Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА:

на тему: «Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків»

08-20.МКР.026.00.000 ПЗ

Виконав: студент 2 курсу групи 1БС–22м
спеціальності 125 Кібербезпека
_____ Дмитро РАДЗІХОВСЬКИЙ

Керівник: к.ф.-м.н., доц. каф ЗІ
_____ Галина ШЕЛЕПАЛО
« ___ » _____ 2023 р.

Опонент: д.т.н., проф., зав. каф. ПЗ
_____ Олександр РОМАНЮК
« ___ » _____ 2023р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«____» _____ 2023 р.

Вінниця – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА:
на тему: «Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків»
08-20.МКР.026.00.000 ПЗ

Виконав: студент 2 курсу групи 1БС–22м
спеціальності 125 Кібербезпека
_____ Дмитро РАДЗІХОВСЬКИЙ

Керівник: к.ф.-м.н., доц. каф ЗІ
_____ Галина ШЕЛЕПАЛО
«11» 12 2023 р.

Опонент: д.т.н., проф., зав. каф. ПЗ
_____ Олександр РОМАНЮК
«12» 12 2023р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«13» 12 2023 р.

Вінниця – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ, д. т. н., проф.
_____Володимир ЛУЖЕЦЬКИЙ
_____ _____2023 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Радзіховському Дмитру Юрійовичу

1. Тема роботи: «Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків.»
керівник роботи: Шелепало Галина Василівна, к.ф.-м.н., доц. каф ЗІ,
затверджені наказом ректора ВНТУ від 18 вересня 2023 року №247.

2. Строк подання студентом роботи 11 грудня 2023 р.

3. Вихідні дані до роботи:
– операційна система – підтримує кросплатформність;
– хмарне середовище – AWS;
– середовище розробки – PyCharm Professional;
– мова програмування – Python;

4. Зміст розрахунково–пояснювальної записки: Вступ. Аналіз предметної області. Розробка структури засобу для вирішення задачі. Розробка системи захисту додатків в хмарному середовищі. Економічна частина. Висновки. Перелік використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: Регіони розподілу ресурсів AWS(плакат А4). Модель спільної відповідальності AWS (плакат А4). Модель спільної відповідальності для контейнерних послуг AWS (плакат А4). Сервіси AWS для керування безпекою(плакат А4). Опис AWS IAM Policy(плакат А4). Основні компоненти AWS IAM (плакат А4). Сценарій використання AWS Secret

Manager (плакат А4). Архітектура розробленої системи захисту (плакат А4). Створені ресурси за допомогою AWS Cloud Formation(плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ		
2	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ		
3	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ		
4	Ольга РАТУШНЯК, к.т.н., доц. каф ЕВПМ		

7. Дата видачі завдання 01.09.2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Науково-технічне обґрунтування	16.09.2023 – 22.09.2023	
4	Розробка технічного завдання	23.09.2023 – 29.09.2023	
5	Розробка рішень	30.09.2023 – 12.10.2023	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 – 10.11.2023	
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.23 – 17.11.2023	
8	Аналіз виконання ТЗ, висновки	18.11.2023 – 24.11.2023	
9	Оформлення пояснювальної записки	25.11.2023 – 30.11.2023	
10	Попередній захист та доопрацювання МКР	28.11.2023 – 01.12.2023	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2023 – 10.12.2023	
12	Представлення МКР до захисту	11.12.2023 – 14.12.2023	
13	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Дмитро РАДЗІХОВСЬКИЙ
(підпис)

Керівник роботи _____ Галина ШЕЛЕПАЛО
(підпис)

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ, д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
19 09 2023 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Радзіховському Дмитру Юрійовичу

1. Тема роботи: «Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків.»
керівник роботи: Шелепало Галина Василівна, к.ф.-м.н., доц. каф ЗІ,
затверджені наказом ректора ВНТУ від 18 вересня 2023 року №247.

2. Строк подання студентом роботи 11 грудня 2023 р.

3. Вихідні дані до роботи:
– операційна система – підтримує кросплатформність;
– хмарне середовище – AWS;
– середовище розробки – PyCharm Professional;
– мова програмування – Python;

4. Зміст розрахунково–пояснювальної записки: Вступ. Аналіз предметної області. Розробка структури засобу для вирішення задачі. Розробка системи захисту додатків в хмарному середовищі. Економічна частина. Висновки. Перелік використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: Регіони розподілу ресурсів AWS(плакат А4). Модель спільної відповідальності AWS (плакат А4). Модель спільної відповідальності для контейнерних послуг AWS (плакат А4). Сервіси AWS для керування безпекою(плакат А4). Опис AWS IAM Policy(плакат А4). Основні компоненти AWS IAM (плакат А4). Сценарій використання AWS Secret

Manager (плакат А4). Архітектура розробленої системи захисту (плакат А
Створені ресурси за допомогою AWS Cloud Formation(плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ	19.09	23.09
2	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ	19.09	30.09
3	Галина ШЕЛЕПАЛО, к.ф.-м.н., доц. каф ЗІ	19.09	21.10
4	Ольга РАТУШНЯК, к.т.н., доц. каф ЕВПМ	19.09	27.11

7. Дата видачі завдання 01.09.2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Науково-технічне обґрунтування	16.09.2023 – 22.09.2023	
4	Розробка технічного завдання	23.09.2023 – 29.09.2023	
5	Розробка рішень	30.09.2023 – 12.10.2023	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 – 10.11.2023	
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.23 – 17.11.2023	
8	Аналіз виконання ТЗ, висновки	18.11.2023 – 24.11.2023	
9	Оформлення пояснювальної записки	25.11.2023 – 30.11.2023	
10	Попередній захист та доопрацювання МКР	28.11.2023 – 01.12.2023	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2023 – 10.12.2023	
12	Представлення МКР до захисту	11.12.2023 – 14.12.2023	
13	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Дмитро РАДЗІХОВСЬКИ
(підпис)

Керівник роботи _____ Галина ШЕЛЕПАЛ
(підпис)

АНОТАЦІЯ

УДК 004.056.5(043.2)

Радзіховський Д.Ю. Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків. Магістерська кваліфікаційна робота зі спеціальності 125 Кібербезпека, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2023. 104 с.

Магістерська кваліфікаційна робота присвячена розробці та аналізу автоматизованої підсистеми, яка має на меті забезпечити безпеку застосунків, розгорнутих в хмарному середовищі. В контексті стрімкого росту хмарних технологій та їх широкого впровадження, питання безпеки є актуальним та вимагає новітніх підходів до його реалізації. У роботі проведено аналіз існуючих підходів до забезпечення безпеки застосунків в хмарах, ідентифіковано ключові ризики та виклики. Основна увага приділена розробці підсистеми, що включає архітектурне проектування, визначення основних компонентів, алгоритмів їх взаємодії та методів автоматизованого керування безпекою. Результатом роботи є програмна реалізація підсистеми, а також її імплементація в реальному хмарному середовищі з подальшим аналізом ефективності. Рекомендації, надані на основі дослідження, сприятимуть оптимізації процесів керування безпекою застосунків у хмарах.

В економічному розділі здійснено оцінку витрат на розробку інформаційної технології.

Ключові слова: хмарне середовище, безпека застосунків, автоматизоване керування.

ABSTRACT

Radzihovsky D.Y. Automated application security management system in the cloud environment. Part 2. Application Security Management Subsystem. Master's thesis on specialty 125 - Cybersecurity, educational program - Security of information and communication systems. Vinnytsia: VNTU, 2023. 104 p.

The Master's thesis is devoted to the development and analysis of an automated subsystem, which aims to ensure the security of applications deployed in a cloud environment. In the context of the rapid growth of cloud technologies and their wide implementation, the issue of security is urgent and requires the latest approaches to its implementation. The paper analyzes the existing approaches to ensuring the security of cloud applications, identifies key risks and challenges. The main attention is paid to the development of the subsystem, which includes architectural design, definition of the main components, algorithms of their interaction and methods of automated security management. The result of the work is the software implementation of the subsystem, as well as its implementation in a real cloud environment with further performance analysis. The recommendations provided on the basis of the study will contribute to the optimization of the security management processes of cloud applications.

In the economic section, an assessment of costs for the development of information technology was made.

Keywords: cloud environment, application security, automated management.

ЗМІСТ

ВСТУП

Сучасний світ переживає епоху цифрової трансформації, де інформаційні технології стають невід'ємною частиною практично всіх сфер життя суспільства. Зокрема, інноваційні рішення в області інформаційних систем, застосунків та інфраструктур набувають ключової важливості для підприємств, організацій та державних установ. Одним з напрямів, що найбільш активно розвивається в ІТ є хмарні технології, що є гнучким, масштабованим та економічно ефективним розв’язком для зберігання, обробки та управління даними.

В нас час, хмарні технології перетворилися на один з ключових компонентів сучасної ІТ-інфраструктури. Вони не лише пропонують нові можливості для зберігання даних та їх обробки, але і дозволяють користувачам отримувати доступ до ресурсів з будь-якої точки світу, надаючи необхідну гнучкість та масштабованість.

Однак, поряд з величезними перевагами, хмарні технології приносять і нові виклики у сферу інформаційної безпеки. Стрімкий розвиток і впровадження хмарних сервісів спричинюють появу нових загроз та вразливостей, що можуть бути використані зловмисниками.

Актуальність полягає в тому, що хмарні платформи невпинно розвиваються, пропонуючи підприємствам гнучкість, масштабованість та економію коштів. Цей швидкий розвиток зумовлює збільшення кількості даних і застосунків, розміщених у хмарі, що в свою чергу, потребує ефективного управління кібербезпекою. Сучасні хмарні середовища характеризуються високою складністю та різноманітністю, оскільки вони включають безліч різних сервісів і рішень. Така складність вимагає комплексного підходу до покращення безпеки, де автоматизовані системи відіграють ключову роль, пропонуючи універсальні та ефективні рішення для різноманітних сценаріїв безпеки в хмарних середовищах.

Об’єктом дослідження є процес автоматизованого налаштування безпеки застосунків у хмарному середовищі.

Предметом дослідження є підсистема керування безпекою застосунків яка автоматизує процеси ідентифікації, авторизації, автентифікації, аналізу та відгуку на потенційні загрози безпеки в хмарних застосунках.

Мета: автоматизації процесів розгортання та налаштування сервісів безпеки, завдяки розробці підсистеми керування безпекою, здатної гарантувати застосункам у хмарному середовищі високий рівень захищеності, враховуючи специфіку та особливості хмарних платформ.

Для виконання магістерської кваліфікаційної роботи необхідно виконати такі завдання:

- проаналізувати сучасні підходи до забезпечення безпеки застосунків у хмарних середовищах;
- визначити основні проблеми і слабкі місця з питань безпеки в хмарних середовищах;
- розробити систему з урахуванням специфіки хмарних середовищ;
- скласти загальні висновки з урахуванням результатів проведеного дослідження.

Новизна даного дослідження визначається комбінацією сучасних методів, підходів та технологій, що забезпечують комплексний підхід до конфігурації розробленої системи безпеки в хмарному середовищі, що дозволяє покращити захищеність додатків.

Практична цінність дослідження є важливим теоретично-прикладним вкладом у розвиток сфери хмарних обчислень та безпеки. Сучасний ринок переповнений організаціями, які активно використовують хмарні рішення в своїй діяльності. Розроблена підсистема зможе забезпечити їх надійність, конфіденційність та цілісність даних, підвищуючи довіру користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз популярних хмарних середовищ

Хмарні обчислення є зростаючим трендом у світі IT, і це не дарма. Хмарні постачальники можуть запропонувати низку переваг порівняно з традиційними локальними апаратними та програмними рішеннями, включаючи більшу масштабованість, гнучкість і економічну ефективність.

Коли вибір падає на хмарне рішення, по суті, відбувається передача управління обчислювальними ресурсами третій стороні. Це означає, що не доведеться турбуватися про початкові витрати, пов’язані з придбанням і обслуговуванням обладнання та програмного забезпечення [1].

Натомість, відкривається можливість отримувати доступ до потрібних ресурсів за потреби, сплачуючи лише за те, що використовується. Це може бути чудовим способом заощадити гроші, особливо якщо обчислювальні потреби піддаються стрибкам або регулярним коливанням.

Крім того, хмарні постачальники часто пропонують високий рівень безпеки та надійності, оскільки вони мають досвід та інфраструктуру для захисту даних. Оскільки все більше компаній і окремих людей шукають способи скоротити свої IT-витрати та підвищити свою гнучкість, хмарні обчислення, ймовірно, стануть ще більш популярними.

До хмарних обчислень корпоративні технології складалися здебільшого з локального обладнання та програмного забезпечення. Підприємства встановлювали та обслуговували фізичні сервери у власних центрах обробки даних, розміщували додатки та зберігали дані на цих серверах. Такий підхід був дорого вартісним і вимагав значного технічного досвіду для управління. Це також ускладнювало масштабування тому, що підприємствам доводилося інвестувати в апаратне забезпечення, оскільки їхні потреби зростали. Ця парадигма також зробила інновації неймовірно бюджетно-затратним [1].

Корпоративний центр обробки даних був основою бізнес-технологій до хмари. Тут містилося все локальне обладнання компанії, включаючи сервери,

системи зберігання та мережеве обладнання. Центри обробки даних були дорого коштовними у їх створенні та обслуговуванні та для безперебійної роботи вони вимагали значного технічного досвіду. Крім того, центри обробки даних часто були не дуже гнучкими, що ускладнювало для компаній збільшення чи зменшення масштабів у міру зміни потреб.

На початку 2000-х почав з’являтися новий тип обчислень: хмарні обчислення. Завдяки хмарним обчисленням підприємства могли отримувати доступ до своїх програм і даних через Інтернет, усуваючи потребу в локальному апаратному забезпеченні. Ця нова модель була набагато гнучкішою, ніж традиційний центр обробки даних, завдяки чому компаніям було простіше збільшувати чи зменшувати свої IT-ресурси за потреби. Крім того, хмарні обчислення були економічно ефективнішими, ніж локальні рішення, оскільки компаніям доводилося платити лише за використовувані ресурси [2].

У 2006 році Amazon Web Services (AWS) стала першим великим постачальником хмарних обчислень. AWS запропонувала розрахункову модель, що дозволяла компаніям платити лише за використані ними ресурси. AWS швидко набула популярності, а в 2009 році були запущені Microsoft Azure і Google Cloud Platform (GCP). Ці два провайдери пішли кроками AWS, пропонуючи власні послуги хмарних обчислень з оплатою за використання [3].

Існує чотири основні типи хмарних обчислювальних середовищ: приватні, публічні, гібридні та спільноти.

Приватна хмара — це хмарне середовище, яким володіє та керує одна організація. Приватні хмари пропонують підвищену безпеку та конфіденційність, оскільки лише авторизовані користувачі можуть отримати доступ до даних і програм [4].

Приватна хмара — це обчислювальна мережа, що складається з кількох фізичних серверів, не підключених до загальнодоступного Інтернету. Приватні хмари можуть запропонувати багато переваг порівняно з традиційними розгортаннями серверів, включаючи підвищену безпеку, покращену продуктивність та більшу гнучкість [4].

Однією з ключових переваг приватної хмари є покращена безпека. Зберігаючи конфіденційні дані та програми поза межами загальнодоступного Інтернету, компанії можуть зменшити свій ризик хакерства та інших кіберзагроз.

Крім того, приватні хмари можуть запропонувати кращу продуктивність, ніж публічні хмари, оскільки вони не схильні до тих самих проблем із перевантаженням і затримкою.

Нарешті, підприємства можуть мати більший контроль над своїм середовищем у приватній хмарі, оскільки вони не залежать від одного постачальника послуг. У результаті, впровадження приватної хмари стало ідеальним рішенням для дуже великих організацій, зокрема фінансових установ, федеральних урядів і великих телекомунікаційних компаній, що мають особливі вимоги до безпеки, продуктивності або контролю.

Загальнодоступна хмара – це хмарне середовище, яким володіє та керує постачальник публічних хмар. Загальнодоступні хмари пропонують підвищену гнучкість і масштабованість, оскільки компаніям доводиться платити лише за ті ресурси, які ними використовуються [4].

Публічна хмара – це тип хмарних обчислень, що надає послуги через загальнодоступний Інтернет. Загальнодоступні хмари належать і керуються сторонніми компаніями, які надають доступ до своїх ресурсів, включаючи програми та сховище, через Інтернет [4].

Компанії використовують публічні хмари, щоб заощадити на витратах, пов’язаних зі створенням і підтримкою власної інфраструктури. Публічні хмари також використовуються для масштабованості, оскільки організації можуть швидко додавати або видаляти ресурси за потреби. Крім того, загальнодоступні хмари пропонують високу доступність та безпеку, оскільки зазвичай ними керують досвідчені професіонали.

Хоча загальнодоступні хмари пропонують багато переваг, вони також пов’язані з деякими ризиками, наприклад, витік даних та збої в роботі хмари.

Найпоширенішими причинами витоку даних у загальнодоступних хмарних середовищах є зловмисні або злочинні атаки (58%), потім системні збої

(36%) і помилки людини (34%). Деякі помітні порушення безпеки даних, що сталися в загальнодоступних хмарних середовищах, включають [5]:

- Equifax: у 2017 році агентство кредитної звітності Equifax зазнало витоку даних, у результаті чого розкрито особисту інформацію 145 мільйонів людей. Порушення було спричинене недоліком безпеки в програмі, що була розміщена на публічній хмарній платформі.
- Yahoo: у 2013 році інтернет-компанія Yahoo постраждала від витоку даних, у результаті чого була розкрита особиста інформація 3 мільярдів людей. Порушення зумовлене недоліком безпеки в програмі, яку розмістили на публічній хмарній платформі.
- Dropbox: у 2012 році служба обміну файлами Dropbox зазнала витоку даних, у результаті було розкрито особисту інформацію 68 мільйонів людей. Порушення виявлене через недолік безпеки в програмі, що була розміщена на публічній хмарній платформі.

Хоча витоки даних, безперечно, викликають чимале занепокоєння для компаній, які використовують публічні хмари, але збої також є ризиком.

У сучасному світі підприємства керуються постійним підключенням для кінцевих користувачів, безперервність бізнесу може бути ключовим фактором прийняття рішень. Збої можуть виникати з різних причин, включаючи збої апаратного забезпечення, помилки програмного забезпечення та помилки людини.

Гібридна хмара — це середовище хмарних обчислень, що поєднує в собі приватні та публічні хмари. Гібридні хмари пропонують найкраще з обох хмар, забезпечуючи підприємствам безпеку та конфіденційність приватної хмари, а також гнучкість і масштабованість публічної хмари [4].

Основна перевага гібридної хмари полягає в тому, що вона набагато гнучкіша, ніж традиційна публічна або приватна хмара. За допомогою гібриду кінцевий користувач можете вибрати, які програми та дані зберігати в загальнодоступній хмарі, а які зберігати в приватній хмарі. Це дає можливість пристосувати хмарне рішення до конкретних потреб.

Крім того, гібридна хмара також може забезпечити більшу безпеку та надійність, ніж, окремо використана, публічна чи приватна хмара. Використовуючи поєднання локальних і зовнішніх ресурсів, є можливість створити більш надійну та резервну систему, де менша ймовірність простоїв. Зрештою, гібрид забезпечує вищий рівень гнучкості, масштабованості та безпеки, ніж публічна чи приватна хмара.

Гібридна хмара поєднує в собі локальну інфраструктуру або приватну хмару з публічною хмарою. Гібридні хмари дозволяють організаціям зберігати одні дані та програми на місці, використовуючи еластичність загальнодоступної хмари для інших даних та програм.

Основним недоліком гібридних хмар є те, що ними може бути складніше керувати, ніж публічними чи приватними хмарами, оскільки вони включають два окремих IT-середовища. Крім того, гібридні хмари можуть бути дорожчими, ніж публічні чи приватні хмари, оскільки організації повинні платити і за локальну інфраструктуру, і за публічні хмарні послуги. Іншим недоліком гібридних хмар є те, що вони можуть бути менш безпечними, ніж приватні хмари, оскільки дані та програми зберігаються і на локальних серверах, і в публічній хмарі.

Хмара спільноти — це середовище хмарних обчислень, яким володіє та керує група організацій. Хмари спільноти забезпечують підвищену безпеку та конфіденційність, оскільки лише авторизовані користувачі можуть отримати доступ до даних і програм. Крім того, хмари спільноти пропонують додаткову перевагу економії масштабу [4].

Є багато прикладів хмар спільноти: від навчальних закладів до державних установ або компаній тієї ж галузі. Наприклад, хмарою спільноти Google під назвою Google Apps for Education користуються мільйони студентів і викладачів у всьому світі. Іншим прикладом є Amazon Web Services (AWS) GovCloud, що допомагає урядовим установам відповідати суворим вимогам безпеки [6].

Ці хмари спільноти можуть запропонувати багато переваг організаціям, що робить їх привабливими варіантами для багатьох компаній та установ, які

можуть не мати власних талантів або можливостей створювати та керувати публічними чи приватними хмарними ресурсами.

Існує чотири основні типи хмарних служб: інфраструктура як послуга (IaaS), платформа як послуга (PaaS), безсерверне обчислення та програмне забезпечення як послуга (SaaS). Кожною з цих послуг можуть користуватися клієнти будь-якого мережевого рівня різних розмірів [7].

IaaS стосується надання віртуалізованих обчислювальних ресурсів, включаючи сховище, мережі та сервери. IaaS є масштабованим і зазвичай пропонується у вигляді платної послуги, де оплата здійснюється лише за ресурси, що використовуються. Постачальники IaaS, зазвичай, пропонують користувачам портал самообслуговування для керування своїми ресурсами, а також API для автоматизації. IaaS можна використовувати для розміщення програм, веб-сайтів і баз даних. Сервіси IaaS часто застосовуються для надання звичайного веб-хостингу, але цю модель можна використовувати для широкого спектру цілей. Розробники, які використовують модель IaaS, як правило, потребуватимуть ресурсів інфраструктури програмного забезпечення для розгортання та підтримки своїх можливостей хмарних обчислень. IaaS може забезпечити економічно ефективний спосіб побудови та масштабування інфраструктури [8].

Платформа як послуга, або PaaS, — це модель хмарних обчислень, що надає розробникам платформу для створення, тестування та розгортання програм. Середовище PaaS, зазвичай, має все, що необхідно для підтримки розробки додатків, включаючи операційну систему, інструменти, проміжне програмне забезпечення та інфраструктуру. Це дозволяє розробникам зосередитися на своєму коді, а не турбуватися про керування основною обчислювальною інфраструктурою. Постачальники PaaS пропонують різноманітні послуги, зокрема зберігання, мережу та обчислювальну потужність. Це дозволяє легко масштабувати додатки за потреби. PaaS часто використовується для розробки веб- і мобільних додатків, а також для великих даних. Його можна розгорнути локально або в хмарі. PaaS має багато переваг для

розробників. Це може допомогти заощадити час і гроші, спрощуючи процес розробки програми. Крім того, PaaS дозволяє легко розгортати програми на кількох платформах. Це робить його гарним вибором для організацій, яким потрібна гнучкість для запуску своїх програм локально або в хмарі [8].

SaaS (Software as a Service) — це модель поставки програмного забезпечення, де застосунки доступні користувачам через інтернет на основі підписки. Замість традиційного підходу до покупки та встановлення програмного забезпечення на індивідуальні комп'ютери або сервери, користувачі отримують доступ до застосунків віддалено через хмару. Постачальники SaaS забезпечують централізовані оновлення, тому всі користувачі отримують доступ до останніх версій застосунків без додаткових витрат часу або грошей. Застосунки доступні з будь-якого пристрою, що має доступ до інтернету та сучасний веб-браузер. Постачальники можуть надавати додаткові ресурси без необхідності витрат на придбання та обслуговування нового обладнання. Сьогодні SaaS-застосунки охоплюють широкий спектр бізнес-завдань: CRM-системи (наприклад, Salesforce), офісні застосунки (Google Workspace, Microsoft 365), облікові системи (QuickBooks) та інші. З ростом хмарних технологій очікується, що SaaS продовжить свою еволюцію. Інтеграція штучного інтелекту, розширена реальність, блокчейн та інші передові технології можуть зробити SaaS ще більш адаптивним та потужним інструментом для бізнесу. Технологія SaaS революціонізує спосіб, за допомогою якого організації отримують та використовують програмне забезпечення. Вона пропонує швидкість, гнучкість, та економію, при цьому викликаючи нові питання щодо безпеки, управління та інтеграції. Попри виклики, SaaS залишається однією з ключових технологій для сучасних організацій, яка буде продовжувати впливати на ландшафт IT у майбутньому [8].

Безсерверне обчислення (Serverless) — це підхід до розробки та експлуатації програмного забезпечення, де розробники фокусуються на написанні коду, не турбуючись на чому він виконується, тобто, про підготовку, налаштування та управління серверами. Це можливо завдяки тому, що

обчислення відбуваються у віртуальних контейнерах, які запускаються відповідно до вхідних подій, наприклад, HTTP-запиту. Цей підхід забезпечує автоматизоване масштабування застосунків, оптимізуючи ресурси та витрати. Відмінною особливістю serverless є те, що розробники платять тільки за реально використаний час виконання коду, а не за простій сервера. Це забезпечує високу ефективність вартості для застосунків із непостійним навантаженням. Серед популярних платформ, які пропонують serverless-рішення, можна виокремити AWS Lambda, Azure Functions та Google Cloud Functions. Проте, не дивлячись на численні переваги, зокрема швидке впровадження та відсутність необхідності управління серверною інфраструктурою, serverless також має обмеження, наприклад у відношенні до тривалості виконання завдань, стартової затримки та інтеграції із традиційними системами [8].

Однією з головних переваг хмарних обчислень є те, що вони можуть допомогти зменшити витрати на IT. Замість того, щоб інвестувати та підтримувати власну локальну інфраструктуру, краще використовувати хмарні служби. Це може допомогти зменшити початкові капітальні витрати, а також поточні операційні витрати. Хмарні технології дозволяють швидко збільшувати або зменшувати масштаб за потреби, не роблячи великих початкових інвестицій. Це може допомогти швидко реагувати на зміни попиту. Хмарні обчислення можуть покращити можливості аварійного відновлення. Хмарні обчислення створюють можливість виконувати віддалену роботу кількома різними способами. По-перше, доступ до хмарних програм і служб можна отримати з будь-якого місця, де є підключення до Інтернету. Це означає, що співробітники можуть працювати віддалено, а розробники можуть отримати доступ до даних на хмарному сервері через безпечне підключення до Інтернету. Завдяки можливості переходу між локальною та різними хмарними платформами, компанії можуть уникнути блокування постачальника, що може призвести до зростання витрат. Крім того, за допомогою пропозиції від постачальника хмарних послуг, як-от AWS Cloudfront, компанії можуть керувати витратами на хмару.

Велика трійка хмарних компаній має власні сильні сторони. AWS є найвідомішим гравцем світової хмарної індустрії і стоїть за здатністю Amazon підтримувати значні сезонні коливання попиту з боку споживачів. Завдяки першому виходу на ринок із хмарними послугами та наполегливим зусиллям, щоб завоювати частку ринку, компанія стала лідером ринку та продовжує впроваджувати інновації. Azure від Microsoft став невід'ємною частиною стратегії Microsoft, і компанія має корпоративну історію та продукти для підтримки компаній, що переходять на хмару. Google Cloud є найменшим із великої трійки гравців на ринку хмар, але явно має за собою могутність гіганта реклами для Android [9].

Сьогодні хмарні обчислення пропонують безліч послуг, які можна використовувати для широкого спектру програм, зокрема:

– Хмарне сховище мають такі служби: Amazon Simple Storage Service (S3), Google Cloud Storage та Microsoft Azure Storage, що забезпечують масштабоване зберігання об'єктів у хмарі. Це можна використовувати для зберігання різних даних, включаючи резервні копії, медіа-файли та дані програм [10].

– Хмарні обчислення мають такі служби: Amazon Elastic Compute Cloud (EC2), Google Compute Engine (GCE) і Microsoft Azure Virtual Machines (VM), що надають ресурси обчислювальної потужності в хмарі. Це можна використовувати для розміщення веб-додатків, виконання пакетних завдань або зберігання даних [10].

– Хмарні бази даних мають такі служби: Amazon DynamoDB, Google Cloud Datastore і Microsoft Azure SQL Database, що надають керовані реляційні та нереляційні бази даних у хмарі. Це можна використовувати для зберігання структурованих і неструктурованих даних, наприклад інформація про клієнтів, дані про продукт і дані про замовлення. Це також призвело до появи низки інших служб для забезпечення візуалізації даних на основі корпоративних даних і хмарних даних [10].

– Хмарну аналітику мають такі служби: Amazon Elastic MapReduce (EMR), Google BigQuery та Microsoft Azure HDInsight, що забезпечують обробку та аналітику даних у хмарі. Це можна використовувати для аналізу даних, щоб отримати уявлення про поведінку клієнтів, тенденції продукту чи інші бізнес-дані [11].

– Хмарну безпеку мають такі служби: Amazon Inspector, Google Cloud Security Scanner і Microsoft Azure Security Center, що забезпечують хмарне сканування та моніторинг безпеки. Це можна використовувати для виявлення та усунення загроз безпеці[10].

