $LOG_FILE
    exit 1
fi

# 5. Очищення локальних тимчасових файлів
rm $BACKUP_FILE.sql $BACKUP_FILE.tgz >> $LOG_FILE 2>&1
```

Рисунок 4.31 – Код сценарію Bash для автоматизованого резервного копіювання: інтеграція AWS Secrets Manager та безпарольна передача даних до S3

3. Надання прав доступу.

```
sudo chmod +x /usr/local/bin/backup_rds.sh
```

4. Запуск скрипту.

```
sudo /usr/local/bin/backup_rds.sh
```

5. Перевірка логів скрипту.

```
cat /var/log/backup_rds.log
```

Скрипт **/usr/local/bin/backup_rds.sh** успішно завершив роботу. Exit Code: 0 (Успіх).

Результат: Файл **db_full_backup_[TIMESTAMP].tgz** створено, зашифровано та завантажено в S3-бакет diploma-backup-shtatskyi-2025.

Підтверджується успішне створення файлу бекапу з базою даних та його розміщення у сховищі S3.

4.14 Зовнішній доступ та HTTPS

Наразі сайт доступний лише за IP-адресою і без шифрування (HTTP). Для корпоративного сервера це є неприпустимим. Цей етап закриває вимогу щодо впровадження криптографічних механізмів на рівні трафіку.

1. Купівля домену та налаштування DNS (Route 53). Для отримання SSL-сертифіката необхідна реальна адреса. Здійснюється перехід до AWS Console - Route 53. Виконується перехід у Domain registration (Реєстрація доменів) та пошук вільного імені (наприклад, shtatskyi.com).

Підтверджується оплата, після чого заповнюються контактні дані.

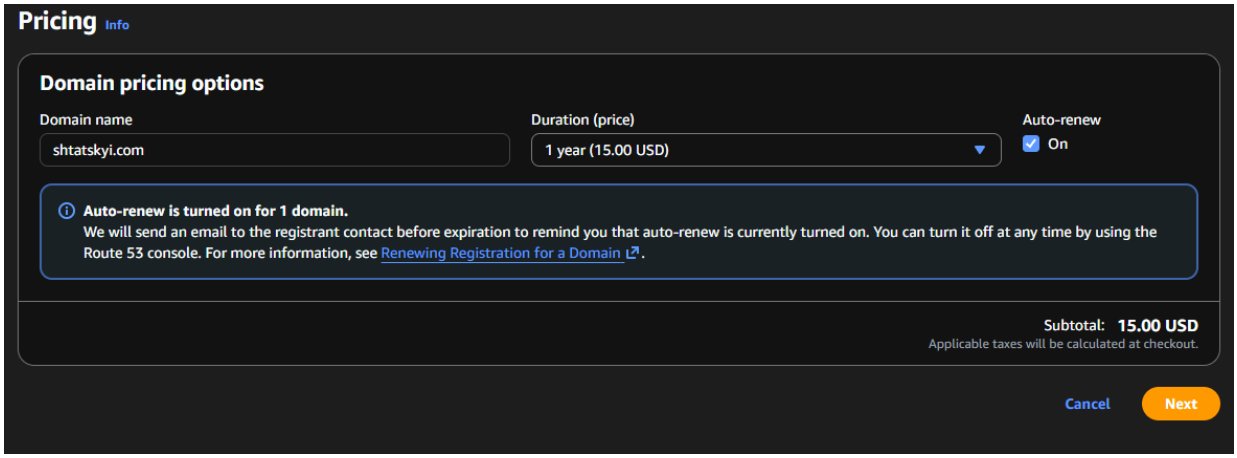


Рисунок 4.32 – Верифікація структури S3-бакету: створення цільової папки для автоматизованого резервного копіювання

Після придбання протягом 5–10 хвилин створюється Hosted Zone (Розміщена зона) - обов'язковий контейнер в сервісі AWS Route 53, який зберігає інформацію про те, як керувати доменом та його піддоменами.

Суть Hosted Zone: Контейнер DNS Hosted Zone містить усі DNS-записи (A-record, CNAME, MX тощо), які визначають, куди повинен спрямовуватися трафік, коли користувач вводить домен (shtatskyi.com).

