Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет інфокомунікацій
(повна назва)

Кафедра інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Впровадження безсерверної архітектури для створення щоденного звіту про продажі
(тема)

Виконав:
здобувач 2 року навчання,
групи ІМІм-23-2
Довбах О.С.
(прізвище, ініціали)
Спеціальність
172 Електронні комунікації та радіотехніка
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
«Інформаційно-мережна інженерія»
(повна назва освітньої програми)
Керівник доц. Кривенко С.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2025 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____

Керівник_____

Харківський національний університет радіоелектроніки

Факультет інфокомунікацій
(повна назва)

Кафедра інформаційно-мережної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 172 «Телекомунікації та радіотехніка»
(код і повна назва)

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

Освітня програма «Інформаційно-мережна інженерія»
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Довбах Олексій Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи Впровадження безсерверної архітектури для створення щоденного звіту про продажі
затверджена наказом університету від 28 жовтня 2024р. № 1148 Ст_
2. Термін подання студентом роботи до екзаменаційної комісії 02 січня 2025р.
3. Вихідні дані до роботи: _____
Реалізувати безсерверну архітектуру для створення щоденного звіту про продажі, який міститиме наступне. Функцію Lambda у віртуальній приватній хмарі VPC, яка підключається до бази даних Amazon RDS із даними про продажі кафе. Функцію Lambda, яка створює та виконує звіт про продажі. Заплановану подію, яка щодня ініціює лямбда-функцію звіту про продажі. Мови реалізації моделі – Python3, HTML.
4. Перелік питань, що потрібно опрацювати в роботі: _____
аналіз сучасної архітектури; модель безсерверної архітектури; результати створення системи обліку запасів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) 16 слайдів у форматі PowerPoint _____

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Кривенко С.А.		02.01.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Ознайомлення із завданням. Уточнення ТЗ.*	*21.11*	вик
2	*Підбір літератури за темою роботи.*	*21.11-28.11*	вик
3	*Виконання розділу 1*	*29.11-04.12*	вик
4	*Виконання розділу 2*	*05.12-11.12*	вик
5	*Виконання розділу 3*	*12.12-28.12*	вик
7	*Оформлення презентаційного матеріалу, підготовка до захисту у ЕК*	*28.12-02.01*	вик

Дата видачі завдання 28 жовтня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Кривенко С.А
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка випускної магістерської кваліфікаційної роботи містить: 75 стор., 53 рис., 20 джерел.

AWS, AMAZON CLOUD9, AMAZON DYNAMO DB, AMAZON SIMPLE STORAGE SERVICE, AMAZON SIMPLE QUEUE SERVICE, AMAZON Simple NOTIFICATION SERVICE.

Об’єкт дослідження – процес автоматизації створення щоденного звіту про продажі за допомогою хмарних технологій.

Предмет дослідження – робота побудована навколо технологічних аспектів впровадження безсерверної архітектури для побудови системи створення звітів.

Впровадження безсерверної архітектури є ключовим аспектом сучасних систем, які потребують гнучкості, масштабованості та оптимізації витрат.

Метою даної роботи є створення роз’єднаної архітектури для веб-сайту кафе за допомогою Amazon SQS, SNS для розробки моделі дослідження можливостей ефективного масштабування, відмовостійкості, та швидких звітів цього веб-сайту.

Методи досліджень – сервіси Amazon Web Service.

ABSTRACT

The explanatory note of the final master's qualification work contains: 75 pages, 53 figures, 20 sources.

AWS, AMAZON CLOUD9, AMAZON DYNAMO DB, AMAZON SIMPLE STORAGE SERVICE, AMAZON SIMPLE QUERY SERVICE, AMAZON Simple NOTIFICATION SERVICE.

The object of research is the process of automating the creation of a daily sales report using cloud technologies.

Subject of research - the work is built around the technological aspects of implementing a serverless architecture for building a report generation system.

Implementing serverless architecture is a key aspect of modern systems that require flexibility, scalability, and cost optimization.

The purpose of this paper is to create a decoupled architecture for a cafe website using Amazon SQS, SNS to develop a model to study the possibilities of effective scalability, fault tolerance, and fast reporting of this website.