Хмарні обчислення досягають точки, коли вони, ймовірно, будуть витісняти традиційні форми доставки додатків і послуг всередині компанії, які існують на ринку десятиліттями. Однак використання хмари, швидше за все, лише зростатиме, оскільки організації почуватимуться більш комфортними з ідеєю, що їхні дані знаходяться десь, а не на сервері в підвалі. І зараз постачальники хмарних обчислень все більше просувають хмарні обчислення, як засіб цифрової трансформації замість того, щоб зосереджуватися просто на вартості. Перехід до хмари може допомогти компаніям переосмислити бізнес-процеси та прискорити бізнес-зміни. Деяким компаніям, які потребують активізації своїх програм цифрової трансформації, цей аргумент може здатися привабливим; інші можуть помітити, що ентузіазм щодо хмари зменшується, оскільки витрати на перехід збільшуються.

1.2 Аналіз хмарного середовища AWS

Amazon Web Services (AWS) — це найбільш широко використовувана у світі хмарна платформа, що є значущою частиною бізнес-портфеля Amazon. Оскільки Amazon.com революціонізував світ он-лайн торгівлі, тому AWS змінив парадигму обчислювальної техніки. Amazon Web Services включає різноманітні продукти та сервіси для хмарних обчислень, зокрема сервери, сховища і мережі. Він став лідером у сфері хмарних обчислень, випереджаючи конкурентів, наприклад Microsoft Azure та Google Cloud Platform [11].

Amazon Web Services (AWS) є підрозділом Amazon.com, пропонує доступні послуги хмарних обчислень, що дають можливість зміцнити клієнтську базу від невеликих компаній, як Pinterest, до великих корпорацій, на приклад D-Link.

AWS надає широкий спектр засобів захисту для робочих навантажень клієнтів. Однак клієнти також несуть певну відповідальність за безпеку своїх ресурсів у хмарі.

AWS пропонує безліч послуг, інструментів і методів для забезпечення безпеки в хмарі, зокрема контроль доступу, брандмауер, шифрування, реєстрація, моніторинг і відповідність. Ці послуги підтримують різні сценарії та вимоги, щоб допомогти клієнтам дотримуватися всіх необхідних заходів безпеки [12].

Деякі з основних послуг та інструментів безпеки AWS включають:

- AWS Identity and Access Management (IAM) для управління доступом до ресурсів AWS;
- Virtual Private Cloud (VPC) для створення віртуальної мережі в хмарі AWS;
- AWS Key Management Service (KMS) для управління ключами шифрування;
- AWS Shield і AWS Web Application Firewall (WAF) для захисту від атак DDoS;
- AWS Config, AWS CloudTrail і AWS CloudWatch для аудиту та управління конфігурацією;
- AWS Artifact для надання документів про відповідність.

Клієнти повинні розуміти свою частку відповідальності за безпеку в хмарі. Вони мають вжити для захисту своїх ресурсів таких заходів:

- Створення міцних паролів і багатофакторної аутентифікації.
- Обмеження доступу до ресурсів AWS.
- Шифрування даних.
- Встановлення брандмауерів.
- Моніторинг активності в хмарі.

Клієнти також вимушені регулярно оглядати свої заходи безпеки, щоб переконатися, що вони відповідають їхнім потребам.

AWS забезпечує глобальну інфраструктуру, що складається з таких компонентів:

- Регіони — це географічні зони, де розташовані центри обробки даних AWS.
- Зони доступності — це окремі центри обробки даних у межах регіону.
- Центри обробки даних — це фізичні об'єкти, де розміщується обладнання, програмне забезпечення, мережі та інші ресурси AWS [12].

AWS має широкий спектр заходів безпеки, що застосовуються для захисту своїх центрів обробки даних. Ці заходи включають:

- Фізичну безпеку: центри обробки даних AWS захищені від несанкціонованого доступу за допомогою фізичних бар'єрів, наприклад, огорожі, камери відеоспостереження та персонал охорони.
- Кібербезпеку: центри обробки даних AWS захищені від кіберзагроз за допомогою брандмауерів, систем виявлення вторгнень та інших технологій.
- Безпеку даних: дані клієнтів шифруються на всіх етапах їхнього життєвого циклу, від зберігання до передачі.
- AWS використовує різні типи центрів обробки даних, щоб відповідати потребам різних клієнтів. Наприклад, AWS пропонує центри обробки даних, що оптимізовані для обробки даних, обробки штучного інтелекту та машинного навчання, а також для інших цілей.
- AWS постійно інвестує в свою глобальну інфраструктуру, щоб забезпечити її безпеку та надійність.

AWS прагне до відповідності найвищим стандартам безпеки. Він є сертифікованим за ISO 27001, ISO 27017 та ISO 27018, а також за іншими стандартами безпеки. AWS бере на себе зобов'язання забезпечити безпеку та відповідність своїх центрів обробки даних [13].

AWS має вражаючий досвід у керуванні та захисті великих центрів обробки даних у всьому світі, завдяки підтримці материнської компанії Amazon. Усі ці центри обробки даних оптимізовані за допомогою передових технологій та процесів, включаючи рішення, що гарантують фізичну безпеку, захист від несанкціонованого доступу та багато інших.

Однією з ключових складових безпеки є обмеження доступу персоналу до поверхонь центрів обробки даних, якщо це не є необхідним. AWS ретельно контролює, хто отримує доступ до фізичних пристроїв зберігання даних, щоб гарантувати їхню безпеку та недоступність для недозволених осіб. Додатково, AWS встановлює сегрегацію відповідальності, яка означає, що навіть якщо особа має доступ до фізичного пристрою, вона не матиме повних прав для управління ним або доступу до конфіденційної інформації.

AWS пильно дбає про безпеку і фізичного, і логічного аспекту функціонування своїх центрів обробки даних. Це включає в себе моніторинг, захист від крадіжок, захист від вторгнень, запобігання пожежам, обережні заходи під час природних катастроф та надійне електроживлення. Все це робиться на користь клієнтів, щоб вони мали впевненість в безпеці своїх даних.

Рисунок 1.1 – Регіони розподілення ресурсів AWS

Amazon Web Services зберігає дані та ресурси у різних географічних регіонах у всьому світі, відомих як регіони (див. Рис. 1.). Кожен регіон має дві

або більше зони доступності, що забезпечує високу стійкість до неполадок. Кожна зона доступності складається з одного або декількох центрів обробки даних. Важливо зазначити, що всі ці центри обробки даних активні в роботі та не існує "холодних" центрів обробки даних, що були б офлайн. Дані центри обробки даних для забезпечення надійності та продуктивності всіх послуг AWS містять всі необхідні фізичні та апаратні ресурси, зокрема, сервери, сховища та мережеві пристрої. Основні послуги AWS, а саме: обчислення, зберігання, бази даних та мережі, розгортаються у конфігурації N + 1, це означає, що є достатня потужність для автоматизованого перерозподілу навантаження, у випадку виникнення проблем в одному із центрів обробки даних через природні катастрофи, помилки або інші непередбачувані обставини. Кожна зона доступності розглядається окремо, у вигляді незалежної одиниці, що дозволяє враховувати можливі відмови в роботі. Вони фізично розташовані окремо в межах географічного регіону та розташовані у високих місцевостях. Залежно від потреб бізнесу, вимог щодо дотримання нормативних вимог, аварійного відновлення та стійкості до відмов, потрібно розглядати розробку таким чином, щоб ресурси були розподілені між кількома регіонами, щоб забезпечити доступність, навіть якщо один із регіонів стане недоступним [14].

Цей хмарний постачальник має численні переваги, що роблять його особливим серед конкурентів, які набирають популярність [15].

1. Безпека та надійність: AWS має серверні ресурси, розташовані у 76 зонах доступності, що охоплюють 245 країн і територій. Ці зони були створені не лише для того, щоб дати користувачам можливість встановлювати географічні обмеження для своїх послуг, але й для різноманітності фізичного розташування. Це забезпечує більш високий рівень безпеки та унікальний підхід до запобігання втраті даних у всьому світі. Кожен центр обробки даних по всьому світу підтримується та постійно контролюється. AWS дозволяє отримувати доступ до фізичних центрів обробки даних тільки за потреби, що створює додатковий бар'єр для фізичних вторгнень.

2. Захист від відключень і атак: AWS вкладає значні зусилля в забезпеченні безпеки своїх послуг. Це включає неперервний інтенсивний моніторинг, що дозволяє швидко виявляти та реагувати на потенційні атаки. Такий підхід робить сервери менш вразливими перед відключеннями та атаками.
3. Масштабованість — це одна з ключових переваг, її можуть використовувати початківці та малі підприємства, користуючись послугами Amazon Web Services. AWS пропонує гнучку ціноутворену політику, де вартість послуг змінюється відповідно до обсягу використання. Це робить AWS привабливим вибором для тих, хто планує розвивати свій бізнес. Для стартапів AWS надає всі необхідні інструменти для побудови хмарного бізнесу "знизу вгору". В той же час, вже існуючі компанії можуть скористатися послугами міграції Amazon, що допоможуть безперешкодно перенести свою інфраструктуру на AWS. З приводу розширення бізнесу, то AWS постійно надає додаткові ресурси, пристосовуючись до росту підприємства. Масштабованість AWS дає компаніям можливість працювати, не турбуючись питаннями про бюджет для інфраструктури.
4. Вартість. Раніше компанії стикалися з великими витратами та складністю при створенні та управлінні власними інфраструктурами для зберігання та обробки великих обсягів даних. Вибір між фізичним зберіганням та зберіганням в хмарі вимагав великих фінансових витрат на довгострокові контракти, що могли б обмежити ріст компанії. Для багатьох компаній виникали труднощі при оцінці потреб у зберіганні даних та обчислювальній потужності. Купівля надмірної кількості ресурсів може призвести до надмірних витрат, поряд з тим, недостатньо ресурсів може призвести до недостатньої продуктивності та проблем під час надмірного навантаження. Витрати на електроенергію також можуть бути значними, оскільки бізнеси повинні підтримувати достатню кількість енергії для високого навантаження, навіть, коли вони не використовують всі ресурси. AWS вирішує ці проблеми, дозволяючи компаніям платити лише за те, що

вони фактично використовують. Компанії більше не зобов'язані інвестувати в початкові витрати на інфраструктуру та зберігання даних, вони можуть автоматично масштабувати свої витрати, відповідно до змін потреб. Це дозволяє компаніям ефективно використовувати свої ресурси та зменшує витрати на утримання інфраструктури.

Після аналізу основної інформації, що стосується хмарного провайдера AWS та ретельного вивчення документації та практичних аспектів використання цієї платформи, було визначено основні переваги даного сервісу порівняно з іншими. Однією з найсильніших переваг Amazon Web Services є їхні рішення в галузі кібербезпеки та забезпечення безпеки в цілому.

1.3 Формалізація вимог та постановка задачі

На сьогодні, хмарні технології активно інтегруються в різні сфери життя, від простих додатків до критично важливих систем. Проте, з ростом популярності хмарних рішень зростає і ризик безпеки, особливо для застосунків, що працюють у такому середовищі. Автоматизована система керування безпекою є ключовим компонентом для забезпечення конфіденційності, цілісності та доступності даних у хмарі.

Мета: автоматизації процесів розгортання та налаштування сервісів безпеки, завдяки розробці підсистеми керування безпекою, здатної гарантувати застосункам у хмарному середовищі високий рівень захищеності, враховуючи специфіку та особливості хмарних платформ..

Основні завдання:

1. **Аналіз хмарного середовища AWS**:
    - Дослідження основних сервісів AWS: Lambda, API Gateway, S3, Cognito, IAM.
    - Оцінка можливостей і обмежень кожного сервісу з точки зору безпеки.
2. **Проектування підсистеми безпеки**:
    - Визначення ключових параметрів безпеки для serverless застосунків.

- Розробка архітектури підсистеми з урахуванням особливостей AWS.
- Визначення політик безпеки для IAM.

3. **Реалізація підсистеми**:
    - Розробка демонстраційного застосунку.
    - Інтеграція підсистеми безпеки з демонстраційним застосунком.
4. **Документація та висновки**:
    - Складання інструкцій та рекомендацій з використання підсистеми.

Для розробки підсистеми автоматизованого керування безпекою застосунків у хмарному середовищі AWS, додатково до визначених завдань, необхідно врахувати наступні вимоги:

1. **Аналіз хмарного середовища AWS:**
    - Доступ до AWS з необхідними правами для дослідження сервісів.
    - Інструменти для аналізу безпеки (наприклад, сканери вразливостей).
2. **Проектування підсистеми безпеки:**
    - Система керування проектами.
    - Моделювання архітектури.
3. **Реалізація підсистеми:**
    - Доступ до середовища розробки та тестування на AWS.
    - Інструменти для написання коду, контролю версій та автоматизації розгортання.
4. **Документація та висновки:**
    - Інструменти для документування (наприклад, системи управління контентом).
    - Канали зворотного зв'язку для збору пропозицій та вдосконалень.

2 РОЗРОБКА СТРУКТУРИ ЗАСОБУ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Огляд функцій безпеки, що надаються Amazon Web Services

Amazon Web Services (AWS) пропонує широкий спектр механізмів контролю для захисту користувацьких робочих навантажень. Проте, багато клієнтів не повністю усвідомлюють свої обов'язки щодо забезпечення безпеки і необхідність впровадження контрольних заходів для безпечного використання і розміщення своїх даних в хмарі. AWS пропонує різноманітні послуги і інструменти, включаючи управління доступом, брандмауери, шифрування, логування, моніторинг і відповідність стандартам, що допомагають забезпечувати безпеку в хмарі. Ці сервіси враховують різноманітні випадки використання та сценарії, це дозволяє відповідати усім вимогам безпеки, аудиту та регулювання в хмарному середовищі [16].

Зокрема, AWS Identity and Access Management (IAM) дозволяє керувати безпечним доступом і операціями користувачів AWS. Virtual Private Cloud (VPC) надає можливість захисту інфраструктури в хмарі AWS, створюючи віртуальну мережу, що є продовженням приватної мережі компанії. AWS також пропонує Key Management Service (KMS) для керування шифрувальними ключами і додаткового захисту даних. AWS Shield і AWS Web Application Firewall (WAF) захищають ресурси AWS від поширених загроз, наприклад, від DDoS атак, дозволяючи налаштувати брандмауери на різних рівнях [17].

AWS Config, AWS CloudTrail і AWS CloudWatch сприяють аудиту та управлінню конфігурацією ресурсів AWS. Загальний розвиток хмарних технологій показує, що вони стають все більш прийнятними для організацій, завдяки їхній безпеці, яка часто перевищує можливості традиційних IT інфраструктур і є більш економічно вигідною [17].

Лідери галузі прогнозують, що хмарні обчислення продовжуватимуть стрімкий розвиток. International Data Corporation (IDC) вважає, що хмарні технології мають найбільший потенціал розвитку . AWS забезпечує одне з найбезпечніших і найбільш гнучких хмарних середовищ, мінімізуючи багато

традиційних безпекових навантажень, пов'язаних з IT-інфраструктурою і забезпечуючи конфіденційність та безпеку на високому рівні.

Захист інформації у хмарному середовищі відрізняється від захисту в центрах обробки даних, що знаходяться на власних серверах компанії. При перенесенні серверів, даних та навантажень в хмарне середовище AWS, виникає поділ обов'язків між користувачем та постачальником послуг. AWS забезпечує інфраструктурну надійність, включаючи фізичну безпеку своїх дата-центрів, в той час, як клієнти відповідають за безпеку того, що вони розгортають в хмарі. Клієнти також відповідають за управління доступом до своїх віртуальних мереж та захистом своїх хмарних активів. Цей поділ відповідальності відомий як модель спільної відповідальності AWS у сфері безпеки [18]. На рисунку 2.1 показана така модель відповідальності сторін:

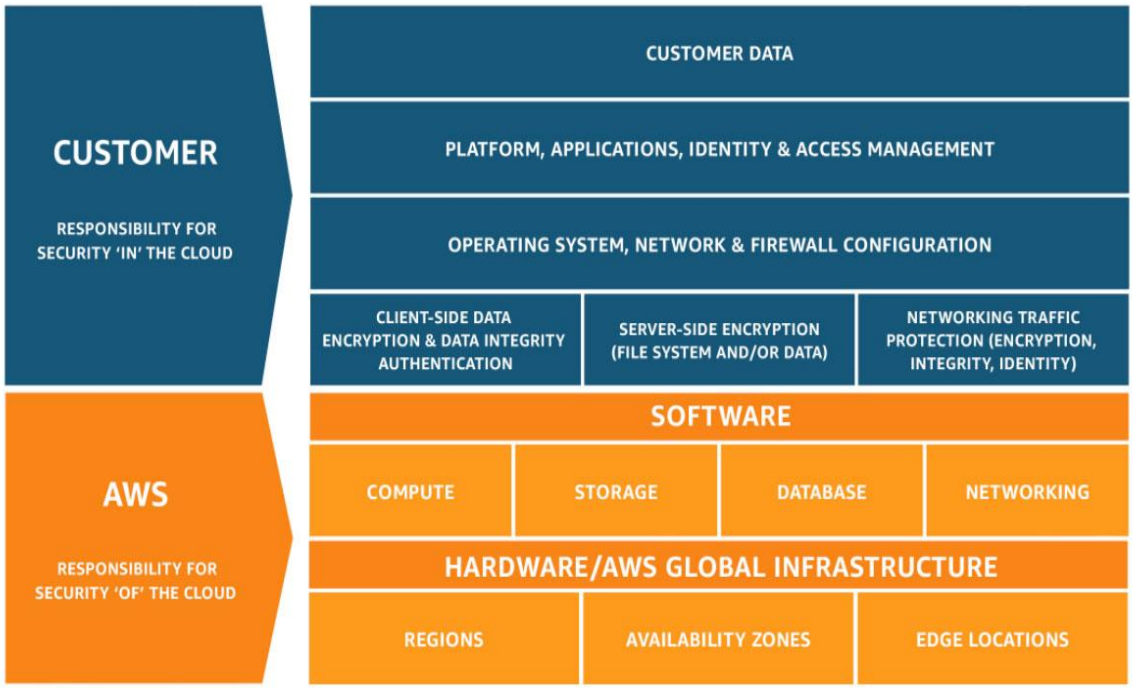


Рисунок 2.1 – Модель спільної відповідальності AWS

Для глибокого розуміння безпеки в сервісах AWS, критично важливо осмислити як розподіляється відповідальність між AWS і користувачем. AWS надає широкий спектр сервісів, які можна класифікувати в три основні групи – інфраструктурні, контейнеризовані та високорівневі сервіси, кожна з яких має

свої особливості управління безпекою, залежно від того, як користувачі взаємодіють з ними і як вони налаштовують доступ до своїх функцій:

Інфраструктурні сервіси. Вони включають обчислювальні можливості, як-от Amazon EC2, і пов'язані з ними послуги, а саме EBS та VPC. Ці сервіси дозволяють створювати безпечні мережеві структури в хмарі, що можуть інтегруватися з існуючими локальними системами. Користувачі самі керують операційними системами, налаштуваннями мережевої безпеки та системами ідентифікації користувачів.

Контейнеризовані сервіси. AWS надає керовані сервіси в ізольованих контейнерах, а саме AWS Elastic Beanstalk чи Amazon RDS. Користувач відповідає за конфігурацію мережевої безпеки та управління доступом через IAM для забезпечення безпечного використання цих сервісів.

Високорівневі сервіси. Ці сервіси, на прикладі, Amazon S3 чи DynamoDB, відокремлюють користувачів від управління платформою, пропонуючи доступ до функціоналу через API. AWS керує основною операційною системою, і користувачі використовують надану інфраструктуру, залишаючись відповідальними за захист своїх даних через інтеграцію з IAM.

Знання цих категорій та їх особливостей допомагає користувачам AWS ефективно розподілити обов'язки безпеки та забезпечити надійний захист своїх хмарних ресурсів.

Концепція розподіленої відповідальності в AWS, також, стосується контейнеризованих сервісів, зокрема Amazon EMR чи Amazon RDS. У випадку цих сервісів, AWS бере на себе управління операційною системою, основною інфраструктурою, програмною платформою та базовими компонентами. Якщо взяти до уваги Amazon RDS для Microsoft SQL Server, то це сервіс баз даних, де всі рівні контейнера, включаючи платформу бази даних, належать до сфери відповідальності AWS.

Хоч AWS надає інструменти для резервного копіювання та відновлення для сервісів на зразок Amazon RDS, користувачі самі мають забезпечувати

планування та реалізацію стратегій високої доступності, стійкості до збоїв, неперервності бізнес-процесів та планів відновлення після аварій.

Користувачі, також, несуть відповідальність за захист своїх даних, управління доступом до цих даних та налаштування мережевих політик безпеки для захисту контейнеризованих сервісів. Це включає налаштування груп безпеки для Amazon RDS та EC2, що регулюють доступ до Amazon RDS та EMR відповідно. На рисунку 2.2 показана модель такої відповідальності для контейнерних послуг:

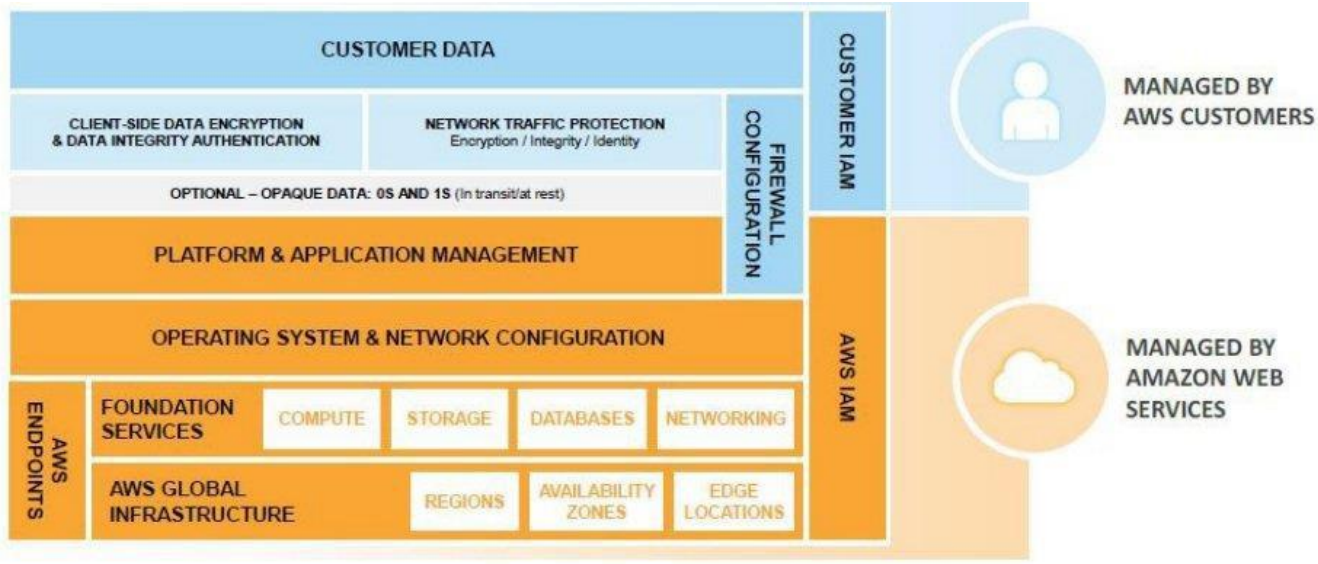


Рисунок 2.2 – Модель спільної відповідальності для контейнерних послуг

AWS надає набір високоабстрактних сервісів, включно з Amazon DynamoDB, Amazon Simple Queue Service та Amazon S3, що дозволяють користувачам зберігати, оновлювати та отримувати дані через визначені API. Відповідальність AWS обмежується управлінням та підтримкою цих сервісів на інфраструктурному рівні, включаючи налаштування та оновлення операційних систем, а також управління платформами. Ці сервіси є інтегрованими з AWS Identity and Access Management (IAM), надаючи користувачам контроль над доступом до своїх даних.

Користувачі відповідають за класифікацію своїх даних і використання інструментів IAM для налаштування детальних дозволів доступу до ресурсів. За допомогою IAM, можна налаштовувати дозволи, ґрунтуючись на ролях, ідентифікації користувачів чи користувацьких групах. Крім того, Amazon S3

пропонує вбудоване шифрування для захисту даних у стані спокою, а HTTPS інкапсуляція використовується для захисту даних під час передачі.

Додатково, AWS надає можливості оптимізації вартості та продуктивності для цих сервісів. Наприклад, з Amazon S3 можна використовувати політики життєвого циклу, щоб автоматично переміщати дані між різними класами сховища відповідно до використання, що може суттєво знизити вартість зберігання. Також, засоби моніторингу та аналітики від AWS дозволяють відстежувати і аналізувати використання даних для кращого розуміння моделей доступу та оптимізації ресурсів. Для поліпшення безпеки, користувачі можуть використовувати AWS Key Management Service (KMS) для управління шифрувальними ключами, а, також, впроваджувати додаткові заходи безпеки, зокрема, багатофакторну аутентифікацію для посилення захисту своїх акаунтів та відповідних даних.

Звернемо увагу на захисні сервіси, що пропонує AWS. Основна мета цих сервісів полягає у забезпеченні інструментів для оборони активів усередині екосистеми AWS.

Управління ідентифікацією та доступом AWS на основі інструментарію з ключовими компонентами та функціями IAM.

AWS Identity and Access Management (IAM) — це комплексний інструментарій для управління ідентифікацією та доступом в середовищі AWS. Це включає в себе декілька ключових компонентів і функцій [19]:

1. Користувачі та групи.

- Для користувачів IAM дозволяє встановлювати політики безпеки на індивідуальному рівні, що дає змогу призначати унікальні набори дозволів для кожного з них.
- Створення групи – це зручний спосіб управління дозволами для багатьох користувачів одночасно, що спрощує адміністрування, коли однаковий рівень доступу потрібен для цілої команди або відділу.

2. Дозволи та політики.

- Кожен дозвіл детально визначає, які дії може виконувати користувач, забезпечуючи можливість створення комплексної матриці безпеки.
- Політики можуть бути візуалізовані та керуватися через AWS Management Console, що дає можливість легкого перегляду та редагування правил доступу (приклад на рис. 2.3).

```
MySQSQueue:
  Type: 'AWS:SQS::Queue'

MyRole:
  Type: 'AWS::IAM::Role'
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
          Service: 'ec2.amazonaws.com'
          Action: 'sts:AssumeRole'
    Policies:
      - PolicyName: root
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action: 'sqs:SendMessage'
              Resource: !Sub ${MySQSQueue.Arn}
```

Рисунок 2.3 – Приклад AWS IAM Policy

3. Ролі.

Ролі можна призначати не тільки живим користувачам, а й сервісам AWS. Це, в свою чергу, дозволяє сервісам виконувати дії в інших сервісах з відповідними дозволами без необхідності використання статичних ключів доступу.

4. Багатофакторна аутентифікація (MFA).

Завдяки MFA, навіть якщо облікові дані користувача скомпрометовані, додатковий рівень перевірки ускладнює несанкціонований доступ.

5. Деталізований контроль доступу.

Використання деталізованих політик IAM для управління доступом до специфічних API операцій дозволяє застосовувати принцип найменшого привілею, мінімізуючи можливість зловживань.

6. Інтеграція з сервісами AWS.

Через тісну інтеграцію з AWS сервісами, IAM забезпечує однакову модель безпеки через всі компоненти AWS, дозволяючи централізовано управляти доступом до ресурсів.

7. Безпечність та відповідність.

IAM сприяє досягненню та підтриманню відповідності регулятивним та галузевим стандартам, що є критично важливим для компаній різних розмірів.

9. Порадник доступу.

Функція Access Advisor допомагає ідентифікувати потенційні ризики безпеки, вказуючи на сервіси, що не використовуються та можуть бути обмежені.

10. Аналізатор доступу.

Access Analyzer допомагає виявити та усунути ненавмисні розголошення ресурсів, забезпечуючи кращу видимість конфігурацій доступу.

11. Звіт про облікові дані IAM.

Звіт про облікові дані IAM надає повний аудит облікових записів, сприяючи поліпшенню управління обліковими записами та політиками безпеки.

Всі основні компоненти AWS IAM зображені на рис.2.4.

Загалом, IAM є критично важливою складовою AWS, яка надає необхідні інструменти для управління доступом, зміцнення безпеки та забезпечення відповідності у середовищі AWS. Він надає гнучкість у контролі того, “Як?”, “Коли?” і “Де?” користувачі взаємодіють з ресурсами AWS, забезпечуючи міцний фреймворк для безпечного управління цифровими ідентичностями та дозволами.