Зв'язок: Саме на цьому етапі людське ім'я (shtatskyi.com) пов'язується з технічною адресою EC2-інстансу (Public IP).

Тип (Public): Оскільки створено Public Hosted Zone, записи в ній є видимими і доступними для запитів з усього публічного інтернету, що необхідно для роботи корпоративного веб-сайту.

2. Створення DNS-запису (A-Record). Здійснюється перехід до панелі керування EC2 та перехід на сервер. Копіюється Public IPv4 - 3.120.39.62. Виконується повернення до Route 53 - Hosted zones. Натискається на назву домену. Натискається Create record. Record name: залишається пустим (для прив'язки кореневого домену). Record type: обирається A - Routes traffic to an IPv4 address... (Це є найважливішим). Value (Значення): Вставляється скопійована Публічна IP-адреса EC2. Натискається Create records.

Аналогічні дії виконуються ще раз, але в полі name вписується www, для того щоб створити A-запис для www.shtatskyi.com.

Наразі домен наявний, але потрібно зачекати 2–5 хвилин, поки DNS-зміни поширяться світом. Цей час є ідеальним для підготовки сервера до встановлення сертифіката.

3. Встановлення SSL/HTTPS (Certbot).

Підготовка Nginx (В терміналі EC2): Certbot має отримати інформацію про домен, який підлягає захисту. Для цього необхідно налаштувати Nginx на прослуховування відповідного домену.

Здійснюється підключення до EC2-термінала (Connect – EC2 Instance Connect). Відкривається файл конфігурації Nginx:

```
sudo nano /etc/nginx/sites-available/default
```

Знаходиться рядок: server_name _; Та замінюється на доменне ім’я. Перезапускається Nginx для застосування змін:

```
sudo nginx -t
sudo systemctl restart nginx
```

Отримання сертифіката: Встановлюється Certbot - це безкоштовний клієнтський додаток, який автоматизує процес отримання та керування сертифікатами SSL/TLS від центру сертифікації Let's Encrypt.

```
sudo apt install certbot python3-certbot-nginx -y
```

Запускається автоматичне отримання SSL/TLS:

```
sudo certbot --nginx
```

Вказується адреса електронної пошти та підтверджується намір отримати сертифікат для обох записів: www.shtatskyi.com та shtatskyi.com. Після успішного створення відобразяться відповідні записи.

```
Deploying certificate
Successfully deployed certificate for shtatskyi.com to /etc/nginx/sites-enabled/default
Successfully deployed certificate for www.shtatskyi.com to /etc/nginx/sites-enabled/default
Congratulations! You have successfully enabled HTTPS on https://shtatskyi.com and https://www.shtatskyi.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
If you like Certbot, please consider supporting our work by:
 * Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
 * Donating to EFF:                    https://eff.org/donate-le
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ubuntu@ip-10-0-1-166:~$
```

Рисунок 4.33 – Підтвердження успішного впровадження SSL/TLS шифрування за допомогою клієнта Certbot

Цей успіх закриває ключову вимогу диплома щодо безпеки. Відтепер корпоративний сервер використовує SSL/TLS для шифрування всіх даних між клієнтом і сервером. Вебсторінка стає доступною для безпечної передачі даних клієнтами.

4.15 Моніторинг та Аудит

1. Інтелектуальне виявлення загроз (GuardDuty) - керований сервіс виявлення загроз (Threat Detection Service) від AWS. Основна функція полягає в постійному моніторингу хмарного середовища для виявлення шкідливої активності та несанкціонованої поведінки.

Здійснюється перехід до AWS Console - GuardDuty та активація сервісу.

GuardDuty використовує машинне навчання (ML), аналіз логів і поведінковий моніторинг для відстеження активності. На відміну від традиційного антивірусу, GuardDuty не встановлюється на інстанс EC2, а працює на рівні всієї інфраструктури AWS.

Для сканування сервера: У консолі керування обирається GuardDuty - Malware scans. Натискається Start new on-demand scan. Прописується ARN EC2 Instance. Починається сканування.