Research methods - Amazon Web Service services.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

AWS – Amazon Web Services – дочірня компанія Amazon.com, що надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки

EC2 – Amazon Elastic Compute Cloud – веб сервіс, котрий надає обчислювальні потужності в хмарі

HTML – HyperText Markup Language – мова розмітки гіпертексту

IDE – integrated development environment – Інтегроване середовище розробки

LAMP – абревіатура набору вільного програмного забезпечення з відкритим кодом, в який входять Linux, веб сервер Apache, MySQL та інтерпретатор Perl/PHP/Python — основні компоненти для побудови життєздатного багатоцільового веб сервера

URL – Uniform Resource Locator – єдиний вказівник на ресурс

S3 – Amazon Simple Storage Service – сервіс для зберігання даних у хмарі.

SNS – Amazon Simple Notification Service – сервіс для надсилання сповіщень у моделі публікація/підписка.

SQS – Amazon Simple Queue Service – сервіс для організації черг повідомлень.

Lambda – AWS Lambda – сервіс для виконання коду без управління серверами.

RDS – Amazon Relational Database Service – сервіс для управління реляційними базами даних.

VPC – Virtual Private Cloud – віртуальна приватна мережа для безпечного розгортання ресурсів.

DynamoDB – Amazon DynamoDB – безсерверна база даних NoSQL.

EventBridge – сервіс для управління подіями та інтеграції додатків.

IAM – Identity and Access Management – сервіс для управління правами доступу та ідентифікацією.

Cloud9 – AWS Cloud9 – інтегроване середовище розробки в хмарі.

ENI – Elastic Network Interface – віртуальний мережевий інтерфейс для мережевих ресурсів.

HTML – HyperText Markup Language – мова розмітки для створення веб-сторінок.

ВСТУП

Кваліфікаційна робота пов’язана з розв’язанням наступної бізнес проблеми. Архітектура кафе підтримує сотні тисяч користувачів. Однак системи кафе надто тісно пов’язані між собою. Важко ввести зміни в один рівень програми, не впливаючи на інші рівні. Наприклад, щоденні звіти про замовлення генеруються з того самого веб-сервера, який також обслуговує веб-сайт кафе для клієнтів. Крім того, власники кафе зазначили, що не отримують звичайний звіт о 17:00 по п’ятницях. Аналіз проблеми показав, що вікно запланованого технічного обслуговування збігається з часом, коли система звітування намагається створити звіт. Сучасна рекомендація для розв’язання проблеми – це роз’єднати архітектуру. Якщо перенести процес звітування в іншу систему, дані звітності не будуть втрачені, навіть якщо веб-сервер стане тимчасово недоступним. Крім того, необхідність створення звіту буде поставлена в чергу та оброблена[1].

Однак власники хочуть отримувати щоденні звіти електронною поштою про всі замовлення, розміщені на веб-сайті. Один власник хоче передбачити попит, щоб у майбутньому він міг спекти правильну кількість десертів (зменшуючи відходи). Інший власник хоче виявити будь-які закономірності в бізнесі кафе (аналітика). Розробник роз’єднаної архітектури налаштував завдання cron на екземплярі веб сервера, який надсилає ці щоденні повідомлення електронної пошти зі звітами про замовлення власникам. Однак завдання cron вимагає ресурсів і знижує продуктивність веб-сервера. В роботі пропонується модель для тримання неважливих для бізнесу завдань звітності окремо. В роботі досліджується модель, яка ще більше роз’єднує архітектуру та переміщує завдання cron у кероване середовище без серверів, яке добре масштабується та зменшує витрати.

Метою кваліфікаційної роботи є дослідження безсерверної архітектури для створення щоденного звіту про продажі.

Для досягнення мети необхідно розв’язати три завдання.

Перше завдання – виконати аналіз сучасної архітектури з тісним зв'язком, який має низку недоліків. Дослідити нову версію програми, яка використовує принципи відокремленої архітектури для покращення доступності, масштабованості та продуктивності при обробці великого обсягу даних.

Друге завдання – запропонувати і дослідити модель безсерверної архітектури.

Нарешті третє завдання – отримати результати для запропонованої моделі безсерверної архітектури, що підтверджують автоматизацію звітів про продажі для кафе, що й надалі допомагатиме власникам бізнесу аналізувати щоденні продажі та планувати інвентаризацію кафе.