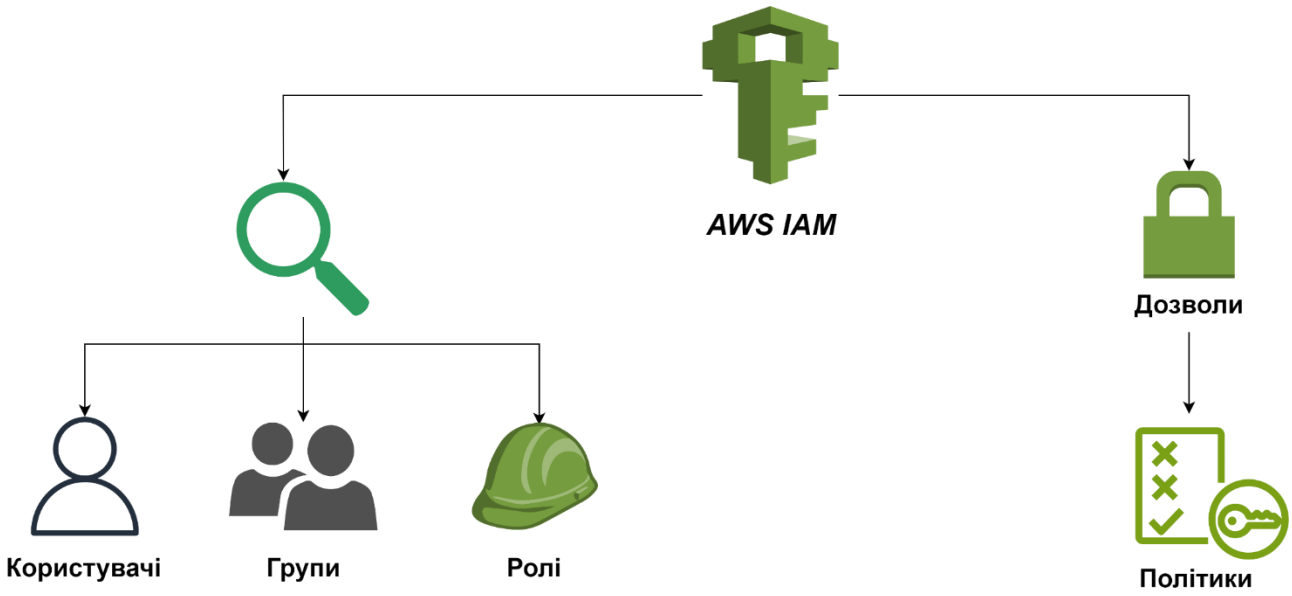


Рисунок 2.4 – Основні компоненти AWS IAM

Віртуальна приватна мережа (VPC), основні компоненти служби AWS VPC зображені на діаграмі рис. 2.5).

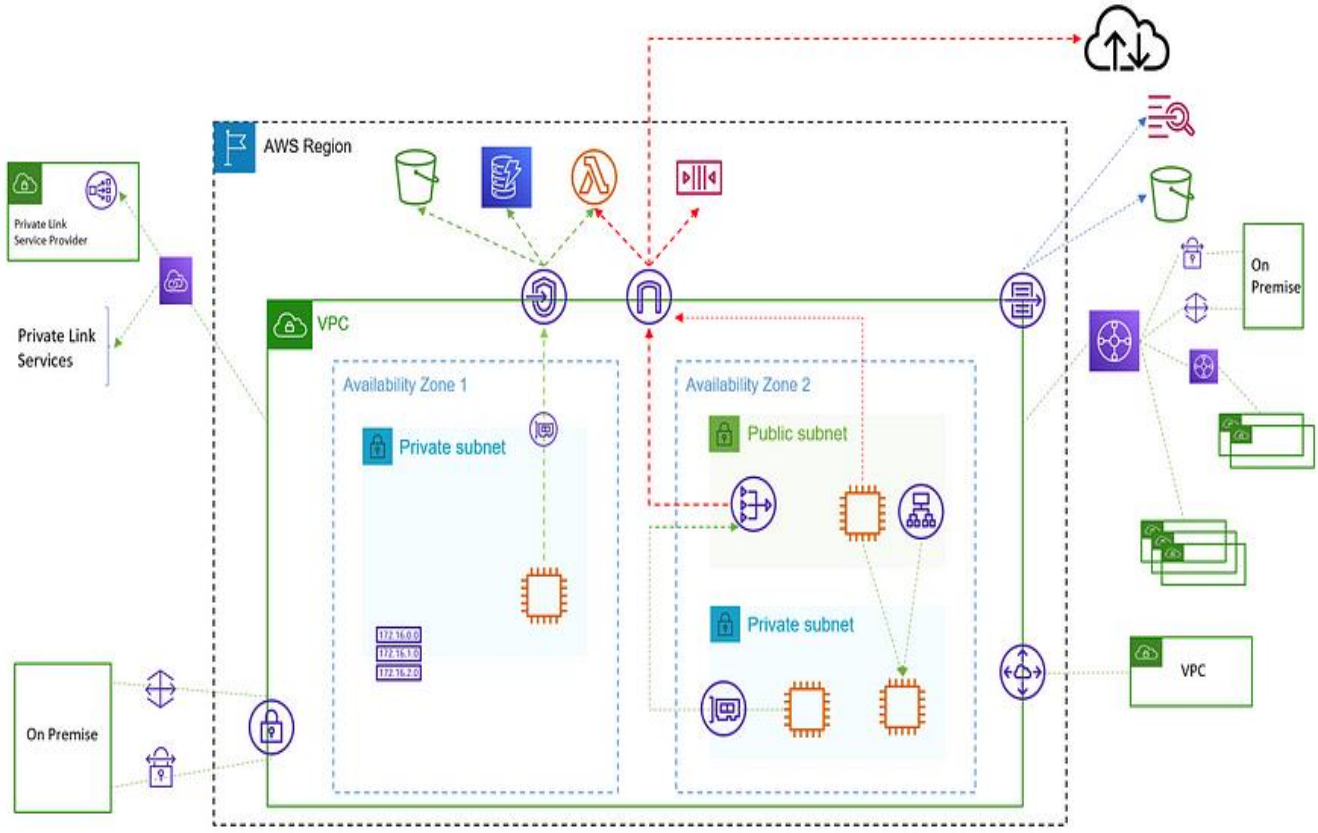


Рисунок 2.5 – Компоненти AWS VPC

Розглянемо опис і характеристики компонентів служби AWS VPC детальніше.

Віртуальна приватна хмара — це послуга від AWS, що дозволяє запускати ресурси AWS у логічно ізольованій віртуальній мережі (схожій на локальний центр обробки даних). Це дає повний контроль над віртуальним мережевим середовищем, включаючи розміщення ресурсів, підключення і безпеку. Це регіональна служба. За замовчуванням кожен регіон у обліковому записі матиме стандартний VPC, який можна швидко запустити [20].

Коли сервіс запускається у публічній хмарі, він стає відкритим для усього світу, що може створювати ризик атак з Інтернету. Для захисту сервісів від зовнішніх загроз їх поміщають у VPC, що обмежує типи трафіку, IP-адреси та користувачів, які мають доступ до сервісів. Це допомагає уникнути небажаного доступу до ресурсів і захищає від таких проблем, як DDOS-атаки. Не всі сервіси підключені до Інтернету, тому їх можна безпечно ізолювати у приватній мережі, відкриваючи доступ до Інтернету тільки для певних машин. Якщо виникає необхідність встановлення програмного забезпечення чи доступу до Інтернету з інстанцій, що ізольовані від мережі, то може з’явитися проблема. Проте, рішення для цього існують, вони розглянуті далі в тексті.

NAT може вирішити проблему інсталяції програм з Інтернету на захищені приватні сервіси. NAT створюється у публічній підмережі з доступом до Інтернету. Якщо дозволити доступ з приватної інстанції до NAT, то приватна інстанція зможе надсилати запити до Інтернету. Цей доступ є одностороннім, тобто ніхто з Інтернету не може отримати доступ до інстанції [21].

AWS Cognito — сервіс Amazon Cognito, створений для надання API та інфраструктури для ключових функцій у сфері управління користувачами, а саме, аутентифікація, авторизація та управління репозиторієм користувачів із різними операціями для веб- та мобільних додатків. Amazon Cognito надає важливі функції для реалізації різних сценаріїв управління користувачами та аутентифікації у веб-застосунках та мобільних додатках. Загалом, будь-які служби керування користувачами та керування ідентифікацією базуються на двох ключових функціях, а саме: автентифікації та авторизації.

Пул користувачів і пул ідентифікаційних даних є двома основними компонентами Amazon Cognito [22].

Пул користувачів — це не що інше, як сховище, де зберігаються дані профілю користувача. Коли нові користувачі реєструються за допомогою веб- або мобільних програм, цей пул користувачів буде оновлено. У результаті під час входу користувачів їхні облікові дані перевірятимуться на основі даних у пулі користувачів.

Пул ідентифікаційних даних спрощують доступ автентифікованих користувачів до інших ресурсів AWS. Пул ідентифікаційних даних надає тимчасові облікові дані вашим користувачам, щоб вони могли отримати доступ до інших ресурсів AWS без повторного введення своїх облікових даних. Amazon Cognito досить гнучкий, щоб дозволити використовувати пули користувачів і пули ідентифікаційних даних окремо. Їх також можна використовувати разом.

У внутрішній мережі підприємства користувачі входять за допомогою ідентифікатора користувача та пароля. Якщо ідентифікатор користувача та пароль успішно підтверджені, вони можуть увійти в мережу. Ця функція називається автентифікацією і виконується пулом користувачів. Якщо вони хочуть отримати доступ до деяких внутрішніх служб або ресурсів, їм не потрібно повторно вводити свої облікові дані. Їм буде дозволено використовувати службу на основі налаштованих політик і правил. Цю авторизацію вмикає деяка служба, яка надає облікові дані користувача або видає тимчасові маркери доступу для доступу до інших серверів або ресурсів. Ця функція називається авторизацією і виконується пулом ідентифікаційних даних.

AWS Secrets Manager - це сервіс, призначений для централізованого управління, доступу та автоматичного оновлення облікових даних, таких як облікові дані баз даних та додатків, OAuth-токенів, API-ключів та іншої конфіденційної інформації протягом їх життєвого циклу. Він особливо корисний для підвищення безпеки в екосистемі AWS, оскільки дозволяє видалити жорстко закодовані облікові дані з вихідного коду додатків. Натомість облікові дані

динамічно отримуються під час виконання через виклики до Secrets Manager, що допомагає знизити ризик витоку облікових даних[23].

Сервіс також сприяє автоматизації оновлення секретів, дозволяючи замінювати секрети довгострокового користування на секрети короткострокового, тим самим додатково знижуючи ризик компрометації секретів. Оновлення секретів більше не вимагає оновлень додатків або їх розгортання, оскільки облікові дані не зберігаються в самому додатку.

AWS Secrets Manager відповідає численним стандартам, допомагаючи користувачам виконувати вимоги щодо дотримання норм. Він працює за моделлю ціноутворення "оплата за використання" без початкових зборів або витрат на секрети, які заплановані до видалення. Хоча шифрування за допомогою управляємого AWS ключа є безкоштовним, за власні шифрувальні ключі, які управляються через AWS KMS, будуть нараховуватися платежі. Крім того, включення автоматичного оновлення за допомогою функцій AWS Lambda або реєстрація API-викликів через AWS CloudTrail може призвести до додаткових витрат залежно від використання.

На рисунку 2.6 описано наступний сценарій використання AWS Secret Manager:

1. Адміністратор бази даних або адміністратор служби створює облікові дані служби для використання. Наприклад, Адміністратор створює ім’я користувача та пароль для API-сервіса для доступу до бази даних.
2. Адміністратор створює запис у AWS Secret Manager як секрет для програми.
3. Коли програмі потрібні облікові дані, вона надсилає запит AWS Secret Manager через захищений канал HTTPs і TLS, а потім AWS повертає облікові дані програмі.
4. Програма або клієнт аналізують облікові дані та використовують їх у програмі за потреби. Наприклад, у рядку підключення або виклику API.

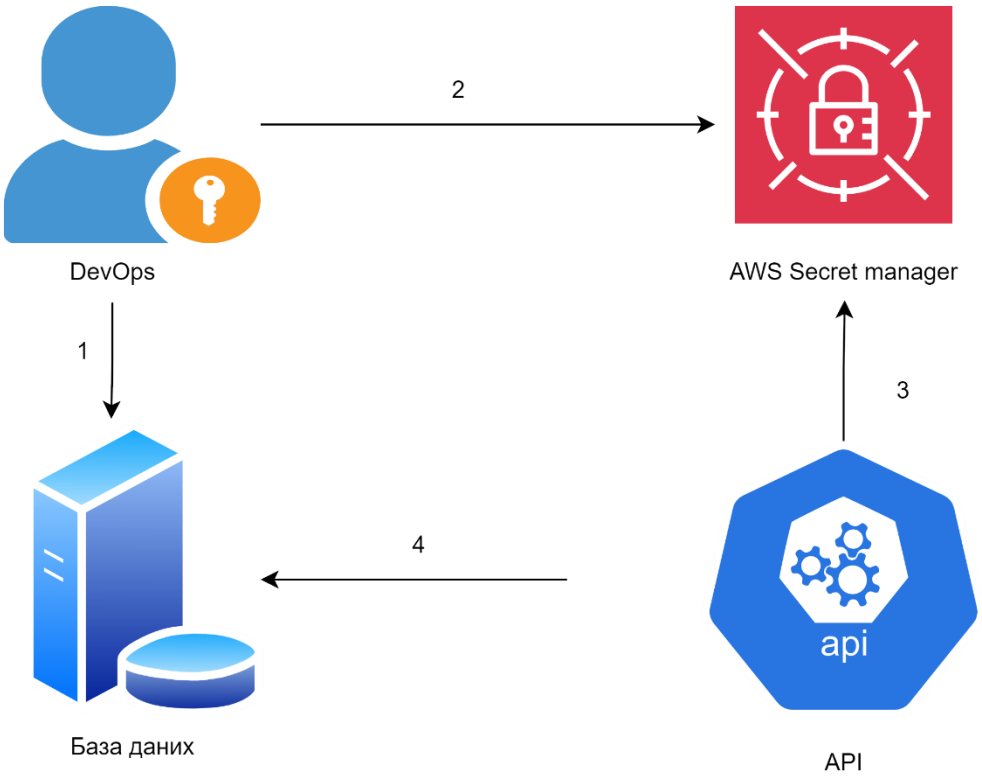


Рисунок 2.6 – Приклад сценарію з використанням Secret Manager

2.2 Розробка архітектури системи

Головна задача розробників полягає у створенні додатків, які б відрізнялися високою ефективністю: були б надійними, зручними у використанні, легко адаптовувалися до нових вимог та були захищені.

У прагненні підвищити продуктивність та оптимізувати час розробки програмного забезпечення, було створено інструменти, які покращують процес планування та моделювання систем, запобігаючи критичним помилкам, які можуть стати відомими лише після написання значної кількості коду. Використання моделей є ключем для чіткої візуалізації структури та функціонування системи, керування архітектурою, зниження ризиків. Моделювання підсилює розуміння системи, що сприяє її спрощенню і забезпечує можливості для повторного використання. Систему можна описати під різними кутами, використовуючи різноманітні моделі, кожна з яких представляє важливу абстракцію системи.

Щоб створити комплексну, безпечну та ефективну веб-програму, розробка буде слідувати найкращим практикам сучасної веб-архітектури, використовуючи весь спектр послуг AWS(рис.2.7) .

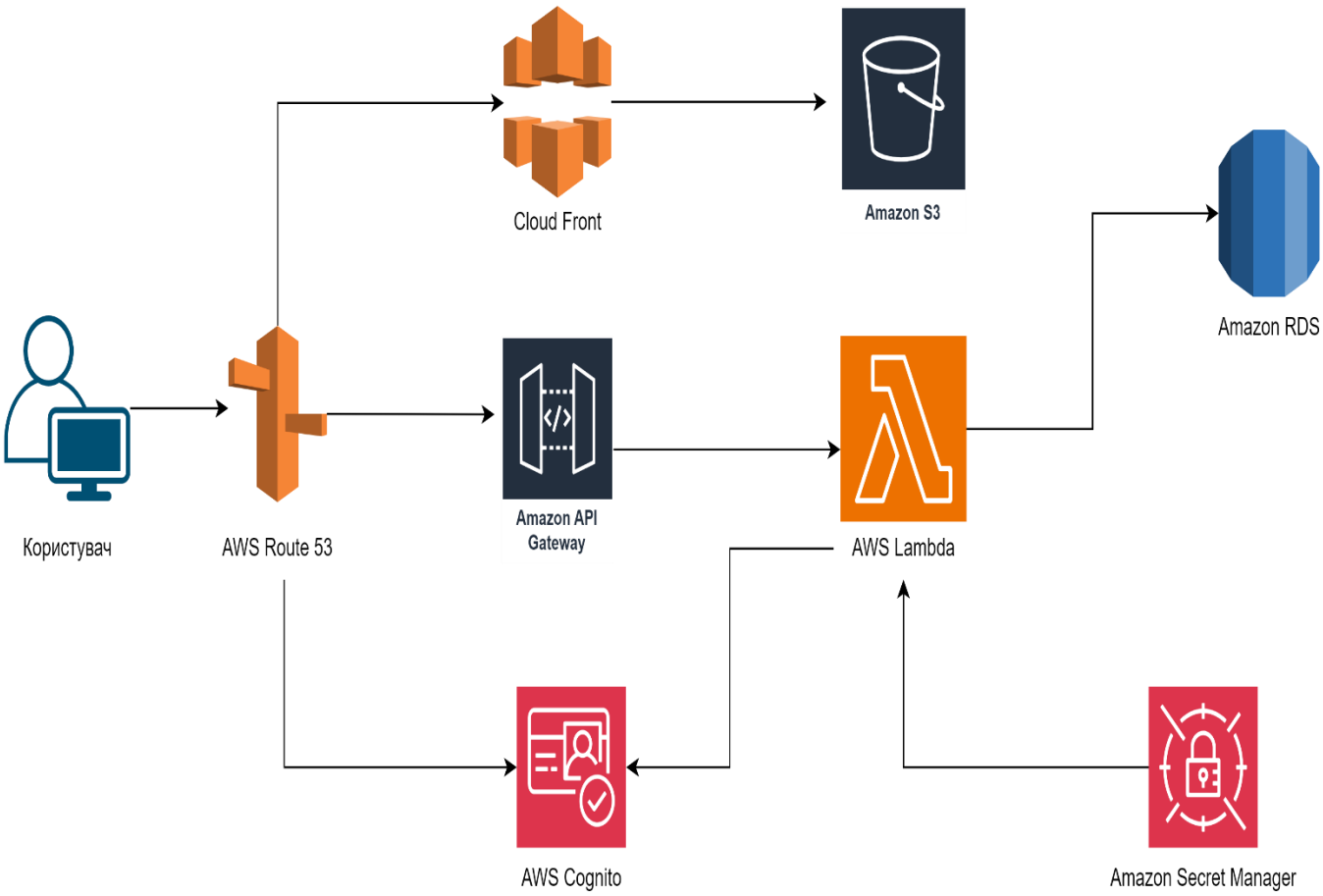


Рисунок 2.7 – Архітектура системи

Починаючи з Amazon S3, статичний веб-контент, як-от файли HTML, CSS і JavaScript, зберігатиметься в надійній службі зберігання. Глобальне охоплення та доступність S3 гарантують, що вміст веб-програми резервовано зберігається на кількох пристроях у кількох приміщеннях, захищаючи від втрати даних. Крім того, точні засоби контролю доступу S3 та інтеграція з AWS Identity and Access Management (IAM) дозволять безпечно керувати доступом до вмісту веб-програми.

Amazon Cognito забезпечить ідентифікацію користувача та службу синхронізації даних, яка забезпечить безпечний доступ та автентифікацію

користувачів. За допомогою Cognito веб-додаток забезпечить реєстрацію та вхід користувачів. Він також підтримує об’єднання з постачальниками ідентифікаційної інформації через SAML 2.0 або OpenID Connect, що означає, що є можливість легко інтегрувати функції входу сторонніх постачальників, таких як Google, Facebook і Amazon, покращуючи взаємодію з користувачем, надаючи більше варіантів входу.

Для RESTful API шлюз Amazon API діятиме як вхідні двері, полегшуючи обробку запитів і відповідей, а також керування трафіком, керування версіями API та моніторинг. Використовуючи безсерверну модель виконання AWS Lambda для наших серверних служб, код буде запускатися без підготовки та керування серверами. Ця безсерверна архітектура є високомасштабованою та економічно ефективною, оскільки оплата відбувається лише за витрачений обчислювальний час.

Amazon RDS підтримає вимоги до бази даних, забезпечуючи змінний розмір реляційної бази даних галузевого стандарту та керуючи трудомісткими завданнями адміністрування бази даних. RDS підтримує кілька механізмів баз даних, включаючи Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database і SQL Server.

За допомогою AWS Secrets Manager відбудеться автоматизація ротації секретів, таких як облікові дані бази даних, ключі API та інша конфіденційна інформація. Це допоможе дотримуватися найкращих практик управління секретами та значно зменшить потенційний ризик зламу облікових даних.

Впровадження Amazon CloudFront, надасть можливість зменшити час завантаження та покращити безпеку. CloudFront розповсюджуватиме веб-додаток по всьому світу за допомогою мережі периферійних місць, надаючи вміст із меншою затримкою. Цей CDN кешуватиме статичні активи веб-програми, зменшуючи кількість прямих запитів до наших сегментів S3, покращуючи взаємодію з користувачем завдяки швидшому часу завантаження. Крім того, інтеграція CloudFront із брандмауером веб-застосунків AWS (WAF) і AWS Shield забезпечує надійний захист від різноманітних типів атак, включаючи

DDoS-атаки на мережевому та прикладному рівні, що забезпечує доступність програми та безпеку користувачів.

Щоб доповнити CloudFront, буде використано Amazon Route 53, який запропонує маршрутизацію трафіку на рівні DNS до веб-програми. Здатність Route 53 підключати запити користувачів до інфраструктури, що працює в AWS, наприклад сегмента Amazon S3, екземпляра Amazon EC2 або Elastic Load Balancer, надає високодоступну та масштабовану хмарну веб-службу системи доменних імен (DNS).

Безпека має першочергове значення, і всі дані, що передаються, будуть зашифровані за допомогою сертифікатів безпеки транспортного рівня (TLS), керованих диспетчером сертифікатів AWS, який також керуватиме процесом оновлення.

Підхід до розробки буде відповідати AWS Well-Architected Framework, щоб переконатися, що ми послідовно дотримуємося найкращих практик і вказівок щодо створення безпечної, високопродуктивної, стійкої та ефективної інфраструктури для програм. Завдяки конвеєрам безперервної інтеграції та безперервного розгортання (CI/CD) ми автоматизуємо наш процес випуску, що дозволить нам створювати, тестувати та розгортати програми швидко та з упевненістю.

Підхід до розробки відповідатиме AWS Well-Architected Framework, забезпечуючи узгодженість у застосуванні найкращих практик і рекомендацій для створення безпечних, високопродуктивних, стійких і ефективних інфраструктур для програм. Завдяки реалізації конвеєрів безперервної інтеграції та безперервного розгортання (CI/CD) процес випуску буде автоматизовано, сприяючи швидкому та надійному створенню, тестуванню та розгортанню програм.

Отже, кульмінацією даного процесу є адаптивний веб-додаток, який не тільки відповідає поточним вимогам, але й надійно розроблений для майбутнього масштабування та вдосконалення. Стратегічне використання послуг AWS створить міцну основу, яка оптимізує доставку цінностей, менше

акцентуючи увагу на тонкощах управління інфраструктурою. Ця збірка ретельно розроблена з урахуванням масштабованості, продуктивності та підвищеної безпеки, що забезпечує стабільну стійкість додатків перед обличчям коливань вимог і нових загроз безпеці. Завдяки впровадженню цих розширених заходів безпеки та використанню безпечної інфраструктури AWS за своєю суттю програма буде добре обладнана для захисту як від внутрішніх, так і від зовнішніх вразливостей. Ця проактивна та комплексна система безпеки гарантує, що веб-програма залишається безпечною та надійною, що є першорядним у сучасній цифровій екосистемі. Це підкреслює прихильність до безпеки, яка йде рука об руку з прагненням до інновацій, забезпечуючи таким чином конкурентну перевагу на цифровому ринку.

3 РОЗРОБКА СИСТЕМИ ЗАХИСТУ ДОДАКТІВ

3.1 Обґрунтування вибору інструментів для реалізації програмного засобу

В задачі розробки веб-системи з інтегрованою системою захисту в рамках хмарного середовища вирішено використати такі мови програмування як Python і JavaScript. За необхідність обрання цих мов для побудови системи стоять важливі причини, зокрема специфіка проєкту, наявність необхідних бібліотек, а також зручність читання коду. Перед початком роботи над проєктом, вкрай важливо вибрати оптимальне програмне забезпечення, яке буде використано для реалізації ключових елементів і модулів спроектованої архітектури.

Обрання мови програмування — це серйозний крок, оскільки від цього великою мірою залежать швидкість та ефективність роботи системи. І хоча на сьогоднішній день існує широкий вибір мов програмування, було вирішено зробити вибір на одній із найбільш широко використовуваних мов програмуванняу у світі — Python.

Python — це мова програмування загального призначення, що відрізняється своєю простотою та читабельністю. З моменту свого створення, основною метою розробників Python було зробити його максимально простим і зрозумілим для користувача. І цю мету вони досягли. Python відомий своєю простотою і наглядністю. Код у Python легко читається завдяки використанню значної кількості пробілів. Таке форматування забезпечує простоту навчання та поширення даної мови серед програмістів різних рівнів. Python відносять до мов програмування високого рівня, що означає невелику кількість фактичного кодування. Переважно, синтаксис Python має більше схожості зі звичайною англійською мовою, ніж з традиційним програмуванням. Одна з ключових особливостей Python — його універсальність. Ця мова програмування підходить для реалізації різноманітних проектів: починаючи з розробки додатків для робочого столу, веб-розробки, закінчуючи обробкою та аналізом даних.

Python має великий набір готових бібліотек та рішень, що значно полегшують розробку та підвищують ефективність процесу.

Також у роботі буде використано JavaScript. З моменту його появи у 1995 році, він став одним з найбільш популярних інструментів для створення функціональних і привабливих веб-сайтів. Окрім веб-розробки, JavaScript також широко використовується в розробці мобільних застосунків, графічних інтерфейсів користувача та багато іншого. JavaScript відрізняється своєю гнучкістю та відкритістю і має потужну інфраструктуру для тестування і відлагодження коду. Його можливо, на приклад, інтегрувати з великою кількістю інших технологій та платформ, таких як Node.js, який перетворює JavaScript з мови для браузерів на повноцінну серверну технологію.

Обидві мови, JavaScript і Python, володіють сильною стороною в тому, що можуть використовуватися для створення Lambda функцій в Amazon Web Services (AWS), які слугують для автоматизації різноманітних завдань в хмарному середовищі. AWS Lambda - це сервіс, який дозволяє запускати код без використання сервера і управління ним, забезпечуючи гнучкість та ефективність у роботі. Така система працює за принципом "відгук на подію", тобто код запускається лише тоді, коли цього вимагає подія. А це означає, що будь-які можливості для запуску програмного коду можуть бути застосовані дуже швидко, але при цьому витрати на запуск та утримання інфраструктури зводяться до мінімуму.

Використовуючи хмарні сервіси Amazon Web Services для розміщення та управління розробленою веб-системою, забезпечується її стабільність та надійність. AWS надає гнучкість вибору обчислювальних ресурсів, що дозволяє оптимізувати витрати. Більше того, хмарне розміщення дозволяє легко масштабувати систему в залежності від потреб користувачів.

Таким чином, використовуючи можливості Python та JavaScript, а також переваги хмарної платформи AWS, буде створено справді потужний, надійний та сучасний веб-додаток. Дане рішення відкриває двері до великої кількості

можливостей, які можуть трансформувати спосіб, яким виконується розробка веб-додатків.

Використання готових бібліотек у розробці проєктів має велике значення та пропонує ряд переваг:

1. **Економія часу:** Готові бібліотеки дозволяють розробникам швидше реалізовувати складні функціональності, оскільки не потрібно писати весь код з нуля. Це особливо важливо у проектах з обмеженими термінами.
2. **Перевірена надійність:** Бібліотеки, що широко використовуються, зазвичай добре протестовані спільнотою. Це зменшує ризики помилок та нестабільності в проєктах.
3. **Підтримка спільноти:** Популярні бібліотеки мають велику спільноту розробників, які можуть надати підтримку, вказати на найкращі практики використання та допомогти вирішити потенційні проблеми.
4. **Стандартизація коду:** Використання загальновідомих бібліотек сприяє стандартизації коду, що робить його більш читабельним та зрозумілим для інших розробників.
5. **Масштабування та оптимізація:** Багато бібліотек оптимізовані для високої продуктивності та підтримки масштабування, що є важливим для великих та складних проєктів.
6. **Оновлення та безпека:** Регулярні оновлення готових бібліотек забезпечують безпеку та сучасність рішень, дозволяючи проекту відповідати актуальним технологічним тенденціям.
7. **Розширюваність:** Готові бібліотеки часто мають модульну структуру, що дозволяє легко розширювати та модифікувати функціональність проекту.

У підсумку, використання готових бібліотек значно спрощує розробку, підвищує якість кінцевих продуктів і дозволяє розробникам зосередитися на унікальних аспектах своїх проектів, замість витрачання часу на розробку базових компонентів.

Під час створення функції Lambda були застосовані наступні бібліотеки Python:

Модуль "os" у Python служить мостом між Python-програмою та операційною системою, надаючи широкий спектр функцій для управління файловою системою та взаємодію з ОС. Він дозволяє розробникам перевіряти існування файлів чи директорій за вказаним шляхом, визначати розмір файлу, а також виконувати численні операції, такі як створення, видалення, перейменування файлів та папок. Модуль "os" надає велику гнучкість при роботі з файловою системою, роблячи його незамінним інструментом для багатьох завдань, пов'язаних із файлами та директоріями у Python-програмах. Деякі з ключових функцій модуля "os" включають :

- **os.path.exists(path)**: Перевіряє, чи існує файл або директорія за заданим шляхом.
- **os.path.getsize(path)**: Повертає розмір файлу в байтах.
- **os.mkdir(path)**: Створює нову директорію за заданим шляхом.
- **os.rmdir(path)**: Видаляє пусту директорію.
- **os.rename(src, dst)**: Перейменовує файл або директорію з **src** на **dst**.