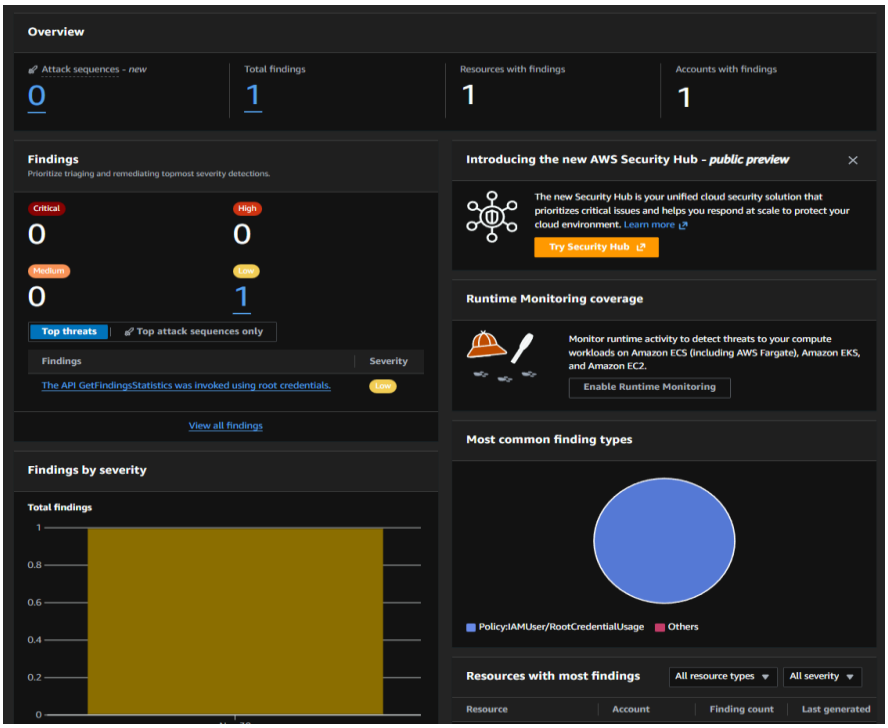


Рисунок 4.34 – Панель моніторингу AWS GuardDuty: верифікація активації сервісу інтелектуального виявлення загроз

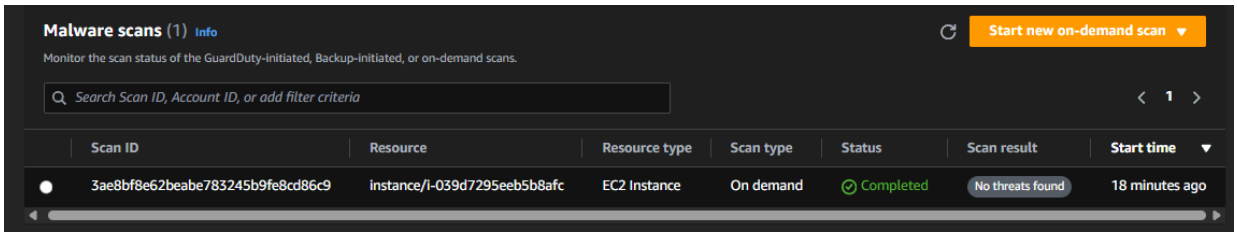


Рисунок 4.35 – Панель моніторингу GuardDuty: результати сканування EC2-інстансу на наявність шкідливого програмного забезпечення

2. CloudWatch Alarm (Автоматизований моніторинг).

Створення каналу сповіщення (SNS Topic): Служба Simple Notification Service (SNS) використовується для надсилання сповіщень (наприклад, на електронну пошту), коли спрацьовує тривога.

Здійснюється перехід до AWS Console - SNS. Виконується перехід у Topics - Create Topic. Type: Standard. Name: Diploma-Alarm-Topic. Далі - Create topic.