В першому розділі цієї роботи під час першого етапу було створено базовий додаток на основі NodeJS через IDE AWS Cloud9, що імітує тісно пов’язану архітектуру, яка демонструє недоліки традиційного підходу. У цій архітектурі веб-сервер і сервер додатків були інтегровані безпосередньо, що призводило до високої залежності між компонентами та обмежень у масштабуванні. Стоворювалась ця архітектура за допомогою заздалегідь підготованого коду за для швидшего розгортання, після чого налаштовувались IAM role для EC2 екземпляра на якому знаходився Cloud9. Потім налаштовувався доступ до S3 bucket на якому зберігатимуться завантажені зображення від користувачів. Після чого запускався застосунок через AWS Cloud9.

На другому етапі було реалізовано нову версію програми, яка використовує принципи відокремленої архітектури для покращення доступності, масштабованості та продуктивності при обробці великого обсягу даних [1]. Це досягнуто за рахунок застосування хмарних сервісів AWS, таких як Amazon SQS і Amazon SNS, що дозволяють мінімізувати затримки та зменшити залежність між компонентами.

Другий розділ пояснювальної записки присвячений розв’язанню другої проблеми. Бізнес-запит для кафе це – підготовки різних звітів завдяки AWS Lambda та завдань cron.

Бізнес-причина оновлення, якому присвячений третій розділ – підвищення продуктивності і зменшення витрат, одночасно підтримуючи вимоги до звітності. Технічні вимоги оновлення архітектури – розгорнути функції Lambda, які підключаються до бази даних Amazon RDS і створюють звіт на основі розкладу.

1 АНАЛІЗ СУЧАСНОЇ АРХІТЕКТУРИ

Сучасна архітектура включає відокремлені програми, які є меншими незалежними будівельними блоками, які зручно розробляти, розгортати та підтримувати. Черги повідомлень забезпечують зв'язок і координацію для цих розподілених програм. Черги повідомлень разом із системами сповіщень можуть значно скоротити кодування відокремлених додатків, одночасно покращуючи продуктивність, надійність і масштабованість[2][3].

У цьому розділі описаний аналіз роботи з програмою обробки зображень на примірнику AWS Cloud9 та використання Amazon Simple Queue Service (Amazon SQS) і Amazon Simple Notification Service (Amazon SNS), щоб створити роз’єднану архітектуру.

1.1 Сценарій

Досліджена програма, яка приймає зображення та обробляє їх для створення тонованих зображень. Дослідження має такі етапи:

Етап 1. На першому етапі описано процес створення додатка на основі NodeJS через IDE від AWS, Cloud9, де компоненти веб додатку, а саме веб-сервер, і сервер додатку, були тісно пов’язані між собою. За для прикладу погано зробленої архітектури через тісний зв’язок(рис 1.1).

Цикл цього застосунку проходив так:

1) Відбувається запуск застосунку
2) Користувач завантажує зображення через веб-додаток.
3) Веб-сервер зберігає зображення в сегменті S3 і оновлює інформацію про файл зображення (метадані) в Amazon DynamoDB разом із початковим статусом.
4) Веб-сервер синхронно ініціює обробку на сервері додатків і очікує,

поки оброблене зображення стане доступним (тісний зв’язок). Поки веб-сервер очікує, сервер додатків виконує обробку, оновлює статус таблиці DynamoDB і надсилає оброблене зображення до сегмента S3.

5) Користувачеві доступне нове тоноване зображення.