Модуль "sys" у Python - це важливий інструмент, який забезпечує інтерфейс до деяких критично важливих змінних і функцій, які взаємодіють безпосередньо з інтерпретатором Python. Він дозволяє розробникам отримувати детальну інформацію про систему, в якій запущено програму, включно з типом операційної системи, а також керувати взаємодією з самим інтерпретатором.
Основні можливості модуля "sys" включають:

- **sys.argv**: Дозволяє робити доступними аргументи командного рядка, передані програмі.
- **sys.exit()**: Виходить з Python, опціонально передаючи статус виходу або повідомлення про помилку.
- **sys.version**: Надає інформацію про версію Python, яка використовується.
- **sys.platform**: Повідомляє про платформу, на якій запущений інтерпретатор (наприклад, Linux, Windows).
- **sys.stdin**, **sys.stdout**, **sys.stderr**: Для управління стандартними потоками введення, виведення та помилок.

Модуль "re" у Python є центральним інструментом для роботи з регулярними виразами, що знаходить широке застосування в різних областях, від обробки природної мови до валідації даних у веб-застосунках. Цей модуль дозволяє ефективно шукати, витягувати, і модифікувати текстові дані, використовуючи спеціалізовані шаблони пошуку. Таким чином, модуль "re" відіграє ключову роль у багатьох аспектах програмування, пов'язаних з обробкою та аналізом тексту в Python. Ключові функції модуля "re" включають:

- **re.match(pattern, string)**: Перевіряє, чи відповідає початок рядка заданому шаблону.
- **re.search(pattern, string)**: Шукає перше співпадіння з шаблоном у рядку.
- **re.findall(pattern, string)**: Знаходить усі співпадіння з шаблоном у рядку.
- **re.fullmatch(pattern, string)**: Перевіряє, чи весь рядок відповідає шаблону.
- **re.sub(pattern, repl, string)**: Замінює співпадіння з шаблоном на інший текст.
- **re.split(pattern, string)**: Розділяє рядок на підстрічки, розділюючи його по шаблону.
- **re.compile(pattern)**: Компілює регулярний вираз, що дозволяє використовувати його кілька разів.

Модуль "random" у Python є фундаментальним інструментом для генерації псевдовипадкових чисел, що має широкий спектр застосувань, включаючи розробку відеоігор, проведення наукових експериментів та моделювання. Цей модуль забезпечує можливість створювати випадкові числа за різними статистичними розподілами, від простих цілих чисел до чисел з плаваючою комою, забезпечуючи важливу функціональність для симуляцій та випадкового вибору даних. До основних функцій модуля "random" відносяться:

- **random.random()**: Генерує випадкове число між 0 і 1.
- **random.randint(a, b)**: Генерує випадкове ціле число від **a** до **b**.
- **random.uniform(a, b)**: Генерує випадкове число з плаваючою комою в діапазоні від **a** до **b**.
- **random.choice(seq)**: Вибирає випадковий елемент зі списку **seq**.

- **random.shuffle(seq)**: Перемішує елементи списку **seq** випадковим чином.

Python містить вбудований модуль логування, який є важливим інструментом для збору інформації про те, що відбувається під час виконання програми. Модуль логування дозволяє розробникам відстежувати події, що відбуваються у програмі, що допомагає у відлагодженні та моніторингу стану програми. Цей інструмент є ключовим для багатьох розробників, особливо при роботі з більш складними системами. Основні можливості модуля логування включають:

- **Визначення рівня логування:** Можливість вказати рівень важливості логів, наприклад, DEBUG, INFO, WARNING, ERROR, та CRITICAL. Це дозволяє відфільтровувати тільки ті повідомлення, які є актуальними для конкретної ситуації.
- **Конфігурація обробників:** Можливість надсилати логи до різних місць зберігання, як-от консоль, файл, віддалений сервер і т.д.
- **Форматування логів:** Кастомізація формату логів, що може включати час, рівень логування, повідомлення та інше.

Таким чином, модуль логування в Python - це потужний і гнучкий інструмент, який є незамінним у процесі розробки, відлагодження, та підтримки програмного забезпечення.

Модуль **psycopg** в Python є популярним адаптером для роботи з базами даних PostgreSQL. Цей модуль використовується для з'єднання з PostgreSQL базами даних, виконання SQL запитів, обробки даних та інших операцій, пов'язаних із базами даних. Це надзвичайно корисний інструмент для розробників, які працюють з Python і потребують взаємодії з базами даних PostgreSQL. Цей модуль є ключовим інструментом для розробників Python, які працюють з базами даних PostgreSQL, надаючи їм гнучкі та потужні засоби для роботи з даними.Основні можливості модуля **psycop g** включають:

- **Підключення до бази даних:** Через **psycopg** можна встановлювати з'єднання з базою даних PostgreSQL, використовуючи різні параметри з'єднання.

- **Виконання SQL запитів:** Модуль дозволяє виконувати SQL запити до бази даних, наприклад, для створення, читання, оновлення та видалення даних.
- **Транзакційна обробка:** Підтримка транзакцій дозволяє забезпечити цілісність даних, виконуючи кілька операцій як єдину транзакцію.
- **Робота з курсорами:** Можливість використання курсорів для обробки великих наборів даних, а також для здійснення ітерації по результатам запиту.
- **Обробка помилок: psycopg** дозволяє ефективно обробляти помилки, що можуть виникнути під час роботи з базою даних.

Модуль **json** у Python використовується для роботи з даними у форматі JSON (JavaScript Object Notation). JSON - це легкий формат обміну даними, який легко читати та писати для людей та легко генерувати та парсити для машин. Цей модуль є особливо корисним для веб-розробки, обробки API-відповідей та серіалізації/десеріалізації структур даних. Основні можливості модуля **json**:

- **Серіалізація (Запис JSON): json.dumps()** конвертує Python об'єкти, такі як словники, списки, числа, в строки у формат JSON. Це корисно для створення JSON-відповідей у веб-застосунках або для зберігання даних у файлі у форматі JSON.
- **Десеріалізація (Читання JSON): json.loads()** використовується для перетворення рядка у форматі JSON назад у Python об'єкт. Це використовується при обробці JSON-даних отриманих з веб-запитів або з файлів.
- **Запис та читання JSON у файл: json.dump()** та **json.load()** відповідно використовуються для запису JSON даних у файл та читання JSON даних з файлу.

Бібліотека **boto3** є офіційною бібліотекою Python для взаємодії з Amazon Web Services (AWS). Вона дозволяє Python-додаткам легко працювати з різноманітними сервісами AWS, такими як S3, EC2, DynamoDB, і багатьма

іншими. **boto3** робить процес інтеграції з AWS простішим і більш інтуїтивним, надаючи доступ до AWS SDK. Основні можливості бібліотеки **boto3**:

- **Взаємодія з AWS сервісами: boto3** дозволяє виконувати різноманітні операції з AWS, такі як створення та управління екземплярами EC2, робота з S3 бакетами, взаємодія з DynamoDB тощо.
- **Легке використання API:** Бібліотека надає простий та інтуїтивно зрозумілий інтерфейс для взаємодії з AWS API.
- **Автоматичне керування ресурсами: boto3** підтримує автоматичне управління ресурсами, що полегшує роботу з такими сервісами, як EC2 або S3.

Захист даних з **boto3**:

- **IAM ролі та політики: boto3** використовує AWS Identity and Access Management (IAM) для управління доступом до ресурсів AWS. Це дозволяє точно визначати, які дії дозволені для кожного користувача або сервісу.
- **Шифрування:** Під час роботи з S3, наприклад, можна використовувати шифрування для захисту даних в спокої. **boto3** підтримує як серверне шифрування (SSE), так і шифрування на стороні клієнта.
- **Безпечне управління ключами:** Інтеграція з AWS Key Management Service (KMS) дозволяє безпечно управляти ключами шифрування.

Перелік згаданих модулів та бібліотек був використаний у процесі розробки програмного засобу, який включає в себе широкий спектр функціональностей та можливостей. Ці інструменти були обрані з метою забезпечення ефективності, надійності та гнучкості у реалізації проєкту.

Вибір середовища розробки є ключовим елементом в процесі програмування, оскільки саме від нього залежать такі важливі аспекти, як якість кінцевого продукту, швидкість розробки та загальна зручність праці програміста. Це особливо актуально при роботі з такою гнучкою та широко використовуваною мовою програмування, як Python, яка має велике різноманіття застосувань - від веб-розробки до наукових досліджень. При виборі середовища розробки важливо враховувати декілька факторів. Наприклад, новачкам може бути

зручніше вибирати середовища з більш інтуїтивним інтерфейсом та вбудованими підказками, що сприяють швидкому навчанню. Досвідчені розробники, з іншого боку, можуть шукати середовища з розширеними можливостями налаштування та оптимізації, які дозволяють максимально ефективно використовувати час та ресурси. Також слід звертати увагу на сумісність середовища розробки з додатковими інструментами та бібліотеками, які можуть бути необхідні у певних проектах. Це включає інструменти для управління версіями, тестування коду, а також інтеграцію з базами даних та хмарними сервісами. В цілому, правильний вибір середовища розробки може значно підвищити продуктивність роботи та якість кінцевих проектів при роботі з Python, забезпечуючи при цьому комфорт та задоволення від самого процесу програмування.

Можна виділити декілька найбільш популярних середовищ для розробки на мові Python:

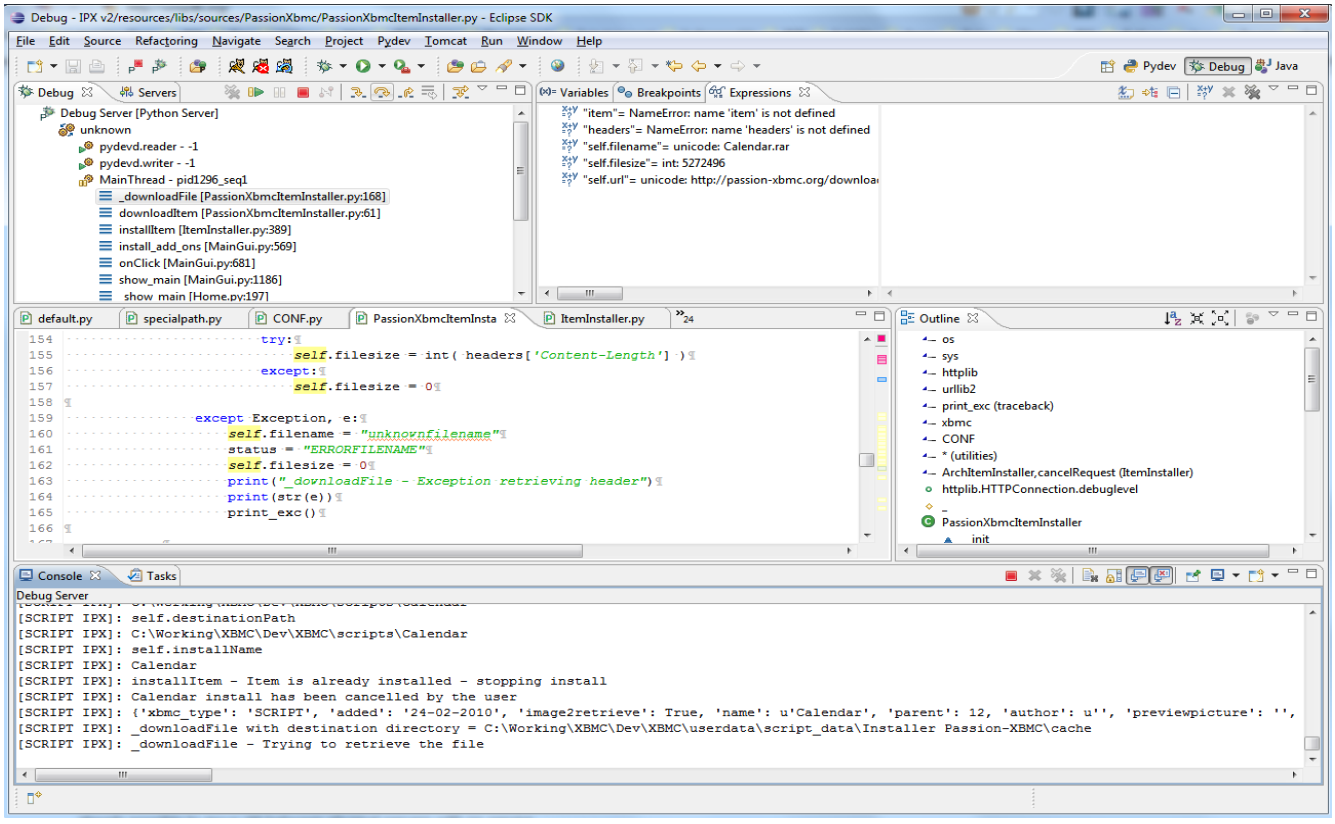


Рисунок 3.1 – Середовище Eclipse

1) Eclipse з плагіном PyDev (рис. 3.1) представляє собою потужне та гнучке середовище розробки з відкритим кодом, яке було розроблено IBM.

Первісно ця IDE була створена для розробки на Java та Android, але завдяки своїй високій адаптивності та розширюваності, Eclipse ефективно підтримує й інші мови програмування, включно з Python. Інтеграція PyDev у Eclipse перетворює цю платформу в повноцінне середовище для розробки на Python. Плагін PyDev додає до Eclipse багато специфічних для Python можливостей, таких як підтримка Django-фреймворку для веб-розробки, графічний дебагер, інструменти для аналізу коду, які допомагають виявляти помилки та вдосконалювати структуру коду. Крім того, PyDev включає можливість інтерактивної роботи з Python консоллю, що дозволяє тестувати код у реальному часі без необхідності перезапуску програми. Ця інтеграція значно розширює функціональність Eclipse, дозволяючи розробникам Python скористатися всіма перевагами, які пропонує ця IDE: висока продуктивність, зручність інтерфейсу, гнучкість у налаштуваннях та широкий спектр доступних інструментів і плагінів. Такий підхід дозволяє зробити процес розробки більш ефективним та приємним, а також спрощує управління комплексними проектами, які можуть включати різні мови програмування та технології.

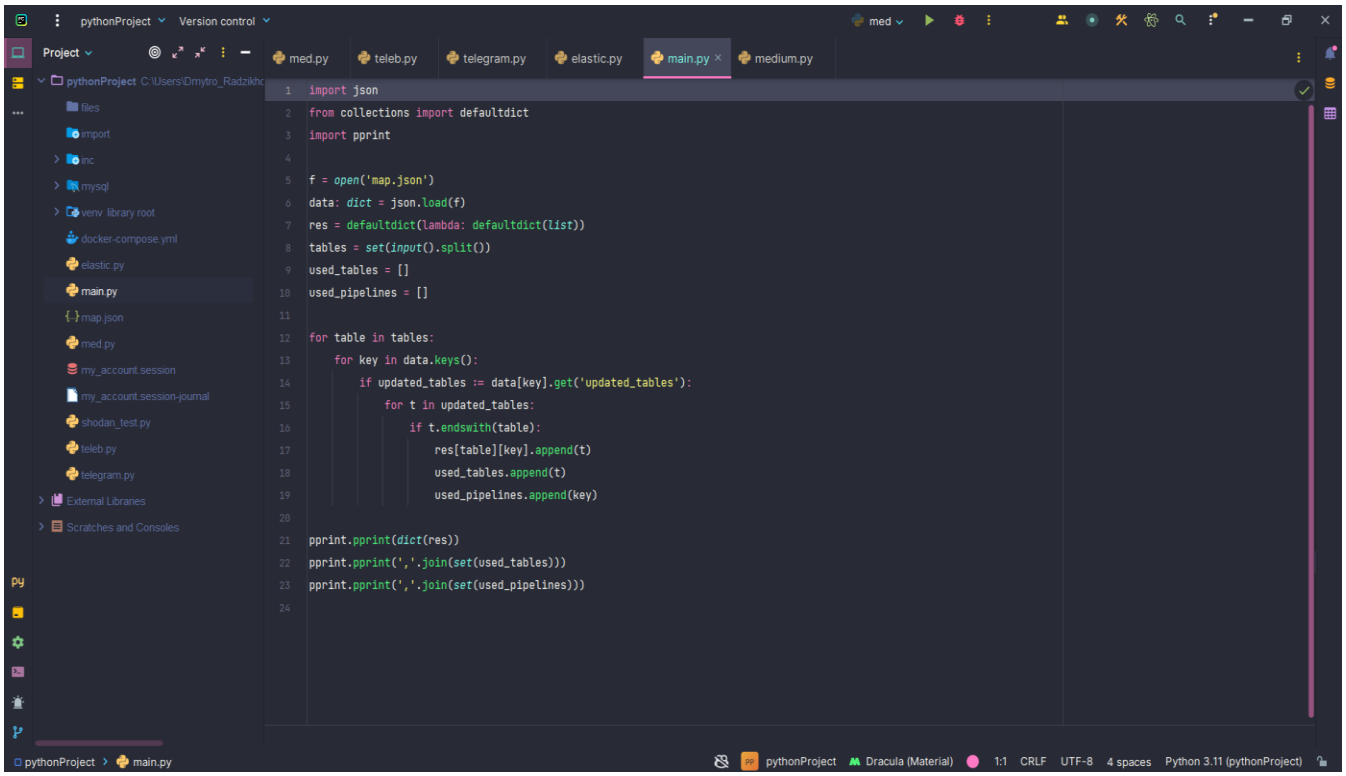


Рисунок 3.2 – Середовище PyCharm

2) PyCharm, розроблений компанією JetBrains, є одним з найвідоміших та найбільш універсальних інтегрованих середовищ розробки (IDE) для Python (рис. 3.2). Ця платформа пропонує безліч функцій, серед яких інтелектуальне завершення коду, ефективна перевірка коду на помилки з підсвічуванням та можливістю швидкого виправлення, автоматизований рефакторинг коду, а також глибокі можливості навігації. PyCharm включає в себе величезну колекцію інструментів, яка робить процес розробки максимально зручним і ефективним. Це охоплює інтегрований налагоджувач, інструменти для тестування коду, Python Profiler для аналізу продуктивності, вбудований термінал, а також інтеграцію з основними системами контролю версій (VCS) і вбудованими інструментами для роботи з базами даних. Окрім цього, PyCharm підтримує можливості віддаленої розробки, включаючи роботу з віддаленими інтерпретаторами, інтегрований термінал SSH, а також інтеграцію з Docker і Vagrant, що робить його ідеальним рішенням для сучасних розробників, які працюють у складних багатофункціональних середовищах. PyCharm також пропонує широку підтримку різноманітних веб-фреймворків Python, мов шаблонів, а також технологій, таких як JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js та інші, що робить його незамінним інструментом для повноцінної веб-розробки. PyCharm доступний у двох версіях: Професійній і версії для спільноти. Версія для спільноти є безкоштовною і з відкритим кодом, що робить її доступною для широкого кола розробників. Професійна версія, яка є платною, надає додаткові професійні інструменти та розширену підтримку, що робить її ідеальним вибором для комерційних проектів і більш складних завдань розробки.
3) Spyder є потужним інтегрованим середовищем розробки (IDE) для Python, який був спеціально створений і написаний на Python для потреб Python-розробників (див. рис. 3.3). Це середовище особливо цінне для вчених, інженерів, аналітиків даних та науковців, оскільки воно поєднує в собі ряд

передових можливостей, які є необхідними для науково-дослідницької роботи. Spyder забезпечує багатофункціональне середовище, яке включає в себе продвинуті інструменти редагування коду, аналізу, налагодження та профілювання.Додатково, Spyder включає в себе різноманітні інструменти для глибокої перевірки коду, які допомагають виявити та усунути помилки, а також оптимізувати код для кращої продуктивності. Це робить процес розробки більш ефективним і менш схильним до помилок, що є особливо важливим у наукових дослідженнях, де точність є ключовою. В цілому, Spyder представляє собою міцну та надійну платформу для розробки на Python, особливо в контексті наукових досліджень, інженерії та аналізу даних. Його спеціалізовані функції і фокус на науковому співтоваристві роблять його одним з провідних виборів для професіоналів у цій сфері.

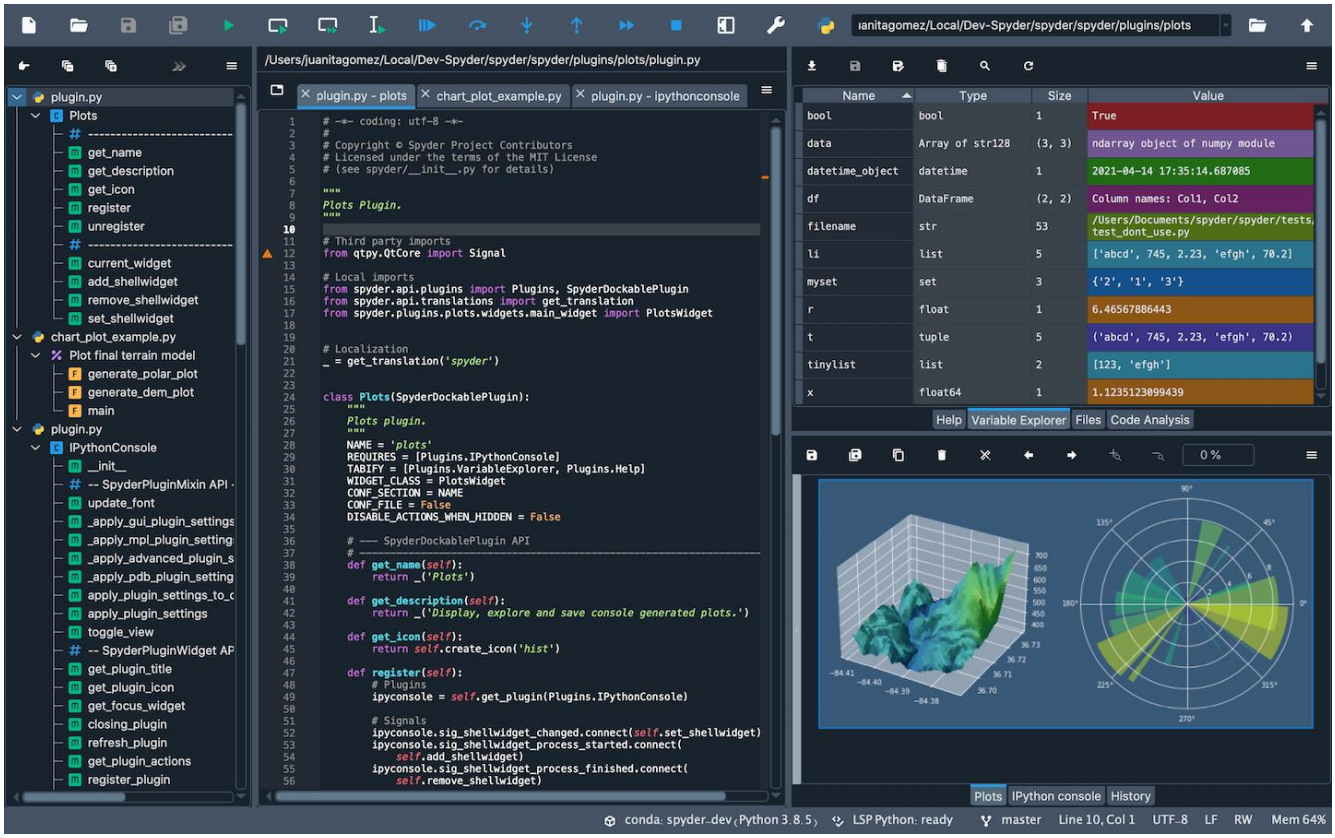


Рисунок 3.3 – Середовище Spyder

4) Sublime Text (див. рис. 3.4), який виступає як потужний редактор коду з інтерфейсом, оптимізованим для програмування на Python, є вибором багатьох розробників завдяки своїй гнучкості та кросплатформенності.

Цей редактор підтримує широкий спектр мов програмування, що робить його універсальним інструментом для різноманітних проектів. Однією з ключових особливостей Sublime Text є його висока налаштовуваність, яка досягається через використання плагінів. Ці плагіни дозволяють користувачам розширювати функціонал редактора, додаючи нові інструменти та можливості, що відповідають їхнім конкретним потребам у розробці. Sublime Text відомий своїми функціями, які значно підвищують продуктивність роботи. Наприклад, функція "Перейти до будь-чого" дозволяє швидко знайти та перейти до будь-якого файлу, символу або рядка, значно спрощуючи навігацію по великих проектах. Крім того, можливість редагування декількох розділів файлу одночасно дозволяє ефективно вносити зміни у різні частини коду без необхідності постійного перемикання між файлами або секціями.

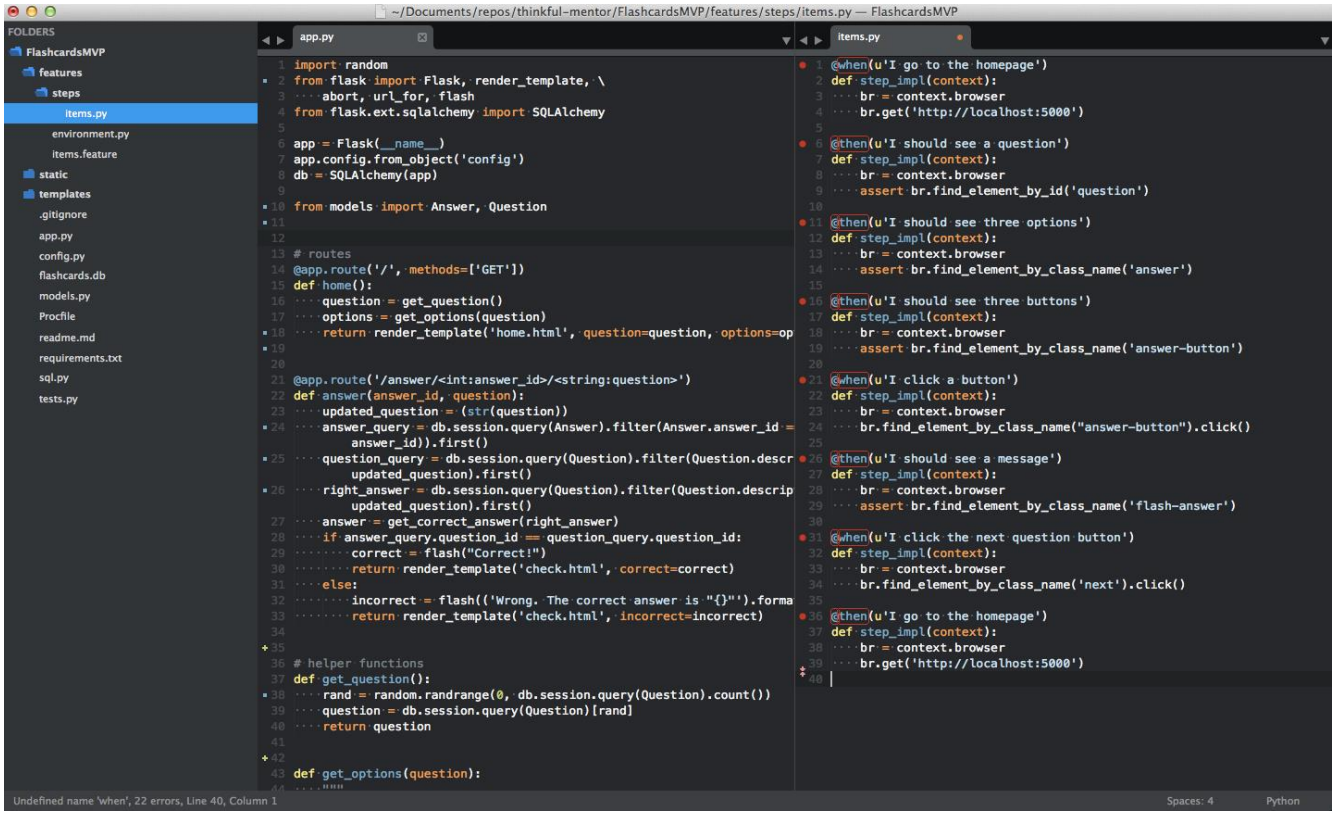


Рисунок 3.4 – Середовище SublimeText

5) Visual Studio Code (VS Code) (див. рис. 3.5) є високофункціональним та розширюваним редактором коду, створеним Microsoft, який важливо не плутати з повноцінним інтегрованим середовищем розробки Visual Studio.

Відрізняючись своєю відкритістю та адаптивністю, VS Code швидко завоював популярність серед розробників і став проектом Microsoft з найбільшою кількістю учасників на GitHub. Це не лише підсилило позиції Microsoft у світі програмного забезпечення, але й визначило компанію як одного з ключових гравців у спільноті розробників. Однією з визначальних рис VS Code є його гнучкість та архітектура на основі розширень. Це означає, що користувачі можуть налаштувати свій редактор, додаючи плагіни та розширення, які адаптують інструмент до конкретних потреб розробки. Наприклад, для роботи з Python достатньо встановити відповідний плагін, що робить VS Code ідеальним для розробки на цій мові, забезпечуючи такі функції, як інтелектуальне завершення коду, налагодження, інтеграція з системами контролю версій і навіть підтримку віддаленої розробки. VS Code також відомий своєю легкістю та швидкістю, що робить його привабливим вибором для розробників, які шукають ефективне та гнучке середовище. Його інтерфейс є інтуїтивно зрозумілим, а велика кількість доступних розширень та плагінів дозволяє легко інтегрувати такі інструменти, як лінтери, форматери коду та інструменти візуалізації. Загалом, Visual Studio Code є сучасним, універсальним та глибоко налаштовуваним інструментом, який відповідає потребам широкого спектру розробників, від новачків до професіоналів. Його популярність у спільноті програмістів заснована на зручності, ефективності та адаптивності, що робить його одним з провідних інструментів у сфері програмування.