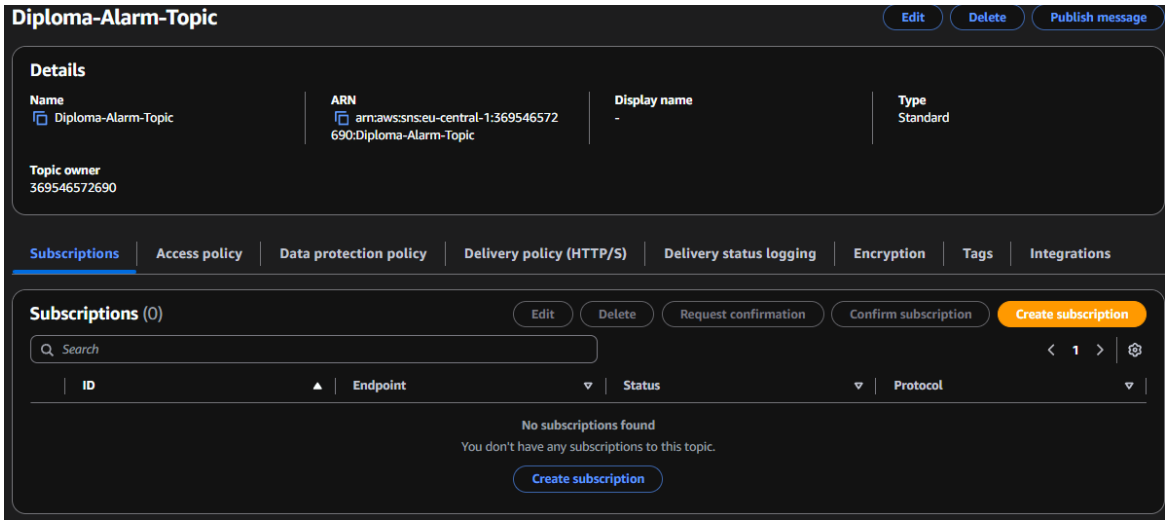


Рисунок 4.36 – Панель керування SNS: створення топіка для сповіщень CloudWatch Alarm

Створення підписки (Отримання Email): На сторінці створеного топіка Diploma-Alarm-Topic натискається Create subscription.

Protocol: Email Endpoint: shtatskiy.gl2@gmail.com

Надалі необхідно підтвердити підписку на сповіщення після отримання відповідного листа на пошту.

Створення Тривоги (CloudWatch Alarm): Здійснюється перехід до панелі керувань CloudWatch. Далі обирається Alarm - Create alarm.

У панелі налаштувань - Select metric обирається EC2 - Per-Instance Metrics.

Знаходиться метрика CPUUtilization (Використання CPU) для Corporate-Web-Server та обирається.