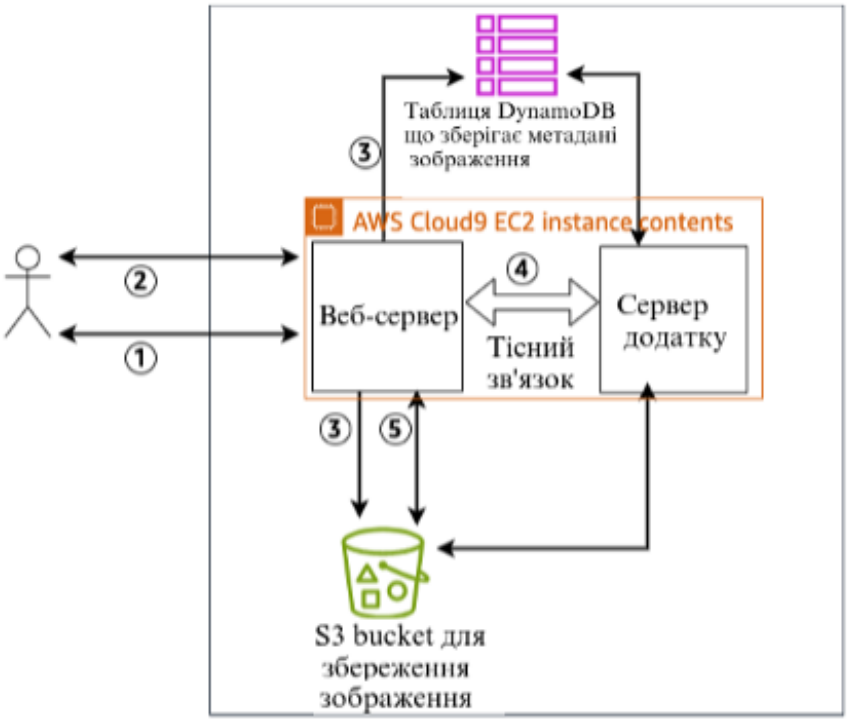

Рисунок 1.1 – Тісно пов’язана архітектура веб застосунку

Етап 2. На цьому етапі досліджується нова версія програми на основі відокремленої архітектури, яка покращує доступність, масштабування та продуктивність програми у сценарії великого обсягу [3]. Досягається це використовуючи комбінацію служб AWS, таких як Amazon SQS і Amazon SNS (рис. 1.2).

Цикл цього застосунку проходив так:

6) Відбувається запуск застосунку, який перевіряє зображення (пошук зображень).

7) Користувач завантажує зображення через веб-додаток.

8) Веб-програма записує зображення в S3 bucket, а також надсилає запит на оновлення інформації (метаданих) файлу зображення в DynamoDB разом із початковим статусом.

9) Щойно файл буде збережено в S3 bucket, у SNS надсилається сповіщення.

10) SNS має двох підписників. SNS публікує повідомлення в Amazon SQS і надсилає сповіщення електронною поштою про те що робить користувач.

11) Користувач запускає механізм обробки для сервера додатків. Цей механізм дозволяє серверу додатка опитувати повідомлення в черзі для обробки.

12) Сервер додатків обробляє повідомлення, шукаючи інформацію про відповідне зображення в таблиці DynamoDB, отримує зображення з Amazon S3 і згодом починає обробку зображення для тонування та зміни розміру. Статус зображення оновлюється в таблиці DynamoDB до завершення процесу.

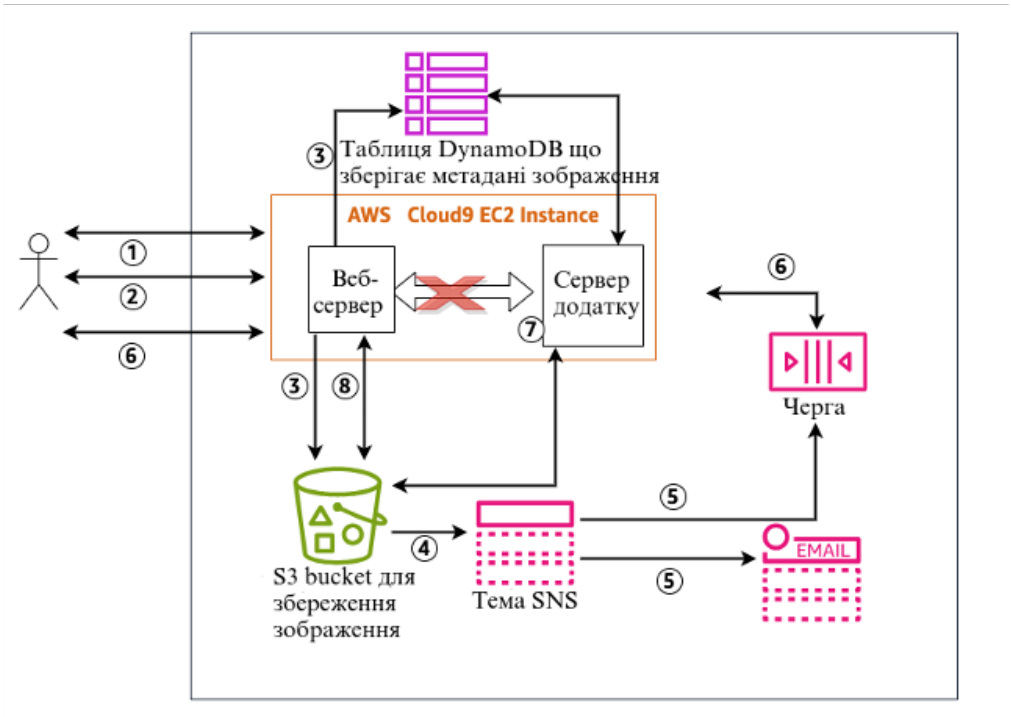

Рисунок 1.2 – Роз’єднана архітектура веб-застосунку

Дослідження розпочиналось з архітектури, подібної до наступної (рис.1.3):