Після ретельного аналізу різноманітних інтегрованих середовищ розробки (IDE), доступних на ринку, було вирішено, що для проєкту найбільш відповідним вибором є PyCharm. Це рішення ґрунтується на багатьох перевагах, які пропонує PyCharm, особливо в контексті розробки на Python. Це дозволить максимально використати потенціал мови Python, забезпечуючи при цьому ефективний процес розробки та високу якість кінцевого продукту

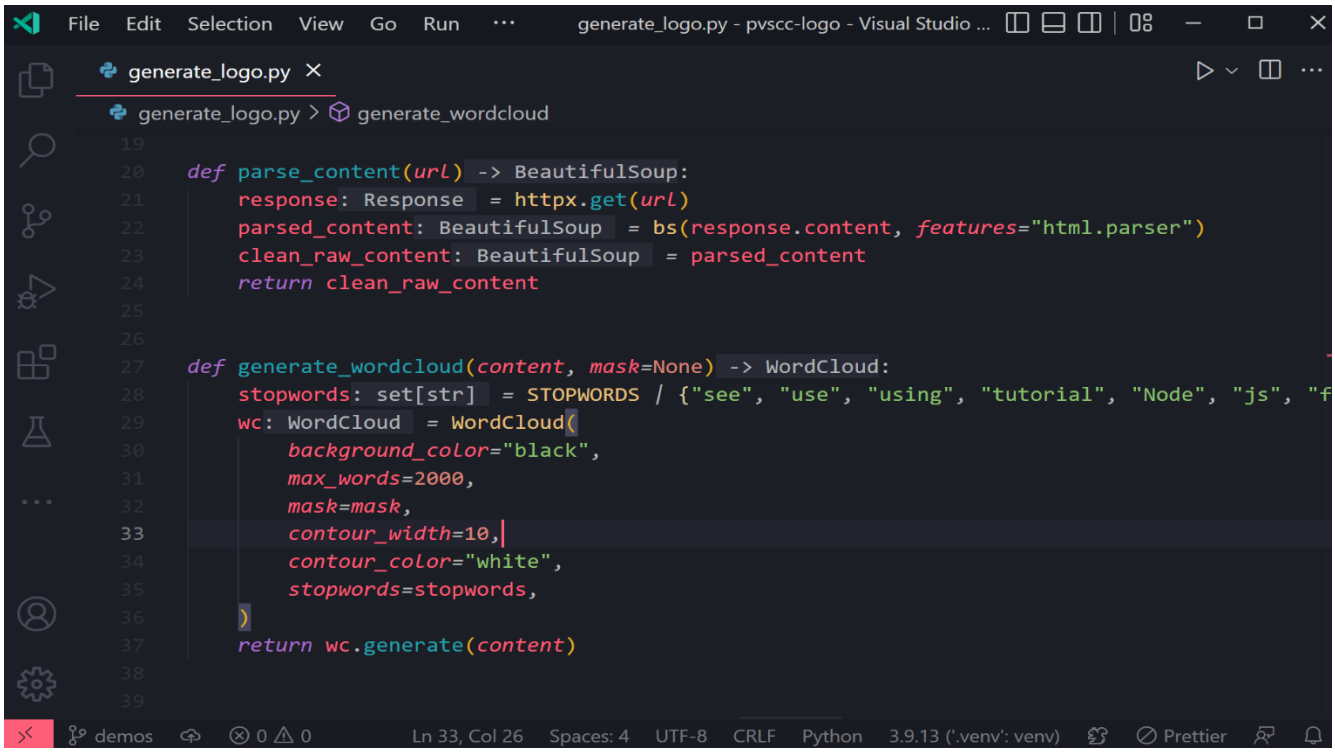


Рисунок 3.5 – Середовище Visual Studio Code

3.2 Розробка програмного засобу

Проєкт зосереджений на створенні спрощеного веб-додатку під назвою 'SkyMessage'. Він розроблений для аутентифікації за допомогою Amazon Cognito та розміщення на Amazon S3, що забезпечує безпечну та надійну платформу. Додаток забезпечує комунікацію через REST API, який створений за допомогою AWS API Gateway. Ця конфігурація підтримується AWS Lambda та DynamoDB, надаючи міцну бекенд-інфраструктуру.

'SkyMessage' створений для надання зручного користувацького досвіду, починаючи з простого процесу входу в систему. Після аутентифікації користувачі отримують доступ до динамічного списку, який показує всіх користувачів, які на даний момент знаходяться в додатку. Однією з ключових функцій 'SkyMessage' є можливість відправки офлайн-повідомлень, що дозволяє користувачам спілкуватися з друзями, навіть коли вони не в мережі. Цей аспект додатку додає йому універсальності та залученості користувачів, роблячи його практичним інструментом для постійного спілкування. Крім того, включення

AWS CloudFront у архітектуру 'SkyMessage' забезпечує покращену продуктивність. Інтеграція CloudFront сприяє доставці контенту із зменшеною затримкою та вищими швидкостями передачі, значно покращуючи загальний користувацький досвід. Цей додаток розроблений із акцентом на плавну взаємодію користувачів та ефективну продуктивність, використовуючи потужні хмарні можливості AWS для створення міцного та зручного чат-додатку.

Для початку роботи над проєктом, який використовує хмарні технології Amazon Web Services (AWS), першим кроком є реєстрація облікового запису на офіційному сайті AWS. Реєстрація на https://aws.amazon.com/ надає доступ до широкого спектру служб та ресурсів AWS, які є необхідними для реалізації різноманітних аспектів проєкту.

Після успішної реєстрації користувач потрапляє на головну сторінку AWS (рис. 3.6), де можливе створення користувача з адміністративним доступом. Такий доступ важливий для виконання завдань, пов'язаних із управлінням віртуальними приватними хмарами (VPC), функціями AWS Lambda, API Gateway, S3 та іншими ресурсами. Важливість цього кроку полягає в забезпеченні безпеки та контролю над хмарними ресурсами.

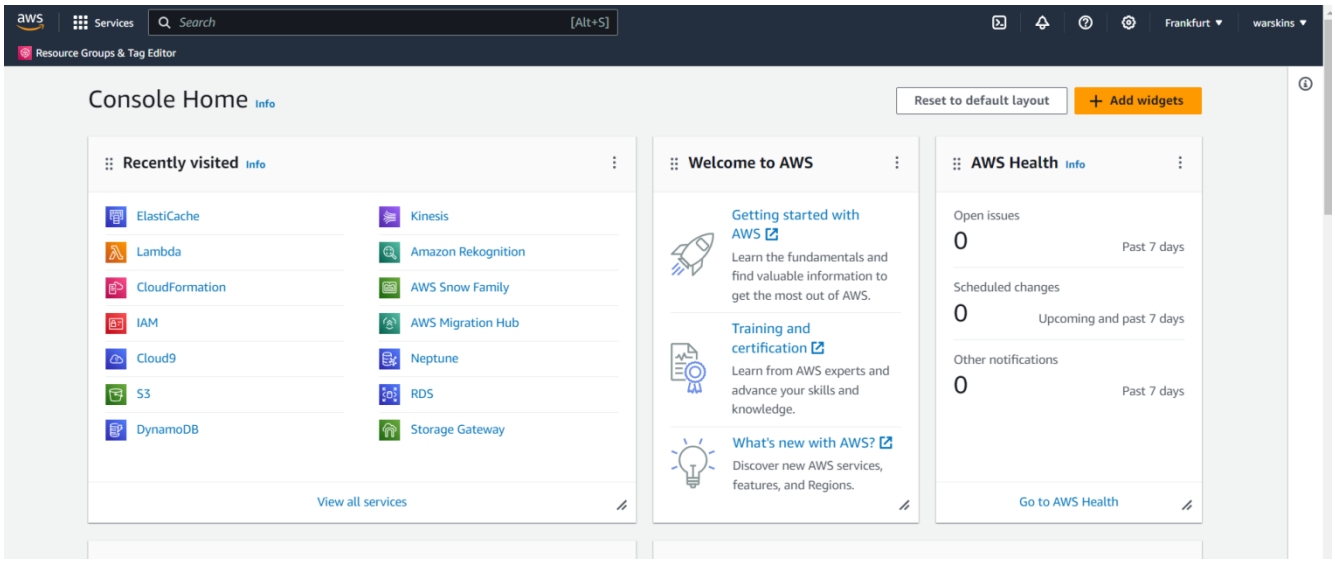


Рисунок 3.6 – Головна сторінка AWS консолі

Для створення користувача з адміністративним доступом необхідно перейти на сторінку 'Services' на головній панелі управління AWS та знайти сервіс 'Identity and Access Management' (IAM). IAM дозволяє керувати

користувачами, групами, ролями та політиками безпеки в AWS. У розділі 'Users' сервісу IAM вибрати опцію 'Add user', що дозволяє створити нового користувача. На цьому етапі важливо ввести ім'я користувача, яке відображає його роль у проекті. Опція 'Programmatic access' надає користувачу доступ до AWS через API. Такий доступ є критично важливим для розробки скриптів, автоматизації задач та інтеграції з сервісами AWS. Він дозволяє ефективніше управляти ресурсами AWS, автоматизувати рутинні задачі та швидко реагувати на зміни у хмарному середовищі. Під час завершення процесу створення користувача, необхідно призначити політики доступу. Правильно сконфігуровані політики доступу забезпечують, що користувачі матимуть лише необхідні повноваження для виконання своїх завдань, знижуючи ризики безпеки. Після створення користувача та надання йому необхідних policy, відбудеться перехід на особистий профіль керування даним користувачем (рис. 3.7).

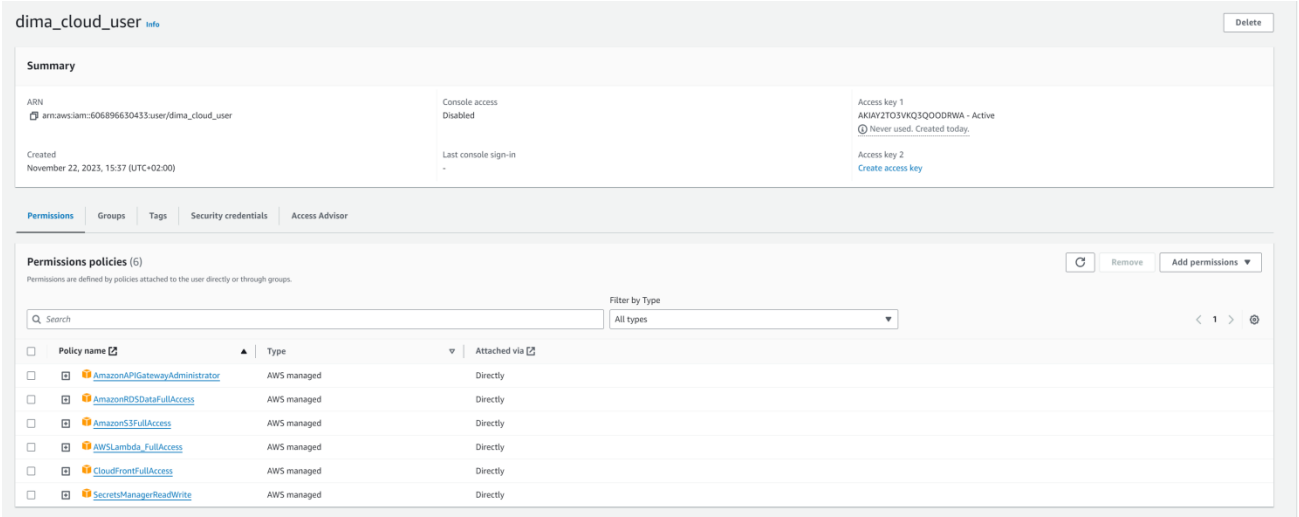


Рисунок 3.7 – Сторінка керування користувачем в IAM

Перед початком розробки будь-якого веб-додатку, що використовує хмарні технології, фундаментальним кроком є налаштування системи управління ідентичністю. Тут на сцену виходить AWS Cognito - передова хмарна служба від Amazon, яка пропонує ефективне рішення для управління ідентичністю та доступом користувачів. Ця служба ідеально підходить для проектів, які потребують надійних, безпечних та масштабованих рішень для реєстрації, входу та управління доступом користувачів до веб- та мобільних додатків. Використання Cognito дозволяє уникнути складнощів, пов'язаних з

розробкою власних серверних рішень для автентифікації та авторизації, тим самим прискорюючи процес розробки та знижуючи витрати.

Розробка ефективного пулу користувачів в Cognito є важливою задачею, яка має безпосередній вплив на функціональність та безпеку веб-додатків. У цьому проєкті вирішено створити базовий пул користувачів, акцентуючи увагу на таких атрибутах, як 'Ім'я користувача' та 'Електронна пошта'. Вибір електронної пошти як основного методу верифікації відповідає сучасним стандартам безпеки та зручності користувачів. Це дозволяє забезпечити легку верифікацію користувачів та уникнути складностей, пов'язаних з багатофакторною аутентифікацією (MFA), яка може бути надмірною для додатків з помірними вимогами до безпеки (рис.3.8) .

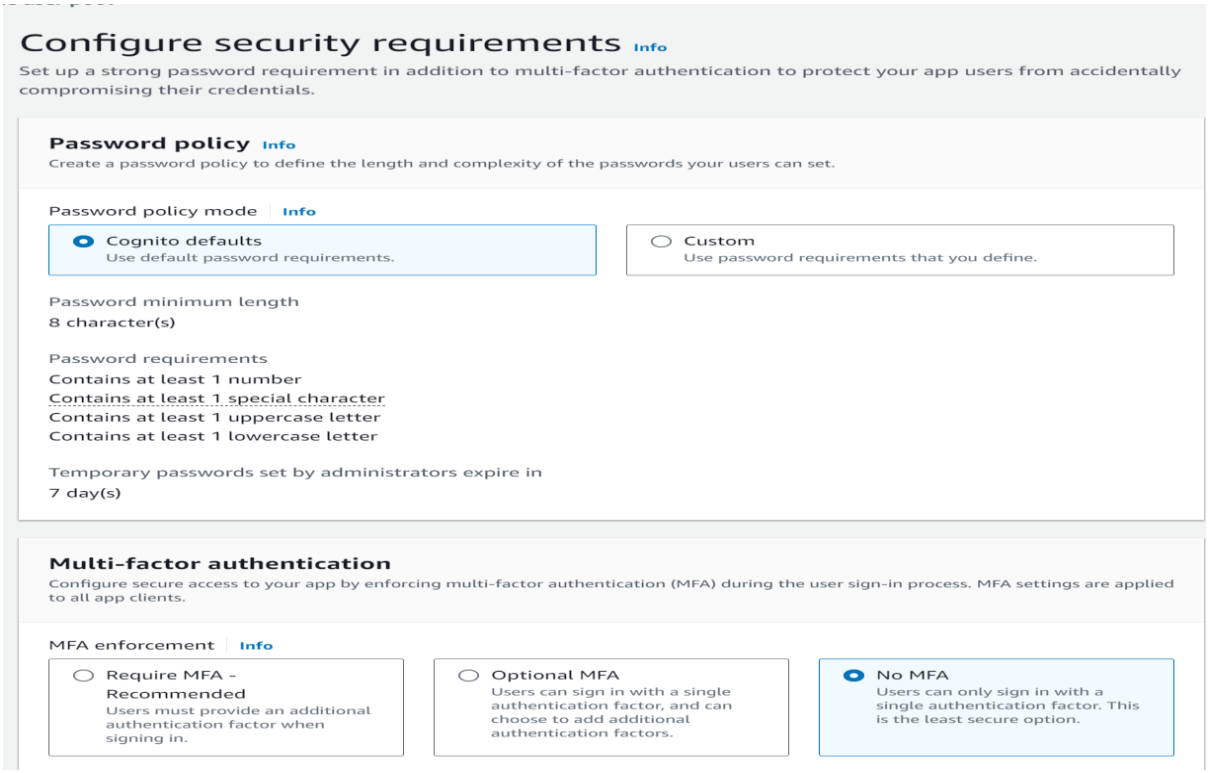


Рисунок 3.8 – Налаштування параметрів безпеки Cognito

Налаштування пулу користувачів без MFA та верифікації за допомогою телефону сприяє зручності та швидкості реєстрації користувачів. Цей підхід особливо корисний для додатків, де швидкість реєстрації та простота доступу є ключовими факторами задоволеності користувачів. Водночас, важливо зазначити, що в залежності від конкретних вимог безпеки та природи даних, які

обробляються у додатку, може бути доцільним реалізувати додаткові заходи безпеки, такі як MFA або інші методи верифікації.

Після налаштування пулу користувачів у Amazon Cognito, критично важливим етапом є його інтеграція з веб-додатком. Цей процес передбачає включення конфігураційних даних, які забезпечують зв'язок між хмарними службами аутентифікації та додатком (рис. 3.9). Важливість цього кроку полягає у створенні безпечного та ефективного механізму управління ідентичністю користувачів, що є фундаментальним для забезпечення безпеки та конфіденційності даних.

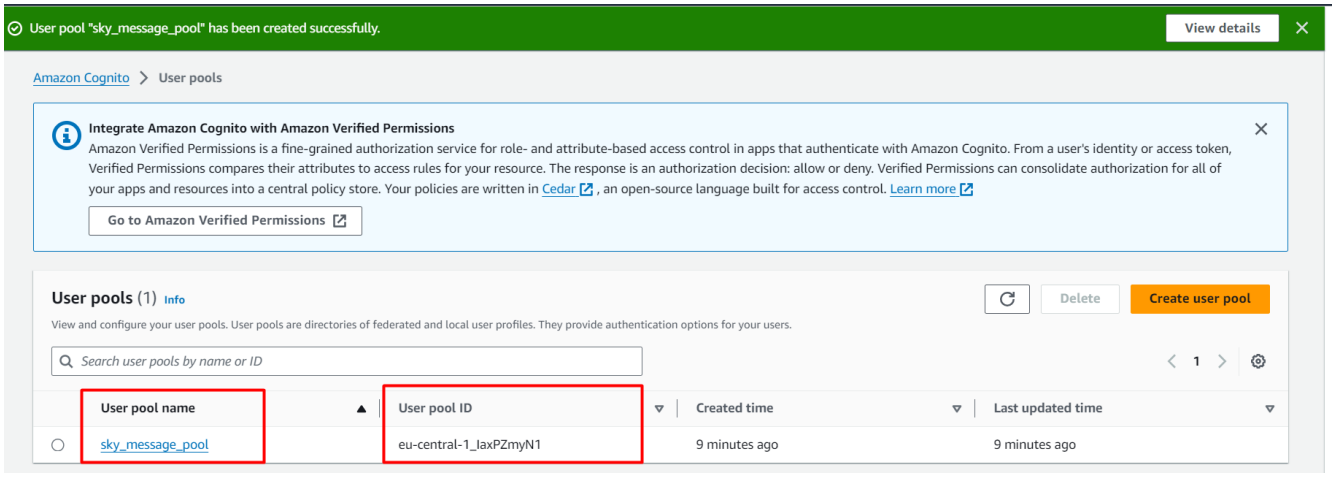


Рисунок 3.9 – Створений Cognito pool з конфігураційними даними

Для здійснення інтеграції, перш за все, потрібно визначити ідентифікатор пулу користувачів та ідентифікатор клієнта. Ці унікальні ідентифікатори служать як ключі доступу, які дозволяють веб-додатку взаємодіяти з пулом користувачів Cognito. Вони виконують критичну роль у процесі верифікації користувачів, надаючи можливість перевірки їхніх облікових записів та прав доступу.

Ідентифікатор пулу користувачів є унікальним кодом, що ідентифікує конкретний пул користувачів у Cognito. Цей код необхідно ввести у відповідному місці конфігураційного файлу веб-додатку. Ідентифікатор клієнта, в свою чергу, є унікальним ідентифікатором, що асоціюється з додатком і дозволяє взаємодіяти з пулом користувачів.

Інтеграція цих ідентифікаторів у веб-додаток є важливою, оскільки вона дозволяє забезпечити безпеку входу користувачів, контролюючи доступ до додатку та захищаючи користувацькі дані. Цей процес також сприяє гнучкості та масштабованості додатку, оскільки Cognito дозволяє легко керувати користувацькими сесіями та доступом до різних ресурсів. Ці ідентифікатори будуть зберігатись в окремому конфігураційному файлі config.js. Приклад файлу наведено нижче.

```
const cognitoPool = {
    UserPoolId: "eu-central-1_IaxPZmyN1",
    ClientId: "717sl7ajgdqqaud1k6sspcndk5",
  };
```

Розробка системи реєстрації та авторизації є фундаментальною частиною створення будь-якого веб-додатку, особливо коли мова йде про використання хмарних технологій, таких як Amazon Cognito. Процес налаштування авторизації включає кілька важливих кроків, кожен з яких має своє значення для забезпечення безпеки та зручності користувачів.

Першим етапом є створення сторінки реєстрації. Така сторінка дозволяє новим користувачам створювати обліковий запис, а існуючим користувачам - увійти в систему. Реалізація ефективної сторінки реєстрації сприяє збільшенню кількості реєстрацій та підвищує загальну задоволеність користувачів.

Для взаємодії з Cognito необхідно додати відповідну бібліотеку. У цьому випадку використовується 'amazon-cognito-identity.min.js', який є частиною пакету amazon-cognito-identity-js. Цей пакет можна завантажити з npm і він дозволяє безпосередньо взаємодіяти з пулом користувачів Cognito, що значно спрощує процес розробки.

Після цього додається логіка реєстрації, яка описує, як користувачі створюють свої облікові записи. Успішна реєстрація перенаправляє користувача на сторінку підтвердження, де необхідно ввести код, отриманий на електронну пошту. Цей крок є важливим для верифікації та підвищення рівня довіри до облікового запису користувача. Дана логіка описана наступним фрагментом коду:

```
function registerUser(event) {
  event.preventDefault();
  console.log("Registering user");

  const username = document.getElementById('username').value;
  const emailAddress = document.getElementById('email').value;
  const userPassword = document.getElementById('password').value;

  const emailAttribute = new AmazonCognitoIdentity.CognitoUserAttribute({
    Name: 'email',
    Value: emailAddress,
  });

  userPool.signUp(username, userPassword, [emailAttribute], null, (error,
result) => {
    if (error) {
      console.error('Signup error:', error);
      alert(error.message || JSON.stringify(error));
      return;
    }
    window.location.href = `confirm.html#${username}`;
  });
}
```

У випадку, коли код підтвердження не отримано або він втратив свою дійсність, система повинна надавати можливість повторного відправлення коду. Це забезпечує додаткову гнучкість та підтримку користувачів, які можуть зіткнутися з проблемами під час реєстрації. Це реалізовано за допомогою наступного кода.

```
function resendVerificationCode(event) {
  event.preventDefault();

  const usernameFromURL = window.location.hash.slice(1);
  const cognitoUserData = {
    Username: usernameFromURL,
    Pool: userPool,
  };

  const cognitoUser = new AmazonCognitoIdentity.CognitoUser(cognitoUserData);

  cognitoUser.resendConfirmationCode((error) => {
    if (error) {
      console.error('Error in resending code:', error);
      alert(error.message || JSON.stringify(error));
      return;
    }
    alert('Confirmation code resent successfully.');
  });
}
```

Після завершення процесу реєстрації користувач перенаправляється на сторінку входу, де розробляється відповідна логіка. Сторінка входу є важливою частиною додатку, оскільки вона відповідає за перевірку даних користувача та надання доступу до функцій додатку. Успішний вхід перенаправляє користувача на головну сторінку з кнопкою "start". Це також оновлює навігаційну панель,

відображаючи ім'я користувача, що ввійшов. Цей функціонал реалізований за допомогою наступного фрагмента кода.

```
function loginUser(event) {
  event.preventDefault();

  const username = document.getElementById('username').value;
  const password = document.getElementById('password').value;

  const authData = {
    Username: username,
    Password: password,
  };

  const authDetails = new AmazonCognitoIdentity.AuthenticationDetails(authData);
  const userConfig = {
    Username: username,
    Pool: userPool,
  };

  const cognitoUser = new AmazonCognitoIdentity.CognitoUser(userConfig);

  cognitoUser.authenticateUser(authDetails, {
    onSuccess: () => {
      console.log('Login successful');
      window.location.href = 'index.html';
    },
    onFailure: (error) => {
      console.error('Login failed:', error);
      alert('Login failed: ' + error.message || JSON.stringify(error));
    },
  });
}
```

Завершуючи роботу з автентифікацією, розробник має тепер веб-додаток з підтримкою ідентичності. Важливість цього не можна недооцінювати, адже сучасні користувачі очікують безпечного, надійного та зручного способу входу та реєстрації у веб-додатках. Такий підхід забезпечує високий рівень безпеки та зручності, що є ключовими для успішної роботи веб-додатку.

Наступним етапом роботи буде розробка функції AWS Lambda, яка буде відповідати за відображення списку користувачів, які увійшли в систему.

Lambda - це функція, яка працює у хмарному середовищі AWS. Функцію можна написати на будь-якій мові програмування, у даному випадку використовується Python. Коли надходить тригер або запит, AWS Lambda ініціює запуск контейнера для виконання коду. Цей час на запуск контейнера з кодом спричиняє певну затримку, яку називають "холодним стартом". Це зазвичай відбувається при першому запиті, а подальші запити не матимуть такої затримки.

Для доступу Lambda до інших ресурсів AWS, необхідно призначити їй роль та політику. Створення ролі та політики відбувається в сервісі IAM. Опис політики здійснюється через JSON (рис. 3.10).

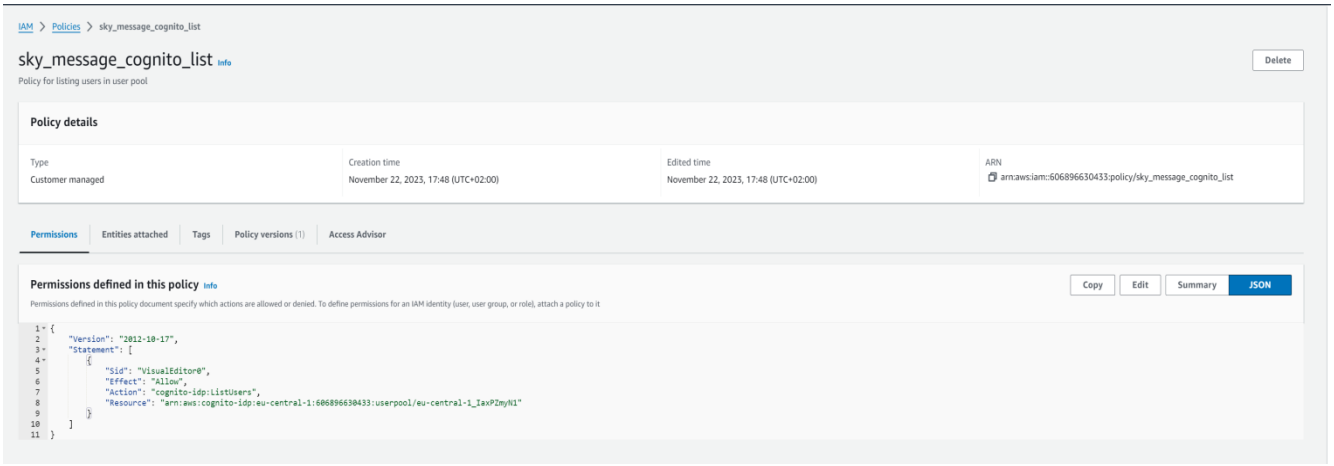


Рисунок 3.10 – Створення політики для Lambda функції

Після створення політики безпеки, потрібно створити роль, яка буде надана Lambda функції для роботи з сервісом AWS Cognito. Кожна роль може мати декілька політик, які зручно маштабувати.

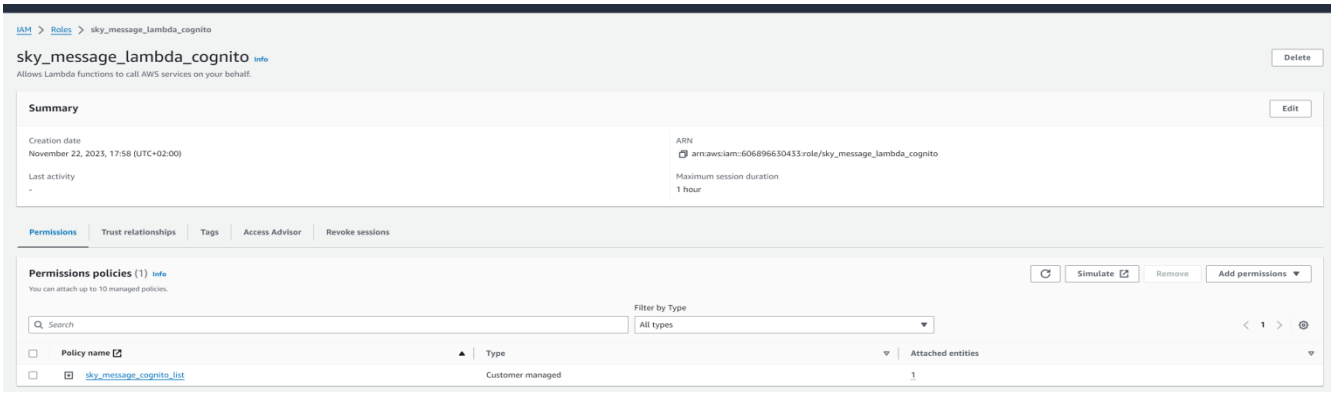


Рисунок 3.11 – Створення ролі для Lambda функції

На цьому етапі мета полягає у створенні функції Lambda, яка може відображати список користувачів, що увійшли в пул користувачів Cognito. Тому було створено відповідну політику та роль, щоб Lambda могла отримати доступ до пулу користувачів.