Conditions (Умови): Threshold type (Тип порога): Обирається Static. Whenever CPUUtilization is: Обирається Greater/Equal. Than (Чим): Вводиться 80. (Тривога спрацює, якщо CPU перевищить 80%).

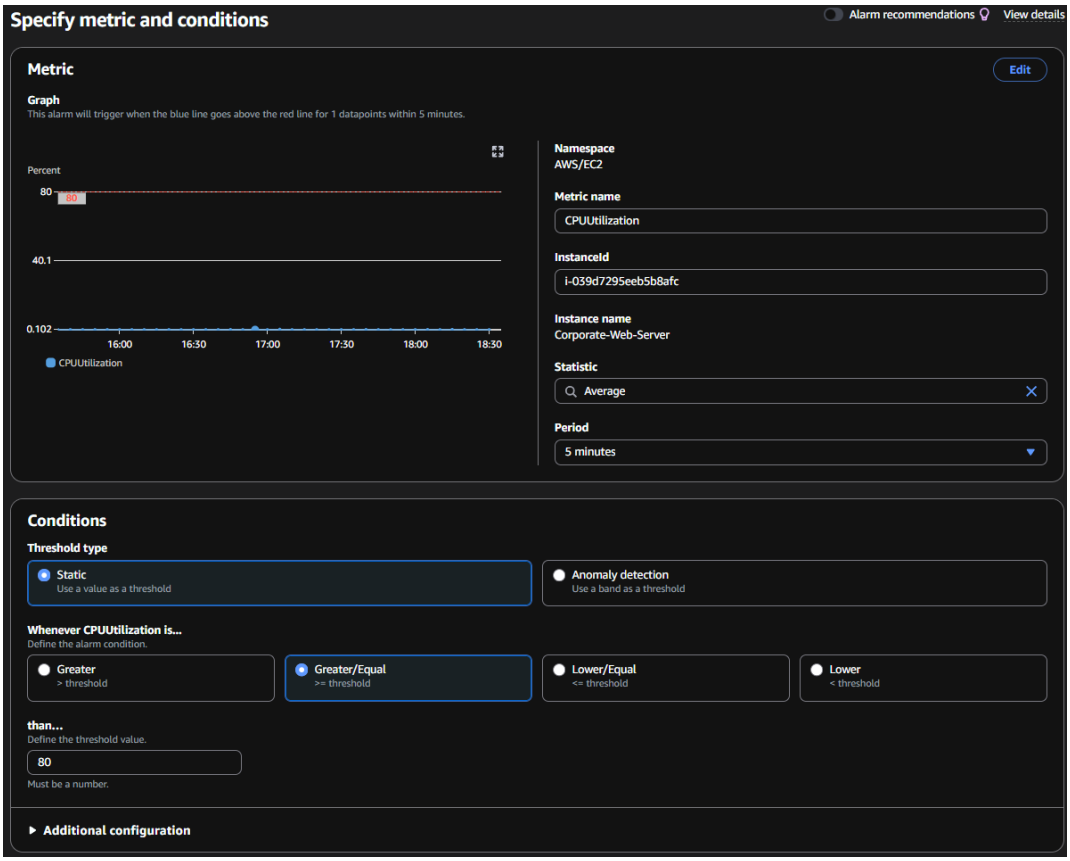


Рисунок 4.37 – Конфігурація тривоги CloudWatch: встановлення статичного порогу 80% для метрики CPUUtilization

Configure actions (Конфігурація дій): У розділі Select notification topic обирається - Select an existing SNS topic. Send a notification to... - обирається Diploma-Alarm-Topic.

Вводиться назва High-CPU-Load-Alarm та завершується створення.

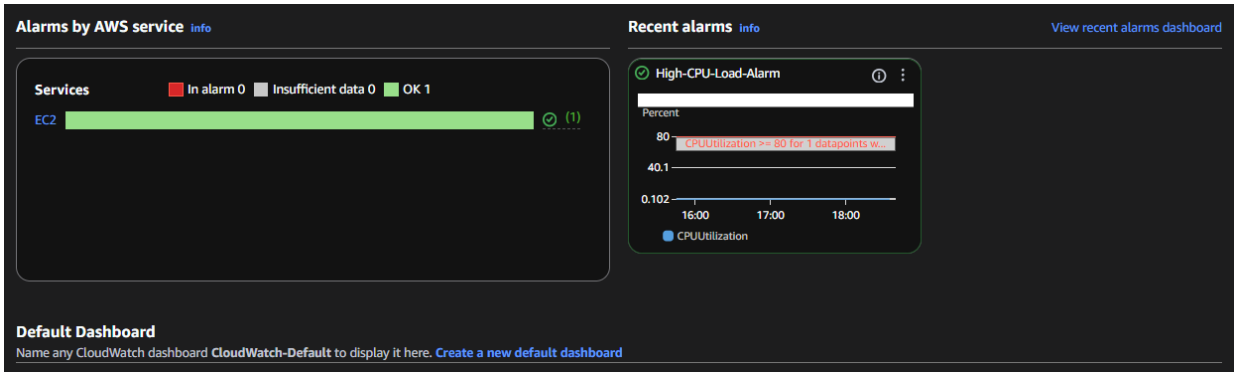


Рисунок 4.38 – Панель моніторингу CloudWatch: верифікація робочого стану тривоги CPU Utilization

3. Створення Dashboard на CloudWatch.

Здійснюється перехід у розділ Dashboards в CloudWatch - Create dashboard. Назва: Corporate-Server-Health.