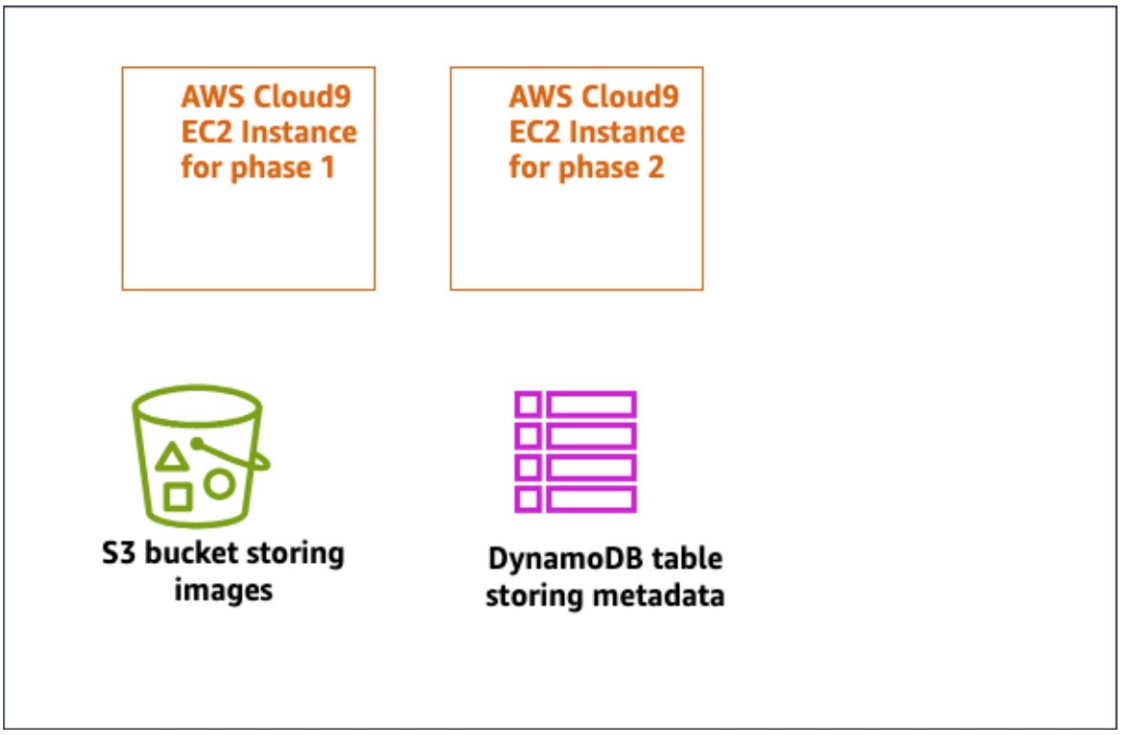

Рисунок 1.3 – Базова архітектура

1.2 Фаза 1: Створення тісно пов’язаної програми

Крок 1. Вхід в AWS Cloud9 IDE

Для початку в AWS Cloud9(посилання було в Guided lab><«AWS Details») в налаштуваннях відключено функцію тимчасових credentials, бо вже була роль по IAM яка призначена для цієї практичної (рис.1.4).

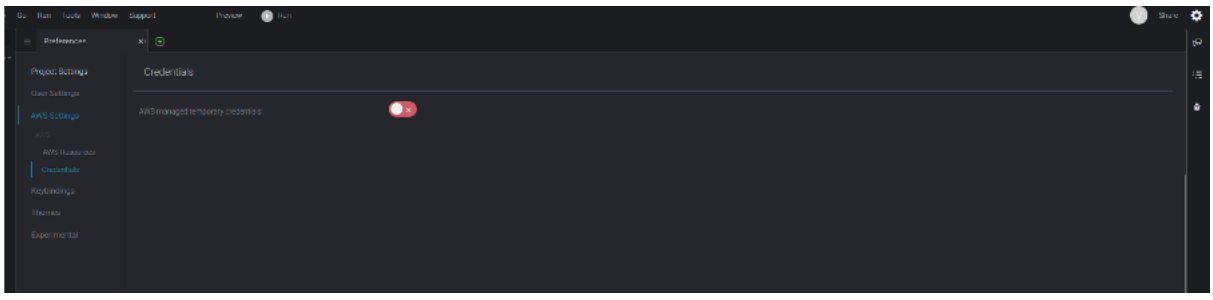

Рисунок 1.4 – Налаштування Cloud9

Крок 2. Завантаження файлів і встановлення програми

Далі встановлено необхідні файли в додатковому терміналі, встановлено їх наступними командами (додаток Б):

wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACACAD-3-113230/17-lab-mod13-guided-SQS/code.zip

unzip code.zip cd phase_1/web_server_1/ npm install

Потім в ще одному терміналі у відповідній теці встановлено «npm» (рис. 1.5).

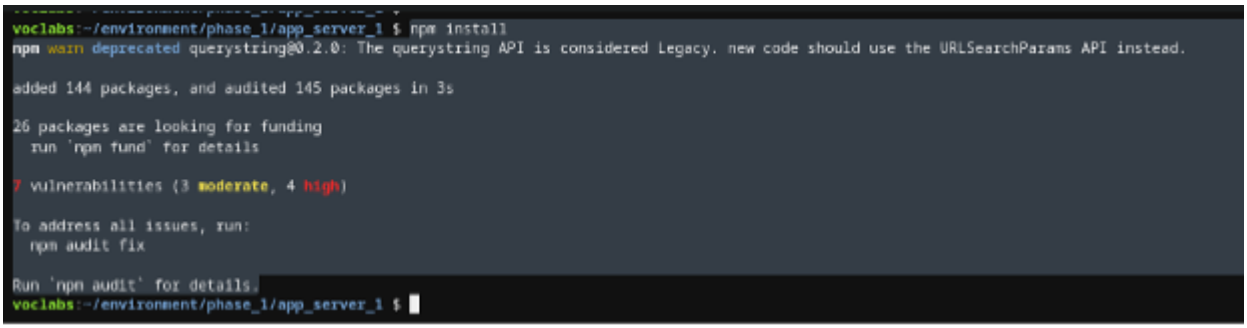

Рисунок 1.5 – Встановлення Npm

Примітка: Як видно зі знімку екрана в процесі встановлення виникла помилка, яка вирішується командами: npm audit fix та npm fund.

Крок 3. Налаштування групи безпеки та ролі IAM

Потім не закриваючи Cloud9 було відкрито EC2 instance, та налаштовано security group для phase1 (рис 1.6).

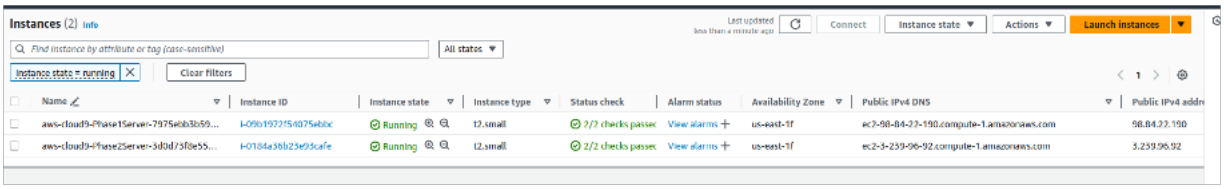

Рисунок 1.6 – AWS EC2 instances з Phase1 та Phase2

Було обрано інстанс в якому написано «Phase1», у вкладці «Security», і обрано «security groups» та «Edit inbound rules».

Додано правило за типом «Custom TCP», діапазон портів «8000-8100»(на цих портах працює веб-застосунок), і джерело (правило про те хто отримує доступ за цим правилом) обирано «Anywhere-Ipv4» (рис. 1.7).