Створення Lambda функції в AWS, коли вже написаний код на Python і підготовлена IAM роль, є процесом, який включає кілька етапів інтеграції та налаштування у хмарному середовищі AWS. Починається все з відкриття AWS

Management Console, де вибирається сервіс Lambda для створення нової функції. Під час створення функції обирається Python як виконавче середовище, вказуючи на те, що код було розроблено на цій мові програмування (рис.3.12) .

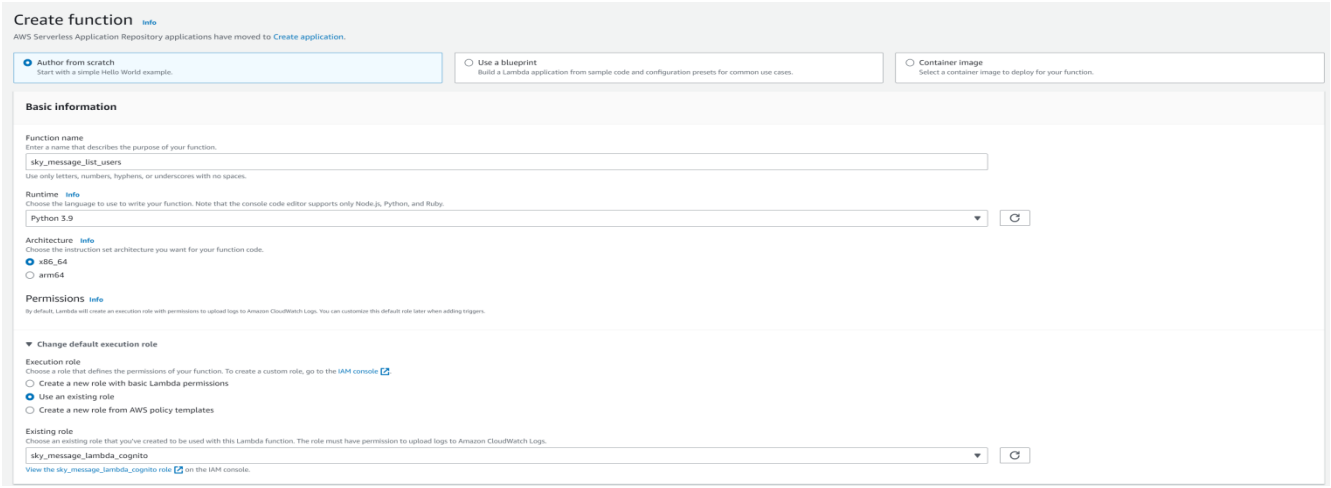


Рисунок 3.12 – Створення Lambda функції

Після вибору ролі та виконуваного середовища, наступним кроком є завантаження коду функції. Це може бути зроблено шляхом прямого введення коду у інтерфейсі AWS (рис. 3.13) або завантаження файлу з кодом. У разі використання зовнішніх бібліотек або залежностей, важливо переконатися, що вони також включені.

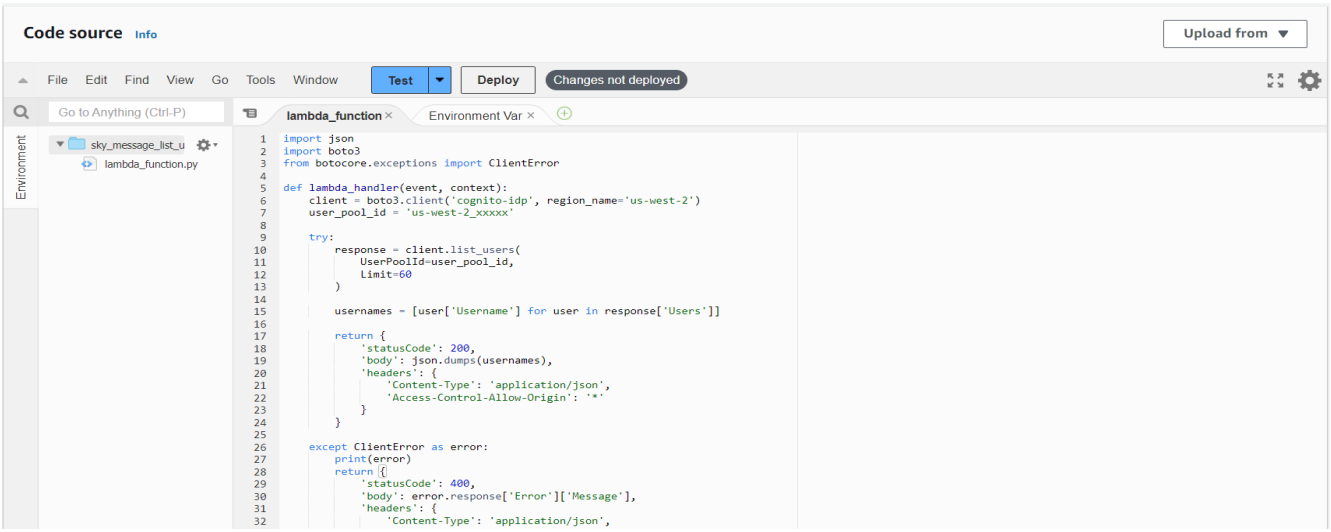


Рисунок 3.13 – Редагування коду Lambda функції у інтерфейсі AWS

Наступний код Lambda функції, демонструє взаємодію з Amazon Cognito для отримання списку користувачів із певного пулу користувачів. Він

реалізований за допомогою Boto3, який є офіційним AWS SDK для Python і забезпечує зручні інструменти для взаємодії з різноманітними AWS сервісами.

```
import json
import boto3
from botocore.exceptions import ClientError

def lambda_handler(event, context):
    client = boto3.client('cognito-idp', region_name='us-west-2')
    user_pool_id = 'us-west-2_xxxxx'

    try:
        response = client.list_users(
            UserPoolId=user_pool_id,
            Limit=60
        )

        usernames = [user['Username'] for user in response['Users']]

        return {
            'statusCode': 200,
            'body': json.dumps(usernames),
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            }
        }

    except ClientError as error:
        print(error)
        return {
            'statusCode': 400,
            'body': error.response['Error']['Message'],
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            }
        }
```

Скрипт починається з імпорту необхідних модулів. Модуль json використовується для серіалізації даних у формат JSON, що є стандартом для обміну даними між сервером та клієнтом. boto3 використовується для створення клієнта, який взаємодіє з AWS Cognito. ClientError з botocore.exceptions використовується для ефективної обробки помилок, які можуть виникати під час роботи з AWS сервісами. Функція lambda_handler є точкою входу для Lambda функції. Вона приймає два параметри: event, який передає дані до функції, та context, що надає інформацію про виконання функції. У цьому випадку обидва параметри не використовуються, оскільки основна мета функції - отримати список користувачів з Cognito User Pool.

Клієнт Cognito Identity Provider створюється за допомогою boto3.client. Вказання регіону, у цьому випадку 'us-west-2', є важливим, оскільки багато AWS

сервісів регіонально-залежні. Змінна user_pool_id визначає пул користувачів, з якого будуть отримані дані.

Основна логіка розташована у блоці try-except, що забезпечує обробку винятків та помилок. Використовуючи метод list_users клієнта Cognito, скрипт запитує список користувачів. За допомогою властивості Limit встановлюється максимальна кількість користувачів, що мають бути повернуті. Список імен користувачів формується шляхом ітерації по отриманих відповідях та збору імен користувачів. В кінцевому результаті, цей список серіалізується у формат JSON і повертається як частина відповіді Lambda функції. У відповіді також встановлюються заголовки HTTP, які дозволяють відображення відповіді у різних клієнтських додатках, зокрема дозволяючи cross-origin запити.У разі виникнення помилок, скрипт обробляє їх та повертає відповідь з відповідним статус-кодом та інформацією про помилку. Використання print дозволяє логувати помилку, що може бути корисним під час налагодження та дебагінгу. У разі виникнення помилок, скрипт обробляє їх та повертає відповідь з відповідним статус-кодом та інформацією про помилку. Використання print дозволяє логувати помилку, що може бути корисним під час налагодження та дебагінгу.

Наступним кроком у розробці хмарного рішення на платформі AWS є налаштування API Gateway, важливого компонента, який відіграє ключову роль у взаємодії з бекенд-сервісами (рис. 3.14). API Gateway функціонує як централізований вхідний маршрут для додатків, надаючи доступ до даних, бізнес-логіки та інших функцій, що виконуються в хмарі. Цей сервіс від AWS забезпечує широкі можливості для розробників, дозволяючи створювати, публікувати, підтримувати та захищати свої API на будь-якому рівні навантаження і з різними вимогами до масштабування. Однією з головних переваг API Gateway є його здатність до створення як RESTful, так і WebSocket API. RESTful API дозволяє розробляти стандартні веб-сервіси, які відповідають на запити HTTP. WebSocket API в свою чергу підтримує реальний двосторонній

зв'язок, що є ідеальним для розробки інтерактивних застосунків, де потрібна висока швидкість відгуку та здатність до негайної обробки подій.

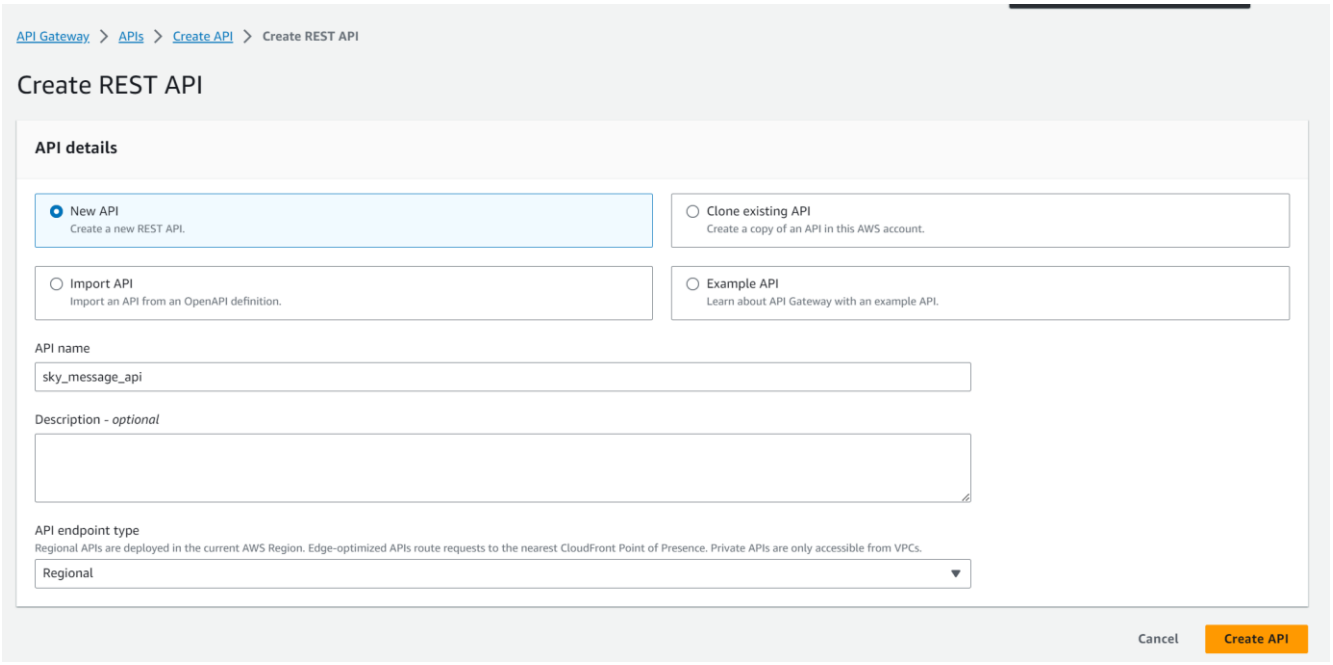


Рисунок 3.14 – Процес створення REST API Gateway

Крім того, API Gateway виконує роль захисника інфраструктури від небажаного трафіку, фільтруючи та керуючи запитами перед тим, як вони досягнуть бекенд-сервісів. Це надає можливість валідації запитів, верифікації токенів доступу, обмеження швидкості запитів та інших заходів безпеки.

Використання API Gateway в архітектурі додатків значно спрощує управління трафіком, масштабуванням та моніторингом, дозволяючи розробникам зосередитися на створенні бізнес-логіки, а не на технічних деталях інфраструктури. Це робить API Gateway незамінним інструментом в екосистемі AWS для створення масштабованих і безпечних веб-сервісів та додатків.

У API Gateway, існує два основних режими роботи: Proxy Mode та Mapping Mode. Ці режими надають гнучкість у налаштуванні та оптимізації взаємодії між фронтендом і бекендом, забезпечуючи ефективну інтеграцію з іншими сервісами AWS.

Proxy Mode в API Gateway пропонує пряме та незмінне проксіювання запитів до вказаної Lambda-функції. У цьому режимі, коли веб-додаток робить HTTP-запит до API Gateway, цей запит передається безпосередньо до Lambda-

функції у вигляді події. Це означає, що всі заголовки, параметри запитів, HTTP-методи та інші дані, які містяться у запиті, автоматично передаються до Lambda. Такий підхід ідеально підходить для сценаріїв, де потрібна максимальна гнучкість у обробці вхідних даних, і дозволяє розробникам Lambda-функцій використовувати весь спектр інформації, яка надходить з фронтенда.

Mapping Mode натомість дозволяє більш контрольоване та цілеспрямоване використання даних, що надходять з HTTP-запитів. В цьому режимі розробник має можливість визначити, які саме дані з запиту мають бути передані до Lambda-функції, та як вони будуть відображені на вхідні параметри функції. Це може включати перетворення URL-параметрів, HTTP-заголовків, або тіла запиту в конкретні змінні або структури даних, які Lambda-функція може більш ефективно обробляти. Цей режим ідеальний для ситуацій, де потрібен високий рівень контролю над тим, як дані з фронтенда перетворюються та використовуються на бекенді.

Окремо необхідно розглянути проблему CORS (Cross-Origin Resource Sharing). Дана проблема часто виникає, коли веб-додаток, розміщений на одному домені, намагається зробити запит до API, що знаходиться на іншому домені. Наприклад, якщо веб-сторінка на aaa.com робить запит до API, розташованого на bbb.com, браузер за замовчуванням блокує такий запит через політику того ж походження (same-origin policy). Для вирішення цієї проблеми на рівні API Gateway, необхідно налаштувати CORS. Це включає в себе визначення відповідних заголовків у відповідях API, які повідомляють браузеру, що запити до API дозволені з певних інших доменів. Наприклад, заголовок Access-Control-Allow-Origin може бути встановлений для дозволу запитів з певних джерел або з усіх джерел (*). CORS також передбачає використання т.зв. "preflight" запитів, які браузер автоматично відправляє перед основним запитом для перевірки, чи дозволяє сервер запит з іншого джерела. Ці запити використовують метод OPTIONS і містять заголовки, які описують тип майбутнього запиту. Сервер, у свою чергу, повинен відповідати на ці запити з вказівкою дозволених методів та заголовків.

Переходячи до створення ендпоінта в API Gateway AWS за допомогою режиму Mapping, кроки, які будуть виконані, є важливими для забезпечення правильної інтеграції та функціональності API. Насамперед, слід вибрати метод, який буде використовуватися для обробки запитів. У даному випадку, це HTTP метод GET, який є стандартним вибором для запитів, що передбачають отримання даних з сервера. GET відомий своєю здатністю швидко і ефективно запитувати дані, не змінюючи стан сервера. Далі, відбувається перехід до вибору способу інтеграції. У цьому контексті, обирається AWS Lambda як цільова платформу. Інтеграція з AWS Lambda означає, що коли запит GET надходить до ендпоінта, він буде автоматично передавати дані до відповідної Lambda-функції (рис. 3.15).

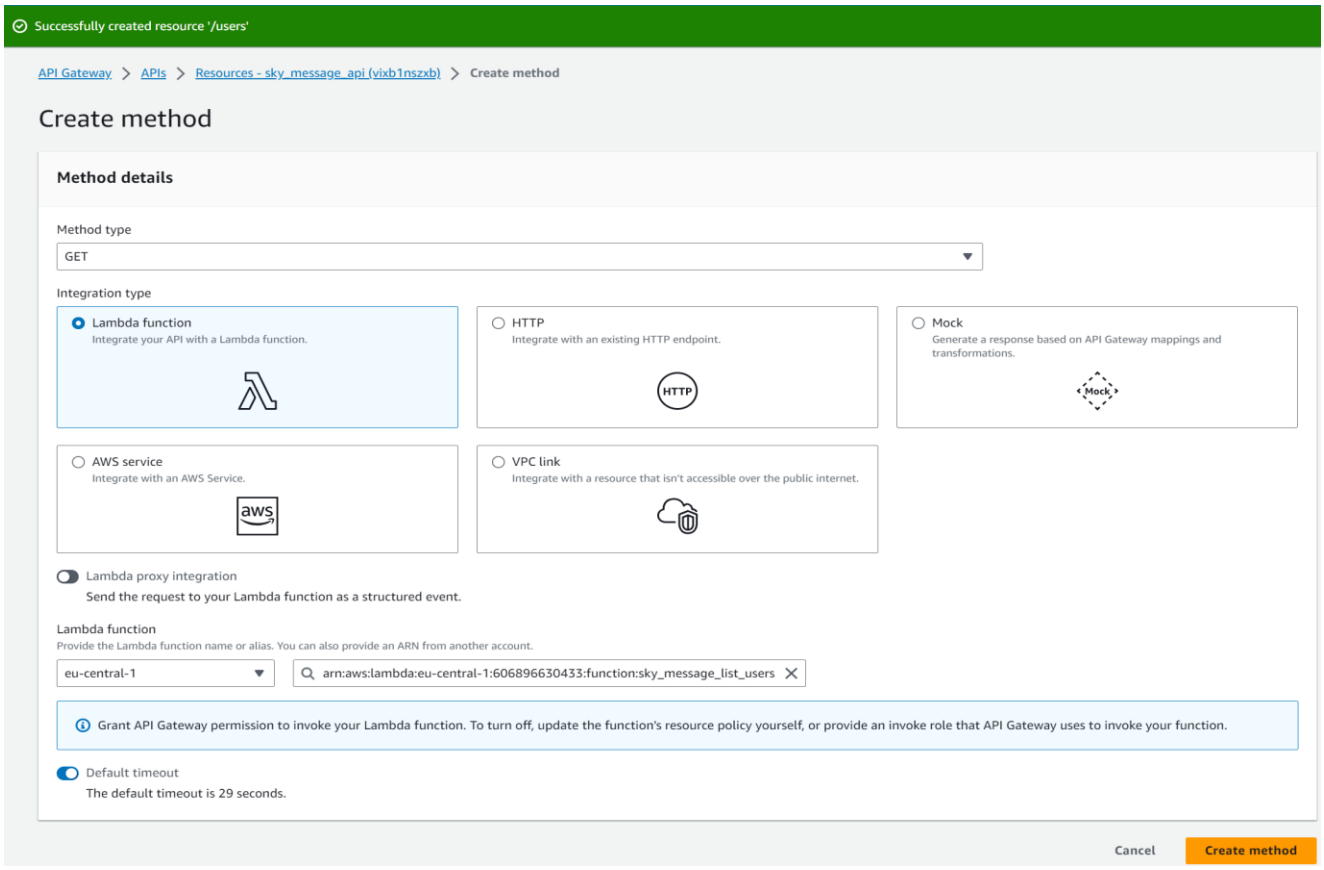


Рисунок 3.15 – Створення GET ендпоінта

Після усіх дій, API буде готовий. Але виникає одна проблема: API не є повністю безпечним. Будь-хто, хто має доступ до URL, може отримати список користувачів, що увійшли в систему. Для підвищення безпеки API є можливість додати авторизатор Cognito до API Gateway. Це важливий крок, адже в

сучасному цифровому світі безпека даних є вирішальною. Додавання Cognito Authorizer дозволяє використовувати можливості Amazon Cognito для управління ідентифікацією та доступом користувачів (рис. 3.16). Це означає, що кожен запит до API буде перевірятися на відповідність правилам безпеки, встановленим у Cognito. Такий підхід забезпечує, що лише авторизовані користувачі матимуть доступ до чутливих даних, що значно знижує ризик несанкціонованого доступу або зловмисних дій.

Завдяки інтеграції з Cognito, API Gateway може використовувати такі методи авторизації, як JSON Web Tokens (JWT) або OAuth токени, що надають надійний механізм перевірки користувачів. Це означає, що кожен запит до API має бути супроводжений валідним токеном, який підтверджує ідентичність користувача та його права на доступ до ресурсів.

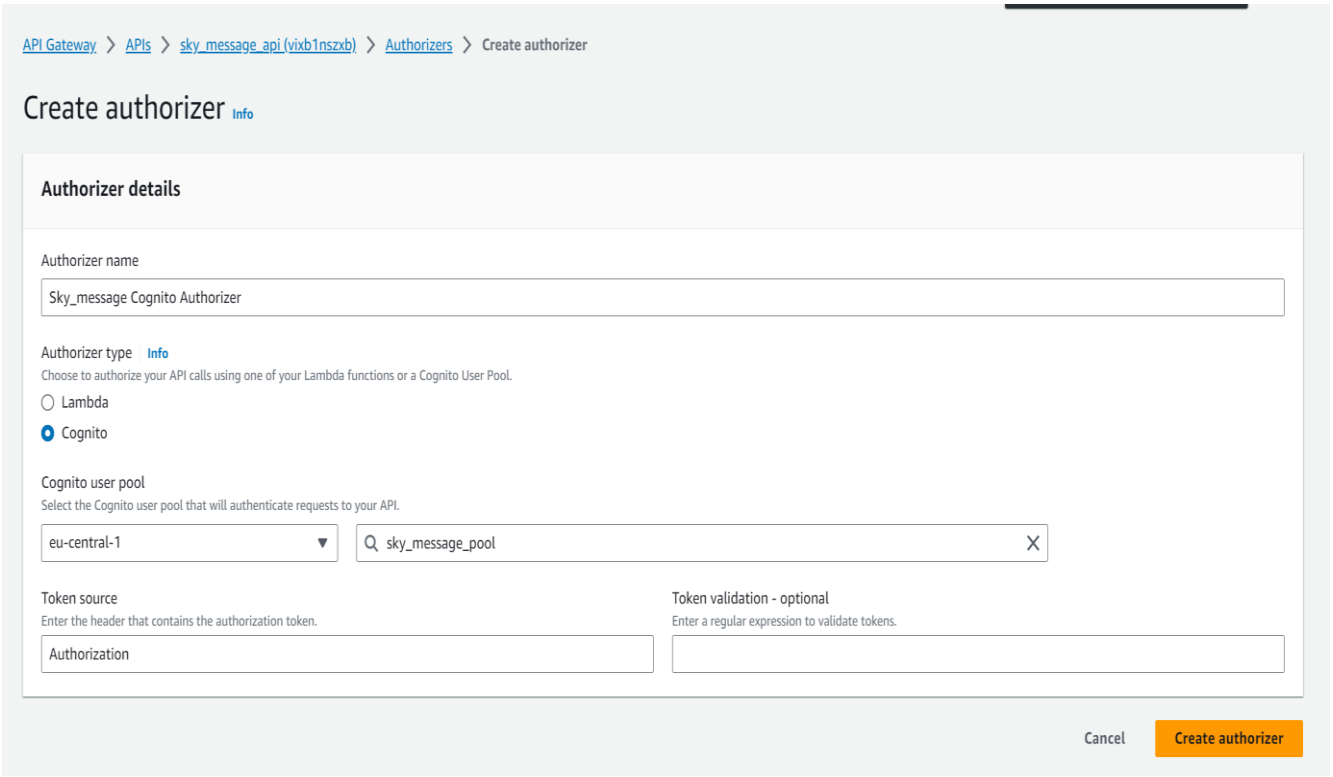


Рисунок 3.16 – Створення Cognito Authorizer

З урахуванням вже впроваджених заходів безпеки, API тепер ефективно захищений за допомогою авторизатора, який забезпечує додатковий рівень контролю доступу. Цей крок становить міцну основу для подальшого розвитку та удосконалення системи.

Фокусуючись на наступному етапі, важливо звернути увагу на автоматизацію процесів розгортання. Автоматизація не тільки спрощує та прискорює процеси, але й забезпечує більшу надійність та консистентність у впровадженні змін. Цей крок є вирішальним у забезпеченні безперебійного та ефективного впровадження оновлень та розширення інфраструктури. Для реалізації цієї стратегії, буде використовуватися CDK, інструмент, який буде детально розглянуто в наступному розділі. Використання CDK є ключовим моментом у стратегії розвитку, оскільки воно сприяє підвищенню ефективності процесів розгортання, одночасно гарантуючи безпеку та швидкість впровадження нових функцій та оновлень.

3.3 Автоматизація розгортання системи

У цьому розділі розглядається процес автоматизації розгортання системи з використанням AWS Cloud Development Kit (CDK) та мови програмування Python. AWS CDK - це потужний інструмент, який дозволяє інфраструктурі, як коду (IaC), визначати хмарну інфраструктуру у звичних мовах програмування. Це дозволяє розробникам використовувати мову Python для опису складних хмарних архітектур, спрощуючи та прискорюючи процес розгортання. Перед початком написання коду важливо зрозуміти ключові концепції та компоненти AWS CDK. Це включає в себе роботу з конструкторами, стеками та додатками в рамках CDK, а також розуміння того, як ці елементи взаємодіють для створення та управління хмарною інфраструктурою.

AWS CDK дозволяє створювати надійні, масштабовані, економічно ефективні застосунки в хмарі зі значною виразною потужністю мови програмування. Цей підхід дає багато переваг, зокрема:

- Створення ресурсів AWS за допомогою високорівневих мов програмування, які автоматично забезпечують безпечні параметри для ресурсів AWS, визначаючи більше інфраструктури з меншим кодом.
- Використання ідіом програмування, таких як параметри, умови, цикли, композиція та успадкування, щоб змоделювати дизайн системи.

- Розміщення інфраструктури, коду програми та конфігурацію в одному місці, гарантуючи, що на кожному етапі буде повна система, яку можна розгортати в хмарі.
- Використання методів розробки програмного забезпечення, такі як перевірка коду, модульні тести та контроль версій, щоб зробити інфраструктуру більш надійною.
- Використання потужностей AWS CloudFormation для передбачуваного та багаторазового розгортання інфраструктури з відкатом у разі помилки.
- Можливість легко ділітися шаблонами проектування інфраструктури серед команд у організації.

AWS CDK підтримує TypeScript, JavaScript, Python, Java, C#/.Net і Go. Розробники можуть використовувати одну з цих підтримуваних мов програмування для визначення хмарних компонентів.

Закладені в основу CDK принципи програмування дозволяють створювати модульні, перевикористовувані, та легко налаштовувані скрипти розгортання. Використання Python для цих цілей не лише сприяє зручності та інтуїтивності для розробників, але й забезпечує потужні можливості для інтеграції з іншими інструментами та сервісами.

Наступним кроком буде розгляд конкретних прикладів коду, які ілюструють процес визначення та розгортання інфраструктури в AWS за допомогою CDK та Python, демонструючи ефективність та гнучкість цього підходу.

Для початку роботи з AWS Cloud Development Kit (CDK) на Python, перш за все необхідно виконати імпорт важливих модулів та класів. Наступний фрагмент коду включає імпорти з бібліотеки aws_cdk, які є фундаментальними для створення та управління хмарною інфраструктурою в AWS. Кожен модуль відповідає за певний аспект хмарної архітектури, такий як управління серверними процесами (_lambda), маршрутизація трафіку (apigw), аутентифікація користувачів (cognito), і т.д. Це забезпечує гнучкість та масштабованість при проектуванні хмарних рішень.

```
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_cognito as cognito,
    aws_s3 as s3,
    aws_dynamodb as dynamodb,
    aws_sns as sns,
    aws_iam as iam,
    aws_logs as logs,
    aws_cloudwatch as cloudwatch,
    aws_cloudtrail as cloudtrail,
    aws_cloudfront as cloudfront,
    aws_secretsmanager as secretsmanager,
    RemovalPolicy,
    CfnOutput,
    Duration,
)
from aws_cdk.aws_cloudfront_origins import S3Origin
from aws_cdk.aws_cloudwatch_actions import SnsAction
from aws_cdk.aws_s3 import BlockPublicAccess
from aws_cdk.aws_sns_subscriptions import LambdaSubscription
from constructs import Construct
```

Головна мета цього коду полягає в тому, щоб надати основу для створення різних хмарних ресурсів, таких як Lambda функції, API Gateway, Cognito для аутентифікації, S3 для зберігання даних, DynamoDB для бази даних, SNS для публікації повідомлень, IAM для управління доступом, CloudWatch для моніторингу, CloudTrail для логування, CloudFront для CDN та Secrets Manager для управління секретами. Це вступ до створення комплексної архітектури, яка буде ефективно використовувати різні сервіси AWS для створення масштабованого, безпечного та надійного хмарного додатка.