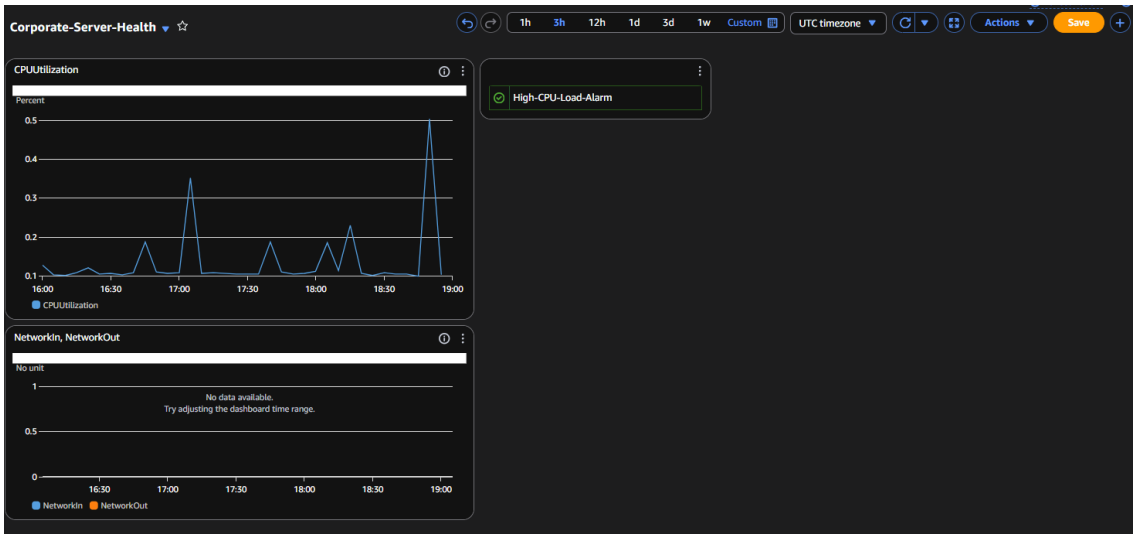


Рисунок 4.39 – Панель моніторингу CloudWatch Dashboard: візуалізація метрик CPU та статусу автоматизованої тривоги

Додавання віджетів (Метрики та Тривоги):

1. Віджет CPU Utilization (Навантаження): У спливаючому вікні обирається тип віджету Line (Лінійний графік) і натискається Next. Metrics (Метрики) - EC2 - Per-Instance Metrics. Знаходиться метрика CPUUtilization для Corporate-Web-Server. Натискається Create widget.

2. Віджет Статусу Тривоги (Alarm Status): Data Type: Alarm status. Обирається раніше створена тривога - High-CPU-Load-Alarm.

ВИСНОВКИ

У ході виконання дипломної роботи було проведено комплексне дослідження хмарних технологій як феномену сучасної IT-індустрії та реалізовано практичне впровадження захищеного корпоративного веб-сервера в середовищі Amazon Web Services (AWS).

Теоретичний аналіз показав, що хмарні обчислення пройшли шлях від концепції таймшерингу на мейнфреймах 1960-х років до сучасних глобальних екосистем IaaS, PaaS та SaaS . Було встановлено, що перехід до хмарної моделі дозволяє організаціям відмовитися від капітальних витрат на власне обладнання на користь операційних витрат за моделлю «pay-as-you-go», забезпечуючи при цьому безпрецедентну гнучкість та масштабованість.

У практичній частині роботи було успішно спроєктовано та розгорнуто відмовостійку серверну інфраструктуру, що відповідає сучасним стандартам кібербезпеки. Реалізація проєкту дозволила досягти наступних результатів.

1. Побудова захищеної архітектури (IaaS): Створено ізольоване мережеве середовище (VPC) з чітким розділенням на публічні та приватні підмережі, що унеможливлює прямий доступ до баз даних з мережі Інтернет. Використання груп безпеки (Security Groups) дозволило реалізувати динамічне керування доступом на основі логічного групування ресурсів, а не статичних IP-адрес .

2. Впровадження моделі «Zero Trust»: Налаштовано систему керування доступом IAM, де взаємодія між компонентами (EC2, S3, RDS) відбувається виключно через IAM-ролі та політики, без зберігання довгострокових ключів доступу на серверах . Для захисту облікових даних адміністратора впроваджено багатофакторну автентифікацію (MFA).

3. Криптографічний захист даних: Забезпечено наскрізне шифрування інформації. Трафік веб-ресурсу захищено протоколом HTTPS з використанням SSL/TLS-сертифіката Let's Encrypt . Дані у стані спокою

(резервні копії в S3) автоматично шифруються за допомогою керованих ключів сервісу AWS KMS (SSE-KMS), що гарантує конфіденційність навіть у разі фізичної компрометації носіїв .

4. Автоматизація та відмовостійкість: Розроблено та впроваджено скрипт автоматичного резервного копіювання бази даних, інтегрований з AWS Secrets Manager для безпечного отримання паролів . Це забезпечує регулярне створення зашифрованих архівів без втручання адміністратора.