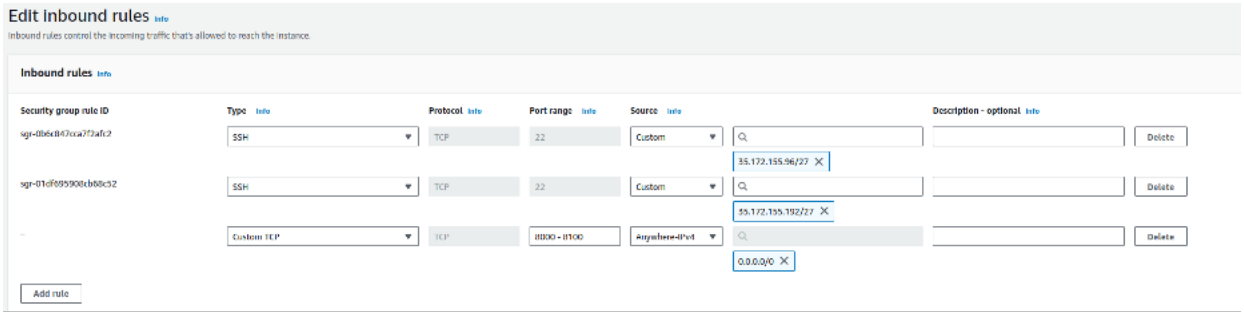

Рисунок 1.7 – Налаштування безпекових політик для Phase 1

Потім після повернення до EC2 instance і вибрано Actions > Security > Modify IAM role для Phase1 і в IAM Role обрано Ec2InstanceProfile (рис. 1.8) потім – теж саме для Phase2 instance.

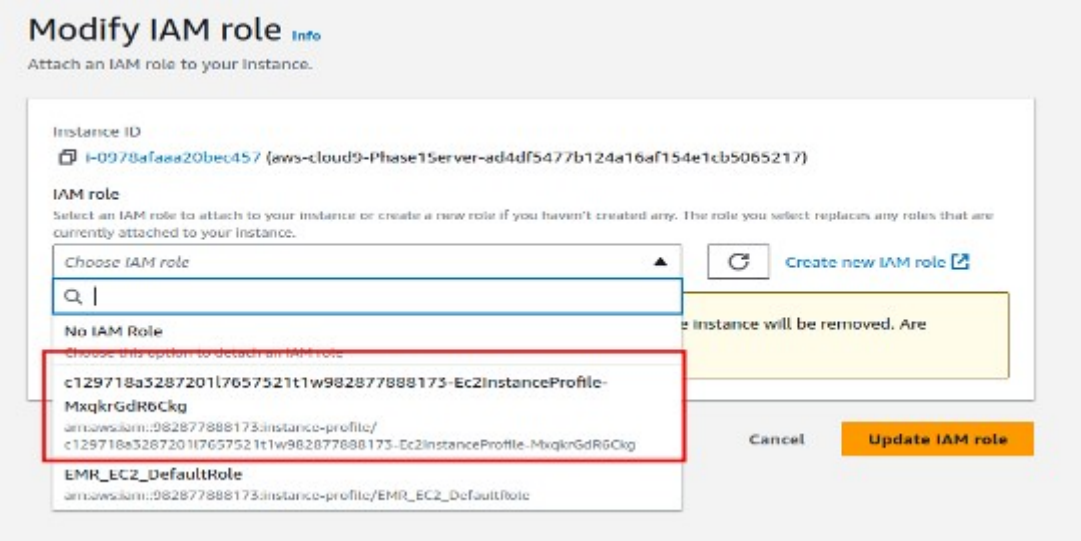

Рисунок 1.8 – Налаштування IAM Role

Крок 4. Налаштування дозволів сегмента S3

Після завершення налаштувань EC2 налаштоано S3 bucket, в S3 обрано сегмент, де написано Phase1, і в розділі permission, прибрана заборона на Block public access (bucket settings)(бо інакше не буде доступу ззовні до S3 bucket).

Потім в налаштуваннях Bucket policy налаштовуваний публічний доступ до всіх об'єктів у S3 сегменті, але тільки для своєї IP-адреси (рис. 1.9).