Клас Stack у AWS Cloud Development Kit (CDK) є фундаментальним елементом для створення та управління хмарною інфраструктурою. Він дозволяє організувати та управляти групами ресурсів AWS, необхідних для застосунків. У конструкторі класу SkyMessageStack, який є нащадком Stack, визначаються налаштування ключових компонентів інфраструктури. Кожен ресурс, ініціалізується з визначеними параметрами, що забезпечують необхідну функціональність та безпеку. Цей підхід дозволяє розробникам створювати модульні, масштабовані та легко управляємі хмарні рішення.

S3 Bucket: Цей ресурс створюється для зберігання статичних файлів застосунку, зокрема index.html, який виступає як головна веб-сторінка. Налаштування public_read_access дозволяє публічний доступ до вмісту бакету.

Конфігурація block_public_access регулює доступність бакету з метою безпеки. Для описання даного сервісу використано наступний фрагмент коду.

```
bucket = s3.Bucket(
    self,
    "SkyMessageHostBucket56546",
    website_index_document="index.html",
    public_read_access=True,
    block_public_access=BlockPublicAccess(
        block_public_acls=False,
        block_public_policy=False,
        ignore_public_acls=False,
        restrict_public_buckets=False
    )
)
```

CloudFront Distribution: Створюється дистрибутив CloudFront, який діє як CDN (мережа доставки вмісту) для бакету S3. Використання CloudFront оптимізує доставку вмісту користувачам, забезпечуючи швидкий доступ до статичних файлів застосунку, незалежно від їх географічного розташування. Ключові параметри, такі як viewer_protocol_policy, налаштовуються для примусового використання HTTPS, що підвищує безпеку при доступі до вмісту. Даний сервіс описаний наступним фрагментом коду.

```
distribution = cloudfront.Distribution(
    self,
    "StaticSiteDistribution",
    default_behavior=cloudfront.BehaviorOptions(
        origin=S3Origin(bucket),
        viewer_protocol_policy=cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS
    ),
    default_root_object="index.html"
)
```

У настпупному фрагменті коду, за допомогою AWS CDK, створюється таблиця DynamoDB для зберігання даних чатів користувачів.

```
table = dynamodb.Table(
    self, 'users_chats',
    partition_key=dynamodb.Attribute(
        name='sender_receiver',
        type=dynamodb.AttributeType.STRING
    ),
    sort_key=dynamodb.Attribute(
        name='timestamp',
        type=dynamodb.AttributeType.NUMBER
    ),
    billing_mode=dynamodb.BillingMode.PROVISIONED,
    read_capacity=1,
    write_capacity=1,
    removal_policy=RemovalPolicy.RETAIN
)
```

Ця таблиця, названа users_chats, є ключовим компонентом для зберігання та організації даних в хмарній інфраструктурі. Вона використовує систему ключів DynamoDB для ефективного управління даними. Partition_key встановлено як sender_receiver з типом STRING, що дозволяє унікально ідентифікувати записи в таблиці. Sort_key, названий timestamp та визначений як NUMBER, використовується для сортування даних в межах одного розділу, забезпечуючи логічний порядок повідомлень. Вибір режиму тарифікації PROVISIONED вказує на фіксовану пропускну спроможність, що є важливим для планування навантаження та витрат. Встановлені параметри читання та запису - по одній одиниці - означають, що таблиця має обмежену пропускну спроможність. Це може бути корисним у сценаріях з невеликим навантаженням або при тестуванні системи. Політика видалення RemovalPolicy.RETAIN забезпечує збереження даних таблиці навіть після видалення стека CDK, що є важливим для забезпечення безпеки та стабільності даних. Цей фрагмент коду ілюструє, як можна ефективно використовувати AWS CDK для створення та управління хмарною інфраструктурою, зокрема засобами зберігання даних, що є фундаментальним для багатьох хмарних застосунків. Використання DynamoDB та її можливостей є ключовим для створення ефективних, масштабованих та надійних хмарних рішень.\

Опис полік доступу IAM для Lambda функцій також здійснюється з використанням AWS CDK. Наступний фрагмент коду демонструє цю здатність.

```
lambda_shared_role = iam.Role(
    self,
    "LambdaSharedRole",
    assumed_by=iam.ServicePrincipal("lambda.amazonaws.com"),
    managed_policies=[
        iam.ManagedPolicy.from_aws_managed_policy_name("service-
role/AWSLambdaBasicExecutionRole"),
        iam.ManagedPolicy.from_aws_managed_policy_name("service-
role/AWSLambdaBasicExecutionRole"),
        iam.ManagedPolicy.from_aws_managed_policy_name("AmazonCognitoReadOnly"),
        iam.ManagedPolicy.from_aws_managed_policy_name("AmazonSESFullAccess")
    ]
)

dynamodb_policy = iam.PolicyStatement(
    actions=["dynamodb:GetItem", "dynamodb:PutItem", "dynamodb:UpdateItem",
"dynamodb:DeleteItem",
             "dynamodb:Query", "dynamodb:Scan"],
    resources=["*"]
```

```
)
secretsmanager_policy = iam.PolicyStatement(
    actions=["secretsmanager:GetSecretValue"],
    resources=["*"]
)

lambda_shared_role.add_to_policy(dynamodb_policy)
lambda_shared_role.add_to_policy(secretsmanager_policy)
```

Цей кодовий фрагмент створює та конфігурує IAM (Identity and Access Management) роль у AWS CDK для використання в Lambda функціях. Ініціалізація iam.Role вказує, що роль призначена для сервісу Lambda (lambda.amazonaws.com). Роль включає кілька управлінських політик, які дозволяють базове виконання Lambda функцій, доступ до Amazon Cognito у режимі тільки для читання та повний доступ до Amazon Simple Email Service (SES). Додатково, визначаються політики для DynamoDB та AWS Secrets Manager. Політика для DynamoDB дозволяє діям, таким як отримання, створення, оновлення та видалення елементів, а також виконання запитів і сканувань у базі даних. Політика для Secrets Manager дозволяє отримувати значення секретів. Ці політики додаються до створеної ролі, розширюючи її можливості виконання різних операцій у AWS. Цей фрагмент коду є важливим для налаштування безпеки та управління доступом до ресурсів AWS в рамках Lambda функцій, забезпечуючи їх правильне функціонування та інтеграцію з іншими сервісами AWS.

AWS Lambda є потужним інструментом для створення безсерверних додатків у хмарному середовищі. Використання Lambda дозволяє розробникам ефективно управляти обробкою подій і даних без необхідності відповідати за підтримку серверів. У наступному кодовому фрагменті описано створення трьох окремих Lambda функцій, кожна з яких виконує специфічну роль у системі.

```
users_lambda_function = _lambda.Function(
    self,
    'GetUsersLambdaFunction',
    runtime=_lambda.Runtime.PYTHON_3_8,
    code=_lambda.Code.from_asset('lambdas/lambda_users'),
    handler='handler.handler',
    role=lambda_shared_role,
)
chats_get_lambda_function = _lambda.Function(
    self,
    'GetChatLambdaFunction',
    runtime=_lambda.Runtime.PYTHON_3_8,
```

```
    code=_lambda.Code.from_asset('lambdas/lambda_chats_get'),
    handler='handler.handler',
    role=lambda_shared_role,

)
chats_post_lambda_function = _lambda.Function(
    self,
    'PostChatLambdaFunction',
    runtime=_lambda.Runtime.PYTHON_3_8,
    code=_lambda.Code.from_asset('lambdas/lambda_chats_post'),
    handler='handler.handler',
    role=lambda_shared_role,
)
```

Тут створюються три Lambda функції в AWS за допомогою AWS CDK: GetUsersLambdaFunction, GetChatLambdaFunction, і PostChatLambdaFunction. Кожна функція використовує Python 3.8 як середовище виконання і визначає місце розташування коду та обробника подій. Вони використовують спільну IAM роль для доступу до інших AWS сервісів, що забезпечує безпеку та гнучкість у виконанні функцій. Кожна з цих Lambda функцій має своє призначення і використовується для різних аспектів обробки даних чату, демонструючи гнучкість та масштабованість AWS Lambda в контексті розробки хмарних додатків.

1. **GetUsersLambdaFunction**: Ця функція призначена для отримання інформації про користувачів. Вона використовує Python 3.8 як середовище виконання і залучає свій код з локального репозиторію в директорії **lambdas/lambda_users**. Обробником подій є функція **handler** в модулі **handler**.
2. **GetChatLambdaFunction**: Функція призначена для отримання даних чату. Також використовує Python 3.8 і залучає код із **lambdas/lambda_chats_get**. Обробником є та сама функція **handler** в модулі **handler**.
3. **PostChatLambdaFunction**: Ця функція відповідає за прийом і обробку повідомлень чату. Як і інші, вона використовує Python 3.8 і код з **lambdas/lambda_chats_post**.

Amazon Cognito є ключовим сервісом AWS, який надає можливості управління ідентифікацією користувачів та доступом до додатків. Використання Cognito значно спрощує процеси аутентифікації, авторизації та управління

користувачами для розробників додатків. Опис даного сервіса наведено наступним фрагментом коду.

```
user_pool = cognito.UserPool(
    self,
    'SkyMessageUserPool',
    user_pool_name='SkyMessageUserPool',
    self_sign_up_enabled=True,
    sign_in_aliases=cognito.SignInAliases(email=True, username=True),
)
app_client = user_pool.add_client("MyUserPoolAppClient",
                                  auth_flows=cognito.AuthFlow(
                                      user_password=True,
                                      user_srp=True
                                  ),
                                  )
```

У представленому коді створюється користувацький пул Cognito, який надає функціональність для реєстрації, входу та управління профілями користувачів. SkyMessageUserPool ініціалізується з дозволом на самостійну реєстрацію користувачів та налаштуваннями для входу з використанням електронної пошти чи імені користувача. Додатково, до цього користувацького пулу додається клієнтський додаток. Він конфігурується з певними потоками аутентифікації, такими як перевірка паролю користувача та застосування протоколу SRP (Secure Remote Password). Це забезпечує додатковий рівень безпеки при вході користувачів у систему.

Amazon API Gateway є інструментом, який дозволяє розробникам створювати, публікувати, підтримувати, моніторити та захищати API. Він уповноважує розробників налаштовувати точки доступу до їхніх застосунків, забезпечуючи при цьому ефективне управління трафіком, автентифікацією користувачів та обробкою помилок. У наступному фрагменті коду конфігурується API Gateway з налаштуваннями для обробки запитів та взаємодії з Amazon Cognito для ідентифікації користувачів, що є критично важливим для безпеки та ефективності застосунку.

```
cors_options = apigw.CorsOptions(
        allow_origins=apigw.Cors.ALL_ORIGINS,
        allow_methods=apigw.Cors.ALL_METHODS,
        allow_headers=['*'],
        max_age=Duration.hours(10)
    )
api = apigw.RestApi(
    self,
    'SkyMessageAppApiGateway',
    rest_api_name='SkyMessageAppApiGateway',
    description='This service serves different Lambda functions',
```

```
    default_cors_preflight_options=cors_options
)

authorizer = apigw.CognitoUserPoolsAuthorizer(
    self,
    'SkyMessageCognitoAuthorizer',
    cognito_user_pools=[user_pool],
)
```

У цьому коді встановлюються параметри для API Gateway, включаючи налаштування CORS, що дозволяє запитам з будь-якого джерела та всім HTTP методам доступ до API. Також створюється API Gateway з назвою 'SkyMessageAppApiGateway', який забезпечує доступ до різних Lambda функцій. Додатково визначається авторизатор, що використовує Cognito User Pools, надаючи можливість автентифікації користувачів через Cognito.

Після створення основної структури API Gateway, наступний крок полягає в додаванні конкретних ресурсів та методів, які будуть керувати запитами. У цьому фрагменті коду створюється ресурс chats в API Gateway та налаштовується модель для обробки POST-запитів до цього ресурсу.

```
chats_post_resource_model = api.add_model(
    "ChatsGetPostModel",
    content_type="application/json",
    model_name="ChatsGetPostModel",
    schema=apigw.JsonSchema(
        schema=apigw.JsonSchemaVersion.DRAFT4,
        title="ChatsGetPostModel",
        type=apigw.JsonSchemaType.OBJECT,
        properties={
            "sender_receiver": apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
            "timestamp": apigw.JsonSchema(type=apigw.JsonSchemaType.INTEGER),
            "sender": apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
            "receiver": apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
            "message": apigw.JsonSchema(type=apigw.JsonSchemaType.STRING)
        },
        required=["sender_receiver", "timestamp", "sender", "receiver", "message"]
    )
)
chats_post_lambda_integration = apigw.LambdaIntegration(
    chats_post_lambda_function,
    request_templates={
        "application/json": """
                    #set($inputRoot = $input.path('$'))
                    {
                      "sender_receiver": "$inputRoot.sender_receiver",
                      "timestamp": "$inputRoot.timestamp",
                      "sender": "$inputRoot.sender",
                      "receiver": "$inputRoot.receiver",
                      "message": "$inputRoot.message"
                    }
                """
    }
)
chats_resource.add_method(
    'POST',
    chats_post_lambda_integration,
    authorizer=authorizer,
    request_models={
        "application/json": chats_post_resource_model
```

```
    }
)
```

Модель ChatsGetPostModel визначає структуру даних, яка очікується у запитах і відповідях, включаючи поля, такі як sender_receiver, timestamp, sender, receiver та message. Lambda інтеграція chats_post_lambda_integration встановлює зв'язок між цим ресурсом та відповідною Lambda функцією. Це означає, що при надсиланні POST-запиту до ресурсу chats, запит буде оброблятися вказаною Lambda функцією. Код також визначає, як формувати тіло запиту в JSON форматі. Наявність авторизатора authorizer, який використовує Cognito User Pools, забезпечує, що доступ до методу POST ресурсу chats мають лише аутентифіковані користувачі. Це важлива частина забезпечення безпеки API, оскільки вона контролює, хто може надсилати дані до системи. В цілому, цей код є ключовим елементом у створенні безпечного та ефективного API для обробки даних чату, забезпечуючи важливий зв'язок між фронтендом застосунку та його серверною логікою.

Після налаштування API Gateway та авторизації, необхідно забезпечити безпечне зберігання конфіденційних даних, таких як облікові дані для бази даних. У наступному фрагменті коду можна побачити використання AWS Secrets Manager для створення секрету, який зберігатиме облікові дані до DynamoDB.

```
db_credentials_secret = secretsmanager.Secret(
    self,
    'DBCredentialsSecret',
    secret_name="sky_message/dynamodb_cred",
    generate_secret_string=secretsmanager.SecretStringGenerator(
        secret_string_template='{"username": "dbmaster"}',
        generate_string_key='password',
        exclude_punctuation=True,
    )
)
```

Створюється секрет з назвою "sky_message/dynamodb_cred", і для цього використовується генератор рядків секретів, який формує JSON об'єкт з користувачем 'dbmaster' та автоматично генерованим паролем. Цей секрет може бути використаний Lambda функціями, які мають відповідні політики IAM для доступу до цих даних. Цей підхід є кращою практикою управління конфіденційною інформацією, оскільки він запобігає необхідності жорстко

закодовувати чутливі дані безпосередньо у вихідному коді, що забезпечує безпечну та ефективну роботу з даними, мінімізуючи ризик небажаного доступу.

На завершення стеку, код використовує CfnOutput для забезпечення зручного доступу до ключової інформації про створені ресурси.

```
CfnOutput(self, 'DBSecretARN', value=db_credentials_secret.secret_arn)
CfnOutput(self, "DistributionDomainName",
value=distribution.distribution_domain_name)
CfnOutput(self, "UserPoolAppClientId", value=app_client.user_pool_client_id)
CfnOutput(self, "DynamoTableName", value=table.table_name)
```

Після виконання деплою AWS CDK стеку, система надає логи, які включають ключову інформацію про створені ресурси (рис. 3.17). Для деплою стеку спочатку потрібно запустити відповідну команду в AWS CDK, яка ініціює процес розгортання. Після завершення, у логах з'являється інформація, така як ARN секрету у Secrets Manager, доменне ім'я CloudFront, назва таблиці DynamoDB, URL API Gateway та ідентифікатор клієнта користувацького пулу Cognito. Ці дані є важливими для подальшої інтеграції та управління розгорнутими ресурсами.

Рисунок 3.17 – Повідомлення в консолі після успішного розгортання

Після успішних повідомлень в консолі, уся інформація про розгортання стеку буде доступна в сервісі AWS CloudFormation.

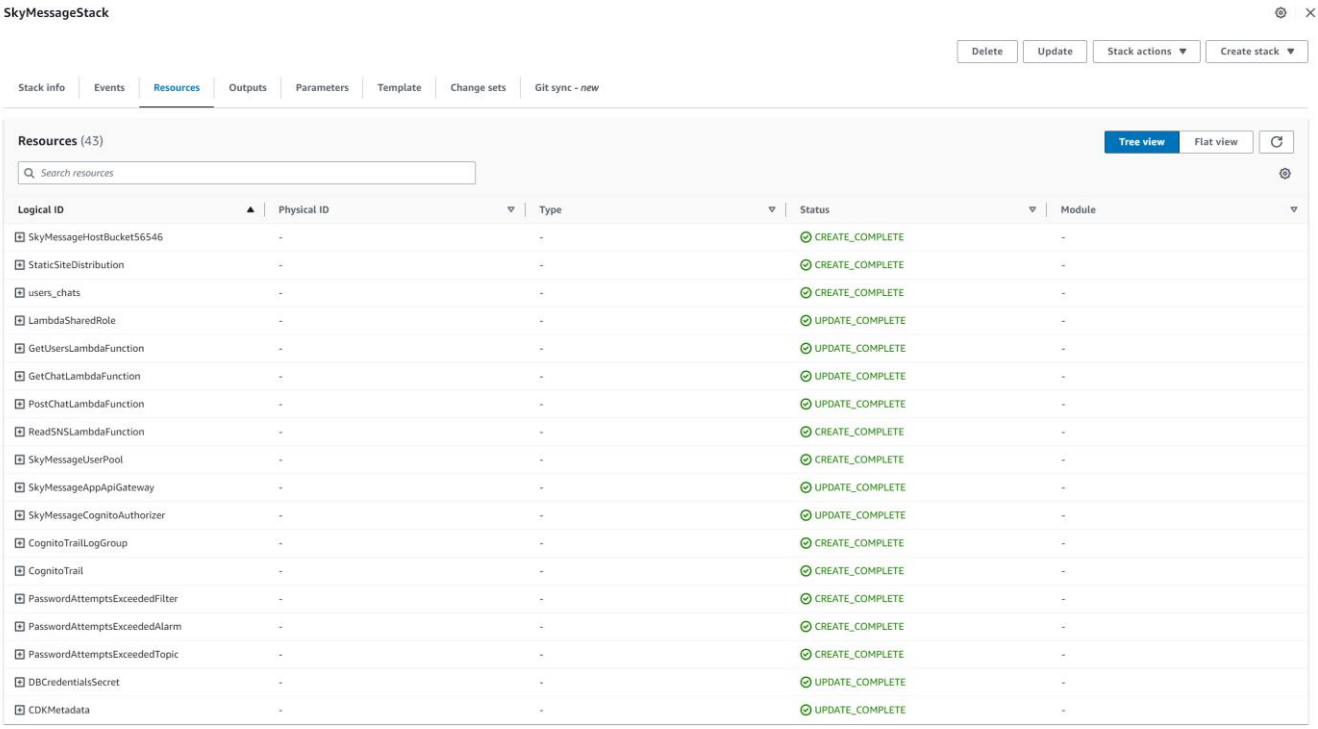


Рисунок 3.18 – Ресурси, розгортуні за допомогою AWS CloudFormation

На зображенні консолі управління AWS відображено стек 'SkyMessageStack', який є частиною сервісу AWS CloudFormation. Цей стек включає низку ресурсів, необхідних для функціонування хмарного додатку, включаючи конфігурації для сховища, обробки запитів, аутентифікації користувачів та комунікацій. Кожен ресурс має статус 'CREATE_COMPLETE' або 'UPDATE_COMPLETE', що підтверджує їх успішне розгортання.

Завершення цього процесу з використанням AWS CDK та Python демонструє ефективність цих інструментів для створення та управління хмарною інфраструктурою, підкреслюючи гнучкість та масштабованість хмарних рішень AWS.

4 ЕКОНОМІЧНА ЧАСТИНА

Для успішного впровадження науково-технічної розробки критично важливо, щоб вона відповідала сучасним вимогам науково-технічного прогресу та враховувала економічні аспекти. Надання оцінки економічної ефективності результатів науково-дослідної роботи є важливою частиною цього процесу. Дослідження, яке представлене у магістерській роботі і присвячене розробці та вивченню "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків", віднесено до науково-технічних робіт, спрямованих на виведення на ринок. Рішення про комерціалізацію розробки може бути прийняте протягом проведення самої роботи, відкриваючи можливості для подальшого виведення на ринок. Цей напрямок визначається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Однак для успішної реалізації цього процесу ключовим є залучення зацікавленого інвестора, який виявить інтерес до втілення даного проекту, і переконання його у доцільності інвестування у цю розробку. З метою досягнення цього завдання були визначені такі етапи виконання робіт:

1. Проведення комерційного аудиту науково-технічної розробки, включаючи визначення науково-технічного рівня та комерційного потенціалу.
2. Розрахунок витрат на реалізацію науково-технічної розробки.
3. Проведення розрахунку економічної ефективності впровадження та комерціалізації науково-технічної розробки для потенційного інвестора, а також обґрунтування економічної доцільності комерціалізації з точки зору інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" є розробити підсистему керування безпекою, яка здатна гарантувати застосунки у хмарному

середовищі високий рівень захищеності, враховуючи специфіку та особливості хмарних платформ.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 4.1 [24].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					

8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці. Для оцінки науково-технічного рівня і комерційного потенціалу розробки експертами було запрошено трьох незалежних експертів Вінницького національного технічного університету кафедри «Захисту інформації»: Кондратенко Наталія Романівна, к. т. н., професор, Дудатьєв Андрій Веніамінович, к. т. н., доцент, Войтович Олеся Петрівна, к. т. н., доцент.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Кондратенко Н. Р.	Дудатьєв А. В.	Войтович О. П.
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	4	4	5
2. Ринкові переваги (наявність аналогів)	2	3	3
3. Ринкові переваги (ціна продукту)	4	4	3
4. Ринкові переваги (технічні властивості)	3	3	4
5. Ринкові переваги (експлуатаційні витрати)	2	2	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	4	5	5
11. Практична здійсненність (термін реалізації)	3	4	5
12. Практична здійсненність (розробка документів)	4	5	4
Сума балів	$СБ_1$=39	$СБ_2$=44	$СБ_3$=45
Середньоарифметична сума балів $СБ_c$	42,7		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 4.3 [24].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий

31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" становить 43 бали, що, відповідно до таблиці 4.3 рівень комерційного потенціалу розробки високий, що свідчить про комерційну важливість проведення даних досліджень.

Магістерська кваліфікаційна робота "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок, тобто при цьому відбувається комерціалізація науково-технічної розробки. Цей напрямок є для нас пріоритетним, оскільки результатами розробки можуть користуватися не тільки самі розробники, а й інші споживачі, отримуючи при цьому суттєвий економічний ефект.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків", під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [24]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 20500 \cdot 40 / 21 = 37273 \text{грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	20500	931,8	5	4659
Інженер-програміст 1-ї категорії	17000	772,7	40	30909
Всього				35568

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (4.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [Козловський, Лесько, Кавецький];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1 \cdot 1{,}65 / (21 \cdot 8) = 65{,}8$ грн.
$З_{р1} = 65{,}8\ 2 = 131{,}6$ грн.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1. Підготовка робочого місця інженера-розробника програмного забезпечення	2	1	65,8	131,6
2. Інсталяція програмного забезпечення середовища розробки	3	3	88,8	266,5
3. Формування кодів програмних блоків	2	5	111,9	223,7
4. Контроль проходження програмних експериментів	6	2	72,4	434,3
Всього				1056,1

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (35568+ 1056,1) · 11 / 100% = 4028,68 грн.

% = 8943,66 грн.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків".

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.6)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.7 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн

Папір канцелярський офісний (А4)	175	1	175
Папір для заміток (А5)	110	1	110
Начиння канцелярське	190	1	190
Органайзер офісний	183	1	183
Картридж для принтера	950	1	950
Всього			1608
З врахуванням коефіцієнта транспортування			1768,8

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна система підтримки підприємств малого бізнесу у сфері послуг».

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн.,} \qquad (4.7)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – ціна комплектуючих *i*-го виду, грн.;

$К_i$ – коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$;

n – кількість видів комплектуючих.

Зроблені розрахунки бажано звести до таблиці:

Таблиця 4.8 – Витрати на комлектуючі

Найменування комплектувальних	Кількість	Ціна за штуку, грн.	Сума, грн.
КОНЦЕНТРАТОР GEMBIRD USB-A TO USB 3.1 GEN1 (5 GBPS), 3 X USB 2.0	1	300	300
КАБЕЛЬ ДЛЯ ПЕРЕДАЧІ ДАНИХ SATA III 0.3M CABLEXPERT	1	45	45

Флеш память USB Kingston DataTraveler Exodia Onyx 64GB USB 3.2 Gen 1 Black	1	170	170
Всього з врахування коефіцієнт транспортних витрат			566,50

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \qquad (4.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (50000\ 2) / (2 \cdot 12) = 4166{,}67$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер	50000	2	2	4166,67
Робоче місце розробника ПЗ	185700	20	2	1547,50
Всього				5714,17

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.11)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (35568+ 1056,1) · 50 / 100% = 18312,16 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 8000,00 грн;

$\pm\Delta Ц_o$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою **[]**:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.15)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 40\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1 \cdot 500 + 8000 \cdot 400) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 580895,51$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1 \cdot 500 + 8000 \cdot (400 + 300)) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 1016917,7$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1 \cdot 500 + 8000 \cdot (400 + 300 + 200)) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) = 1307322,7$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.16)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =580895,51 /(1+0,18)^1+1016917,7 /(1+0,18)^2+1307322,7 /(1+0,18)^3=

= 1950343,78 грн.

Величина початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \qquad (4.17)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 240494,67 грн.

$PV = k_{інв} \cdot ЗВ$ = 2 240494,67 = 480989,35 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \qquad (4.18)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 1950343,78 грн;

PV – теперішня вартість початкових інвестицій, 480989,35 грн.

$E_{абс} = ПП - PV$ = 1950343,78 - 480989,35 = 1469354,43 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (4.19)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+1469354,43/480989,35\ )^{1/3} - 1 = 0,92.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \qquad (4.20)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$= 0,1+0,25 = 0,35 < 0,92 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \qquad (4.21)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,92 = 1,1 р.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків" становить 43 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий.

Також термін окупності становить 1,1 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою "Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків".

ВИСНОВКИ

Виконання магістерської роботи передбачало глибоке дослідження в сфері безпеки хмарних обчислень. Було проведено ґрунтовний аналіз сучасного стану та перспектив розвитку автоматизованих систем управління безпекою додатків у хмарному середовищі. Розглянуто різні аспекти хмарних технологій, включаючи їх еволюцію, ключові виклики та ризики, пов'язані з безпекою додатків. Виявлено, що ефективне управління безпекою в хмарному середовищі вимагає комплексного підходу, який включає не лише технічні аспекти, але й організаційні та правові моменти. Було проаналізовано ряд завдань та визначено ключові вимоги, які сприяли досягненню мети. Проведено детальний аналіз існуючих методів та викликів, пов'язаних з безпекою додатків у хмарних середовищах.

Під час розробки автоматизованої підсистеми управління безпекою, детально описано архітектуру системи, її основні компоненти, алгоритми взаємодії та методи автоматизації процесів управління безпекою. Підкреслено, що такий підхід забезпечує гнучкість та масштабованість системи, що є критично важливим у швидкозмінному хмарному середовищі. Аналіз ефективності підсистеми показав, що вона здатна ефективно виявляти та реагувати на різноманітні загрози безпеки, забезпечуючи високий рівень захисту додатків. Важливою частиною роботи стала імплементація та аналіз ефективності підсистеми у реальних хмарних середовищах.

Економічний розділ роботи включає аналіз витрат на розробку та впровадження системи, а також оцінку її потенційної окупності. Проведений аналіз підтверджує економічну доцільність розробленої системи, враховуючи зростаючу потребу в ефективних інструментах управління безпекою в хмарних середовищах. Зазначено, що інвестиції в такі системи є стратегічно важливими для компаній, які прагнуть забезпечити надійний захист своїх додатків та даних у хмарі.

Ця робота відкриває нові можливості для подальшого розвитку технологій управління безпекою хмарних додатків. Її результати можна використовувати як

основу для розробки більш надійних та ефективних систем, що забезпечать кращий захист даних та застосунків у хмарному середовищі. Окрім технічного внеску, робота також має важливе практичне значення, оскільки вона сприяє підвищенню рівня безпеки в цифровому світі.

У цілому, магістерська робота є важливим вкладом у сферу хмарних обчислень та безпеки. Результати дослідження є значущими для розвитку нових напрямків у цій галузі, надаючи важливі знання та інструменти для подальшого вдосконалення хмарних технологій та методів захисту даних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хмарні обчислення: приклади використання та переваги. URL: https://onbiz.biz/cloud-usage-examples/ (дата звернення 12.11.2023).

2. 9 фактів про хмарні обчислення, які ви могли не знати. URL: https://training.epam.ua/ua/blog/559 (дата звернення 12.11.2023).