5. Інтелектуальний моніторинг: Для проактивного захисту впроваджено систему AWS GuardDuty, яка використовує машинне навчання для виявлення аномалій та загроз у реальному часі . Системний моніторинг реалізовано через CloudWatch, що дозволяє оперативно реагувати на критичні навантаження сервера .

Таким чином, мета роботи досягнута в повному обсязі. Розроблена інфраструктура демонструє як поєднання сучасних хмарних інструментів дозволяє створити надійну, автоматизовану та захищену систему корпоративного рівня з мінімальними витратами. Отримані результати мають практичну цінність і можуть бути використані як шаблон для розгортання безпечних веб-сервісів у малому та середньому бізнесі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. [Електронний ресурс]: ілюстрація. – Режим доступу: https://www.collegenp.com/technology/what-is-cloud-computing

2. [Електронний ресурс]: ілюстрація. – Режим доступу: https://www.nature.com/articles/s41598-025-23865-4

3. [Електронний ресурс]: ілюстрація. – Режим доступу: https://quintagroup.com/blog/cloud-computing-step-up-your-business-opportunities-with-cloud-services

4. [Електронний ресурс]: ілюстрація. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:IBM_7090_computer.jpg

5. [Електронний ресурс]: ілюстрація. – Режим доступу: https://masonstevens.com.au/the-future-of-the-internet-part-1/

6. [Електронний ресурс]: ілюстрація. – Режим доступу: https://www.investopedia.com/terms/s/software-as-a-service-saas.asp

7. [Електронний ресурс]: інфографіка. – Режим доступу: https://www.moomoo.com/community/feed/oracle-earnings-review-up-9-post-earnings-is-oracle-s-113113129091077

8. [Електронний ресурс]: інфографіка. – Режим доступу: https://www.researchgate.net/figure/Cloud-Computing-Architecture_fig1_384561198

9. Маккарті Дж. Обчислення як комунальна послуга: Передумови появи хмарних обчислень. – Технічний вісник. – 2018. – № 5. – С. 15–28.2.

10. Челлапа Р. Обчислювальна парадигма: Визначення сутності хмарних сервісів. – IT-Наука, 2019. – № 12. – С. 56–69.

11. ДСТУ 8302:2015. Інформаційні технології. Шифрування даних: Основи та застосування алгоритму AES. – Київ: Держстандарт України, 2015. – 45 с.

12. [Електронний ресурс]: інфографіка. - Режим доступу: https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/

13. [Електронний ресурс]: схема. - Режим доступу: https://medium.com/@jitendra.bigtani/aws-networking-made-simple-2e1799054b5a

14. [Електронний ресурс]: ілюстрація. - Режим доступу: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/GettingStarted.html

15. Amazon Web Services. AWS Well-Architected Framework: Security Pillar. Керівництво по найкращим практикам безпеки у хмарі. – Офіційна документація. – Amazon Web Services, 2024. – 112 с.5.

16. AWS Identity and Access Management (IAM). Використання IAM Roles для керування доступом до сервісів RDS та S3. – Офіційний посібник. – Amazon Web Services, 2024. – 88 с.6.

17. AWS Key Management Service (KMS). Керівництво користувача: Принципи роботи та застосування клієнтських ключів (CMK). – Amazon Web Services, 2024. – 150 с.7.

18. AWS CloudWatch. Моніторинг, метрики та автоматизація тривог (Alarms). Керівництво користувача. – Amazon Web Services, 2024. – 150 с.

19. [Електронний ресурс]: ілюстрація. - Режим доступу: https://docs.paloaltonetworks.com/wildfire/u-v/wildfire-whats-new/latest-wildfire-cloud-features/sample-removal-request

20. [Електронний ресурс]: схема. - Режим доступу: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html

21. [Електронний ресурс]: схема. - Режим доступу: https://manoj-mahadadalkar.medium.com/using-multi-factor-authentication-mfa-in-aws-12adcb35122c

22. Let's Encrypt. Керівництво із забезпечення безпеки веб-сервера за допомогою Certbot. – [Електронний ресурс]. - Режим доступу: https://certbot.eff.org. – Дата звернення: 2025-11-30.