3. Історія хмарних обчислень. URL: https://nachasi.com/tech/2017/09/26/istoriya-hmarnyh-obchyslen/ (дата звернення 12.11.2023).

4. Поняття хмарних обчислень: основні моделі та характеристики. URL: https://onbiz.biz/cloud-computing-models/ (дата звернення 12.11.2023)..

5. Information Leakage | CQR. URL: https://cqr.company/ua/web-vulnerabilities/information-leakage/ (дата звернення 12.11.2023)..

6. Хміль Н. А. Зарубіжний і вітчизняний досвід інтеграції хмарних технологій у педагогічний процес вищого навчального закладу. Інформаційні технології і засоби навчання. URL: https://journal.iitta.gov.ua/index.php/itlt/article/download/1279/984 (дата звернення 12.11.2023)..

7. Що таке хмарні обчислення? – Smartik. URL: https://smartik.kiev.ua/shcho-take-khmarni-obchyslennia/ (дата звернення 12.11.2023)..

8. Види хмарних сервісів та приклади їх використання. URL: https://denovo.ua/blog/vidi-hmarnih-servisiv-yakij-obrati-ta-oglyad-hmarnih-provajd-8 (дата звернення 12.11.2023)..

9. AWS vs Azure vs GCP: The big 3 cloud providers compared. URL: https://www.pluralsight.com/resources/blog/cloud/aws-vs-azure-vs-gcp-the-big-3-cloud-providers-compared (дата звернення 12.11.2023).

10. Public Cloud Services Comparison. URL: https://comparecloud.in/ (дата звернення 12.11.2023).

11. Cloud computing with AWS. URL: https://aws.amazon.com/what-is-aws/?nc1=h_ls (дата звернення 12.11.2023).

12. Overview of Amazon Web Services. URL: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html (дата звернення 12.11.2023).

13. ISO and CSA STAR Certified. URL: https://aws.amazon.com/compliance/iso-certified/?nc1=h_ls (дата звернення 12.11.2023).

14. Global Infrastructure Regions & AZs. URL: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ (дата звернення 12.11.2023).

15. Six advantages of cloud computing. URL : https://docs.aws.amazon.com/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html (дата звернення 12.11.2023).

16. Cloud Security – Amazon Web Services. URL : https://aws.amazon.com/security/?nc1=h_ls (дата звернення 12.11.2023).

17. 14 Best AWS Security Tools and Services for 2023. URL : https://sonraisecurity.com/blog/aws-security-tools/ (дата звернення 12.11.2023).

18. Shared Responsibility Model. URL : https://aws.amazon.com/compliance/shared-responsibility-model/?nc1=h_ls (дата звернення 12.11.2023).

19. AWS Identity and Access Management. URL : https://aws.amazon.com/iam/?nc1=h_ls (дата звернення 12.11.2023).

20. What is Amazon VPC?. URL : https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (дата звернення 12.11.2023).

21. Connect to the internet or other networks using NAT devices. URL : https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat.html (дата звернення 12.11.2023).

22. What is Amazon Cognito? . URL : https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html (дата звернення 12.11.2023).

23. What is AWS Secrets Manager?. URL : https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html (дата звернення 12.11.2023).

24. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків.

Автор роботи: Радзіховський Дмитро Юрійович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,64 %. Схожість – 1,36 %.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Дмитро РАДЗІХОВСЬКИЙ
(підпис)

Керівник роботи _____ Галина ШЕЛЕПАЛО

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Система автоматизованого керування безпекою застосунків у хмарному середовищі. Частина 2. Підсистема керування безпекою застосунків.

Автор роботи: Радзіховський Дмитро Юрійович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,64 %. Схожість – 1,36 %.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Валентина КАПЛУН

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Дмитро РАДЗІХОВСЬКИЙ

Керівник роботи _____ Галина ШЕЛЕПАЛО

Додаток Б Лістинг програми

```
import json

from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_apigateway as apigw,
    aws_cognito as cognito,
    aws_s3 as s3,
    aws_dynamodb as dynamodb,
    aws_sns as sns,
    aws_iam as iam,
    aws_logs as logs,
    aws_cloudwatch as cloudwatch,
    aws_cloudtrail as cloudtrail,
    aws_cloudfront as cloudfront,
    aws_secretsmanager as secretsmanager,
    RemovalPolicy,
    CfnOutput,
    Duration,
)

from aws_cdk.aws_cloudfront_origins import S3Origin
from aws_cdk.aws_cloudwatch_actions import SnsAction
from aws_cdk.aws_s3 import BlockPublicAccess
from aws_cdk.aws_sns_subscriptions import LambdaSubscription
from constructs import Construct


class SkyMessageStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        bucket = s3.Bucket(
            self,
            "SkyMessageHostBucket56546",
            website_index_document="index.html",
            public_read_access=True,
            block_public_access=BlockPublicAccess(
                block_public_acls=False,
                block_public_policy=False,
                ignore_public_acls=False,
                restrict_public_buckets=False
            )
        )

        distribution = cloudfront.Distribution(
            self,
            "StaticSiteDistribution",
            default_behavior=cloudfront.BehaviorOptions(
                origin=S3Origin(bucket),

viewer_protocol_policy=cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS
            ),
            default_root_object="index.html"
        )

        table = dynamodb.Table(
            self, 'users_chats',
            partition_key=dynamodb.Attribute(
                name='sender_receiver',
```

```
                type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name='timestamp',
                type=dynamodb.AttributeType.NUMBER
            ),
            billing_mode=dynamodb.BillingMode.PROVISIONED,
            read_capacity=1,
            write_capacity=1,
            removal_policy=RemovalPolicy.RETAIN
        )

        lambda_shared_role = iam.Role(
            self,
            "LambdaSharedRole",
            assumed_by=iam.ServicePrincipal("lambda.amazonaws.com"),
            managed_policies=[
                iam.ManagedPolicy.from_aws_managed_policy_name("service-
role/AWSLambdaBasicExecutionRole"),
                iam.ManagedPolicy.from_aws_managed_policy_name("service-
role/AWSLambdaBasicExecutionRole"),

iam.ManagedPolicy.from_aws_managed_policy_name("AmazonCognitoReadOnly"),

iam.ManagedPolicy.from_aws_managed_policy_name("AmazonSESFullAccess")
            ]
        )

        dynamodb_policy = iam.PolicyStatement(
            actions=["dynamodb:GetItem", "dynamodb:PutItem",
"dynamodb:UpdateItem", "dynamodb:DeleteItem",
                     "dynamodb:Query", "dynamodb:Scan"],
            resources=["*"]
        )
        secretsmanager_policy = iam.PolicyStatement(
            actions=["secretsmanager:GetSecretValue"],
            resources=["*"]
        )

        lambda_shared_role.add_to_policy(dynamodb_policy)
        lambda_shared_role.add_to_policy(secretsmanager_policy)

        users_lambda_function = _lambda.Function(
            self,
            'GetUsersLambdaFunction',
            runtime=_lambda.Runtime.PYTHON_3_8,
            code=_lambda.Code.from_asset('lambdas/lambda_users'),
            handler='handler.handler',
            role=lambda_shared_role,
        )


        chats_get_lambda_function = _lambda.Function(
            self,
            'GetChatLambdaFunction',
            runtime=_lambda.Runtime.PYTHON_3_8,
            code=_lambda.Code.from_asset('lambdas/lambda_chats_get'),
            handler='handler.handler',
            role=lambda_shared_role,

        )

        chats_post_lambda_function = _lambda.Function(
            self,
```

```
            'PostChatLambdaFunction',
            runtime=_lambda.Runtime.PYTHON_3_8,
            code=_lambda.Code.from_asset('lambdas/lambda_chats_post'),
            handler='handler.handler',
            role=lambda_shared_role,
        )

        sns_lambda_function = _lambda.Function(
            self,
            'ReadSNSLambdaFunction',
            runtime=_lambda.Runtime.PYTHON_3_8,
            code=_lambda.Code.from_asset('lambdas/lambda_sns'),
            handler='handler.handler',
            role=lambda_shared_role,
        )


        user_pool = cognito.UserPool(
            self,
            'SkyMessageUserPool',
            user_pool_name='SkyMessageUserPool',
            self_sign_up_enabled=True,
            sign_in_aliases=cognito.SignInAliases(email=True, username=True),
        )

        app_client = user_pool.add_client("MyUserPoolAppClient",
                                          auth_flows=cognito.AuthFlow(
                                              user_password=True,
                                              user_srp=True
                                          ),
                                          )
        cors_options = apigw.CorsOptions(
                allow_origins=apigw.Cors.ALL_ORIGINS,
                allow_methods=apigw.Cors.ALL_METHODS,
                allow_headers=['*'],
                max_age=Duration.hours(10)
            )
                api = apigw.RestApi(
            self,
            'SkyMessageAppApiGateway',
            rest_api_name='SkyMessageAppApiGateway',
            description='This service serves different Lambda functions',
            default_cors_preflight_options=cors_options
        )


        authorizer = apigw.CognitoUserPoolsAuthorizer(
            self,
            'SkyMessageCognitoAuthorizer',
            cognito_user_pools=[user_pool],
        )

        chats_resource = api.root.add_resource('chats',
default_cors_preflight_options=cors_options)
        users_resource = api.root.add_resource('users',
default_cors_preflight_options=cors_options)

        chats_get_lambda_integration = apigw.LambdaIntegration(
            chats_get_lambda_function,
            request_templates={
                "application/json": json.dumps({
                    "sender_receiver": "$input.params('sender_receiver')",
                })
            }
```

```
        )

        chats_resource.add_method(
            'GET',
            chats_get_lambda_integration,
            authorizer=authorizer,
            request_parameters={
                "method.request.querystring.sender_receiver": True
            }
        )

        chats_post_resource_model = api.add_model(
            "ChatsGetPostModel",
            content_type="application/json",
            model_name="ChatsGetPostModel",
            schema=apigw.JsonSchema(
                schema=apigw.JsonSchemaVersion.DRAFT4,
                title="ChatsGetPostModel",
                type=apigw.JsonSchemaType.OBJECT,
                properties={
                    "sender_receiver":
apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
                    "timestamp":
apigw.JsonSchema(type=apigw.JsonSchemaType.INTEGER),
                    "sender":
apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
                    "receiver":
apigw.JsonSchema(type=apigw.JsonSchemaType.STRING),
                    "message":
apigw.JsonSchema(type=apigw.JsonSchemaType.STRING)
                },
                required=["sender_receiver", "timestamp", "sender", "receiver",
"message"]
            )
        )

        chats_post_lambda_integration = apigw.LambdaIntegration(
            chats_post_lambda_function,
            request_templates={
                "application/json": """
                            #set($inputRoot = $input.path('$'))
                            {
                              "sender_receiver": "$inputRoot.sender_receiver",
                              "timestamp": "$inputRoot.timestamp",
                              "sender": "$inputRoot.sender",
                              "receiver": "$inputRoot.receiver",
                              "message": "$inputRoot.message"
                            }
                        """
            }
        )

        chats_resource.add_method(
            'POST',
            chats_post_lambda_integration,
            authorizer=authorizer,
            request_models={
                "application/json": chats_post_resource_model
            }
        )

        users_get_lambda_integration =
apigw.LambdaIntegration(users_lambda_function)
```

```
        users_get_response_model = api.add_model(
            "UsersGetResponseModel",
            content_type="application/json",
            model_name="UsersGetResponseModel",
            schema=apigw.JsonSchema(
                schema=apigw.JsonSchemaVersion.DRAFT4,
                title="UsersGetResponseModel",
                type=apigw.JsonSchemaType.ARRAY,
                items=apigw.JsonSchema(type=apigw.JsonSchemaType.STRING)
            )
        )

        users_resource.add_method(
            "GET",
            users_get_lambda_integration,
            authorizer=authorizer,
            method_responses=[{
                "statusCode": "200",
                "responseModels": {
                    "application/json": users_get_response_model
                }
            }]
        )

        log_group = logs.LogGroup(
            self, "CognitoTrailLogGroup",
            retention=logs.RetentionDays.ONE_YEAR,
            removal_policy=RemovalPolicy.DESTROY
        )

        # Создание CloudTrail с интеграцией CloudWatch Logs
        trail = cloudtrail.Trail(
            self, "CognitoTrail",
            is_multi_region_trail=True,
            include_global_service_events=True,
            management_events=cloudtrail.ReadWriteType.ALL,
            cloud_watch_log_group=log_group,
            cloud_watch_logs_retention=logs.RetentionDays.ONE_YEAR
        )

        filter_pattern = logs.FilterPattern.literal("{$.errorMessage =
\"Password attempts exceeded\"}")
        metric_filter = logs.MetricFilter(self,
"PasswordAttemptsExceededFilter",
                                          log_group=log_group,
                                          filter_pattern=filter_pattern,
                                          metric_namespace="SkyMessageApp",

metric_name="PasswordAttemptsExceeded",
                                          metric_value="1")


        alarm = cloudwatch.Alarm(self, "PasswordAttemptsExceededAlarm",
                                 metric=metric_filter.metric(),
                                 threshold=1,
                                 evaluation_periods=1,
                                 datapoints_to_alarm=1,

treat_missing_data=cloudwatch.TreatMissingData.NOT_BREACHING,

comparison_operator=cloudwatch.ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRES
HOLD)
```

```
        sns_topic = sns.Topic(self, "PasswordAttemptsExceededTopic")
                alarm.add_alarm_action(SnsAction(sns_topic))

        sns_topic.grant_publish(sns_lambda_function)


        sns_topic.add_subscription(LambdaSubscription(sns_lambda_function))


        db_credentials_secret = secretsmanager.Secret(
            self,
            'DBCredentialsSecret',
            secret_name="sky_message/dynamodb_credentials",
            generate_secret_string=secretsmanager.SecretStringGenerator(
                secret_string_template='{"username": "dbmaster"}',
                generate_string_key='password',
                exclude_punctuation=True,
            )
        )

        value=db_instance.db_instance_endpoint_address)
        CfnOutput(self, 'DBSecretARN', value=db_credentials_secret.secret_arn)
        CfnOutput(self, "DistributionDomainName",
value=distribution.distribution_domain_name)
        CfnOutput(self, "UserPoolAppClientId",
value=app_client.user_pool_client_id)
        CfnOutput(self, "DynamoTableName", value=table.table_name)
```

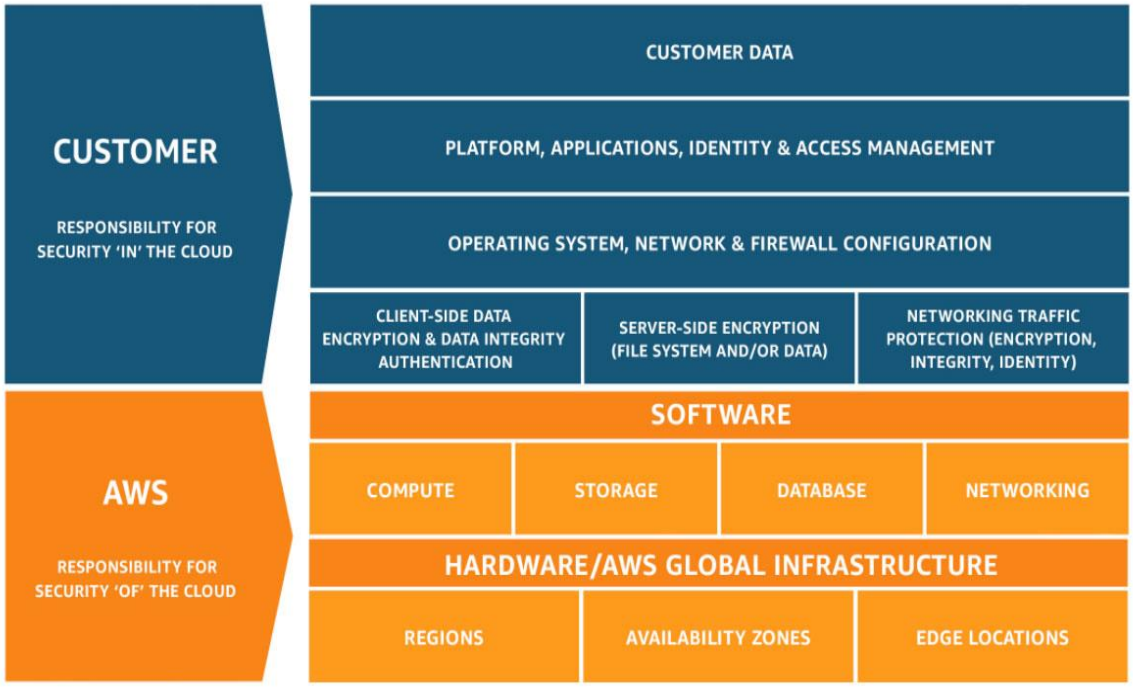
Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

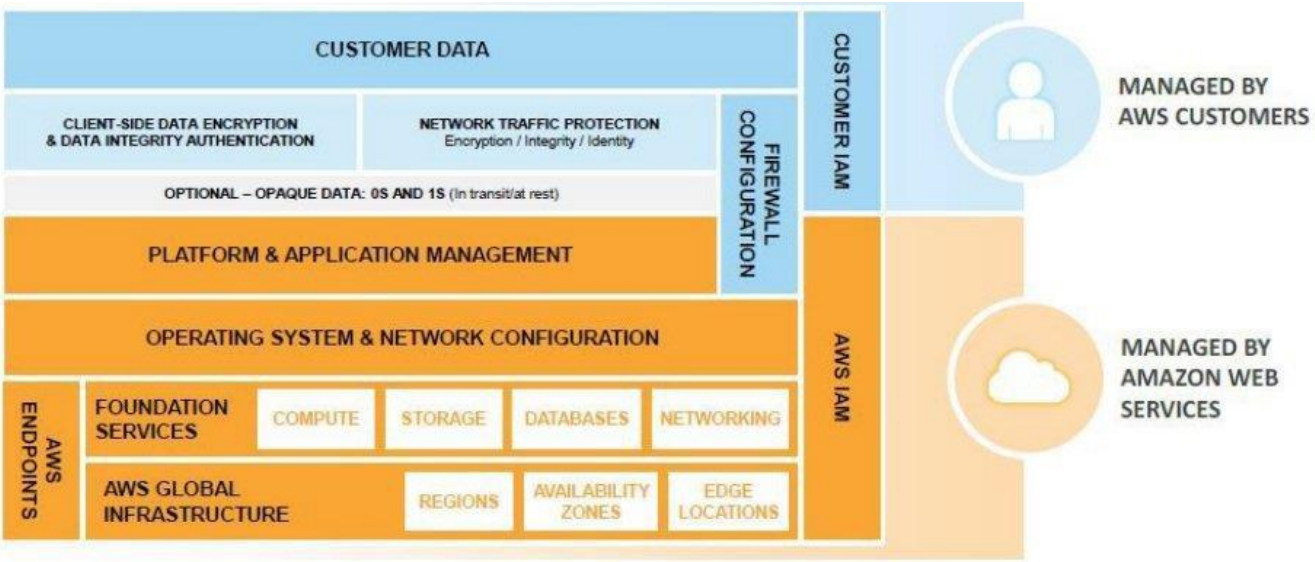
СИСТЕМА АВТОМАТИЗОВАНОГО КЕРУВАННЯ БЕЗПЕКОЮ ЗАСТОСУНКІВ У ХМАРНОМУ СЕРЕДОВИЩІ. ЧАСТИНА 2. ПІДСИСТЕМА КЕРУВАННЯ БЕЗПЕКОЮ ЗАСТОСУНКІВ

РЕГІОНИ РОЗПОДІЛУ РЕСУРСІВ AWS

МОДЕЛЬ СПІЛЬНОЇ ВІДПОВІДАЛЬНОСТІ AWS

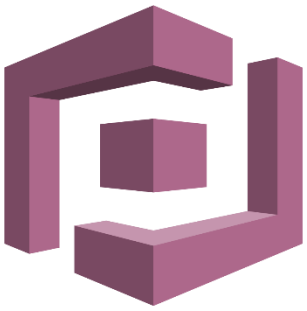


МОДЕЛЬ СПІЛЬНОЇ ВІДПОВІДАЛЬНОСТІ ДЛЯ КОНТЕЙНЕРНИХ ПОСЛУГ



СЕРВІСИ AWS ДЛЯ КЕРУВАННЯ БЕЗПЕКОЮ

AWS Secrets Manager



AWS Cognito

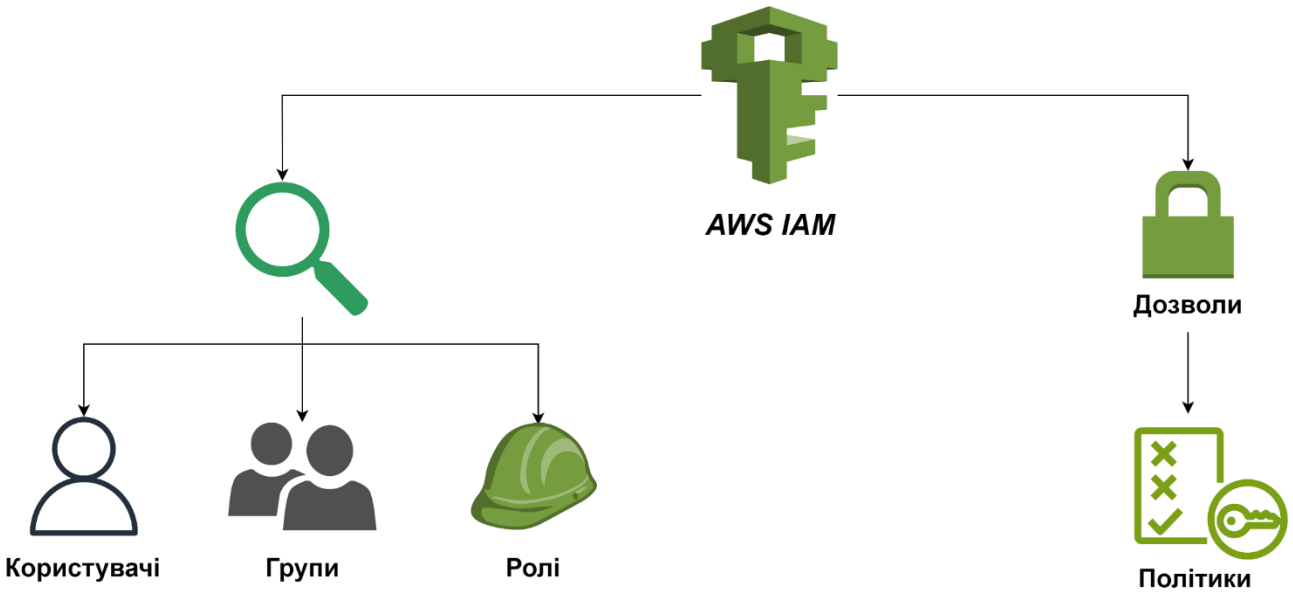
Віртуальна приватна мережа

ОПИС AWS IAM POLICY

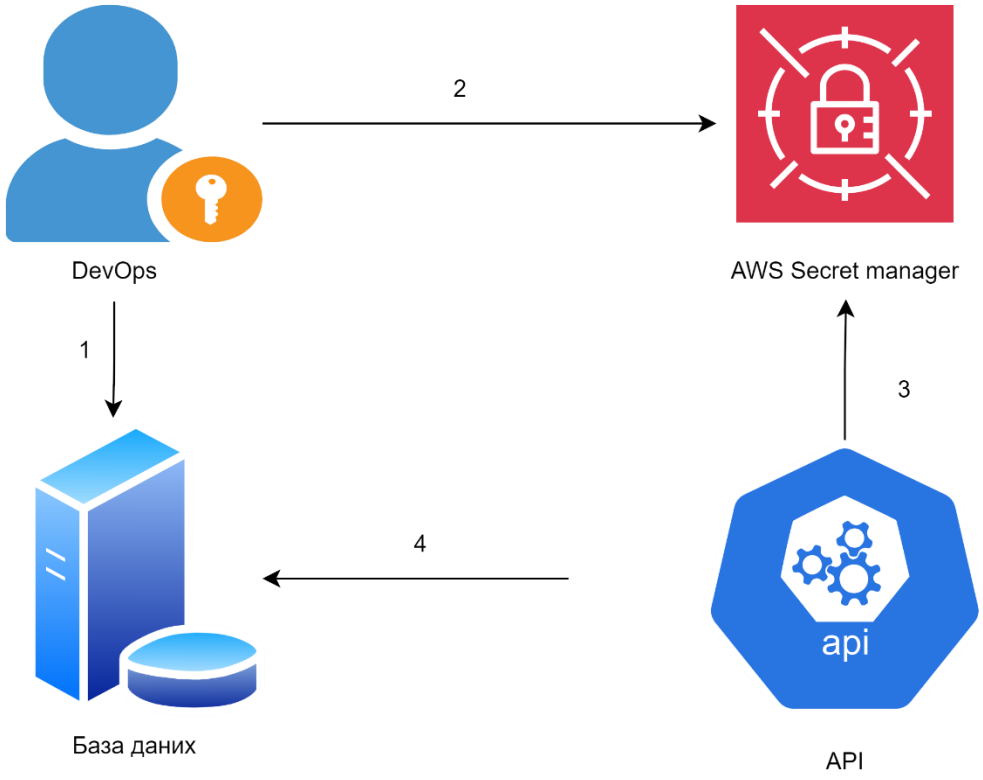
```
MySQSQueue:
  Type: 'AWS:SQS::Queue'

MyRole:
  Type: 'AWS::IAM::Role'
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
          Service: 'ec2.amazonaws.com'
          Action: 'sts:AssumeRole'
    Policies:
      - PolicyName: root
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action: 'sqs:SendMessage'
              Resource: !Sub ${MySQSQueue.Arn}
```

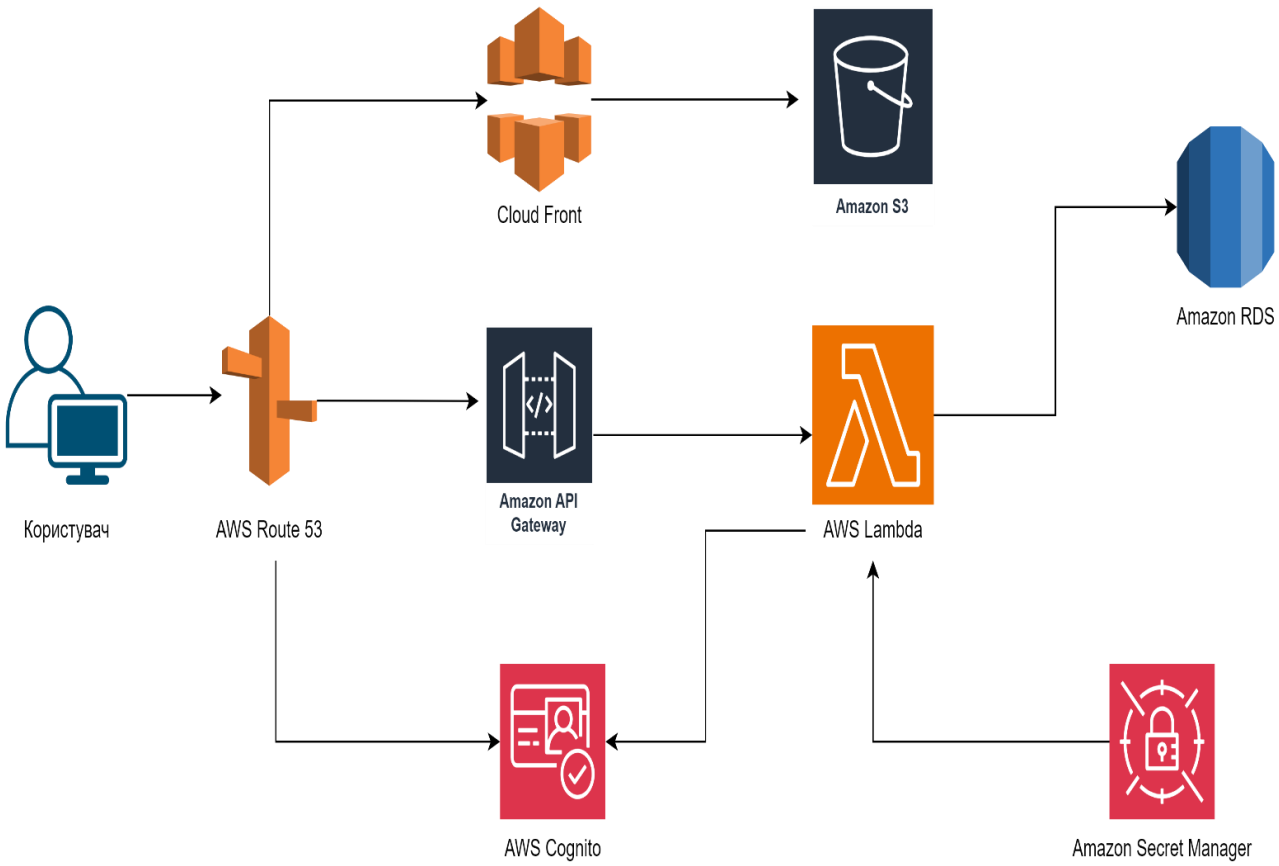
ОСНОВНІ КОМПОНЕНТИ AWS IAM



СЦЕНАРІЙ ВИКОРИСТАННЯ AWS SECRETS MANAGER



АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ



СТВОРЕНІ РЕСУРСИ ЗА ДОПОМОГОЮ AWS CLOUD FORMATION