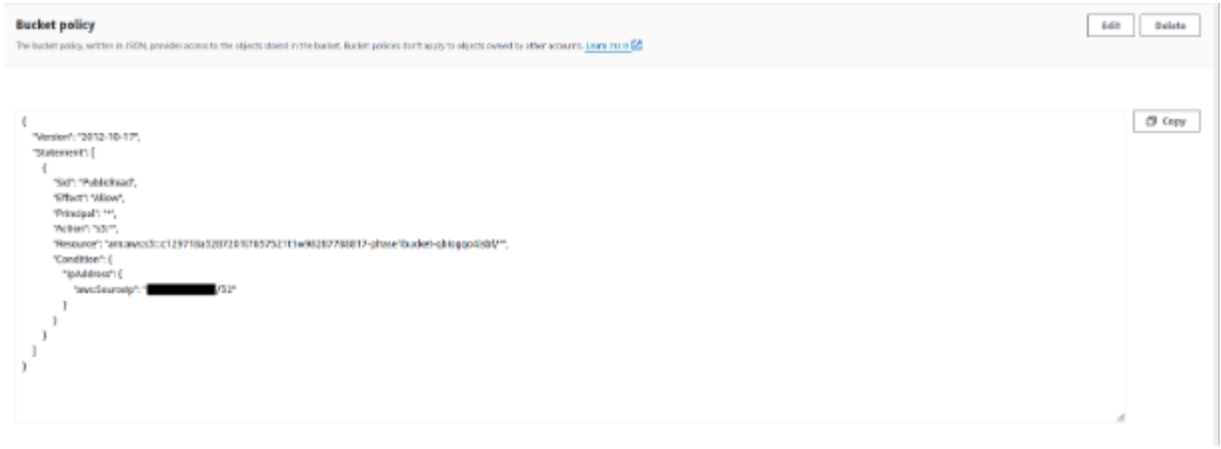

Рисунок 1.9 – Налаштування політик S3 bucket

Крок 5. Налаштування та запуск програми

Після було виконано перехід до Cloud9, та налаштовано web_server_1/static/js/config.js файл. В полях CONFIG.FULL* прописані IP адреса EC2 instance і посилання на S3 bucket (рис. 1.10).

```
Do not replace any other values except those mentioned in the lab guide you are following
*/

//GLOBAL
CONFIG = {};

CONFIG.STATUS_STR_ARR = [
    "Got original image",
    "Saved original image to s3",
    "App server received image url",
    "App server got original image buffer",
    "Processed image",
    "Saved adjusted to s3",
    "Complete"
];

CONFIG.FULL_URL_STR = "http://18.234.117.11:8008/upload";
CONFIG.FULL_IMAGE_PATH_STR = "https://arn:aws:s3:::c129718a3287201l7657521t1w98287788817-phase1bucket-qbiogqo4lsbf.s3.amazonaws.com";
CONFIG.FULL_IMAGE_URL_STR = "http://18.234.117.11:8008/get-image-urls";
```

Рисунок 1.10 – Загальне налаштування веб сервера

Далі налаштовано web_server_1/libs/config.js файл. Де прописано на якому порту працює веб сервер застосунку (рис. 1.11).

```
CONFIG = {};

CONFIG.SECRET_STR = "secret_squirrel";

CONFIG.DYNAMO = {};
CONFIG.DYNAMO.API_VERSION_STR = "2012-08-10";
CONFIG.DYNAMO.REGION_STR = "us-east-1";

CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase1";

CONFIG.PROCESSING_URL_STR = "http://18.234.117.11:8009/process";

CONFIG.S3 = {};
CONFIG.S3.API_VERSION_STR = "2006-03-01";
CONFIG.S3.REGION_STR = "us-east-1";
CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a3287201l7657521t1w98287788817-phase1bucket-qbiogqo4lsbf";
```

Рисунок 1.11 – Налаштування веб сервера

І також налаштовано app_server_1/libs/config.js, де прописана адреса S3 bucket до якого застосунок звертається, щоб отримати зображення для обробки (рис.1.12).

```
1  //GLOBAL
2  CONFIG = {};
3
4  CONFIG.SECRET_STR = "secret_squirrel";
5
6  CONFIG.DYNAMO = {};
7  CONFIG.DYNAMO.API_VERSION_STR = "2012-08-10";
8  CONFIG.DYNAMO.REGION_STR = "us-east-1";
9  CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase1";
10
11 CONFIG.S3 = {};
12 CONFIG.S3.API_VERSION_STR = "2006-03-01";
13 CONFIG.S3.REGION_STR = "us-east-1";
14 CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a3287201l7657521t1w98287788817-phase1bucket-qbiogqo4lsbf";
```

Рисунок 1.12 – Налаштування веб застосунку

Після всіх налаштувань виконано запуск веб-застосуноку, командою «npm start». Це було зроблено в двох терміналах, знаходячись в phase_1/web_server_1 і знаходячись в phase_1/app_server_1 після чого вони починали прослуховувати порти 8008, і 8009.

Після запуску було перевірено чи працює веб-застосунок, для цього скопійовано публічну IP адресу в EC2 instance «Phase1», і вставлено її в адресний

рядок в веб браузері, і додано номер порту «:8008» на якому знаходився веб-застосунок (рис. 1.13).

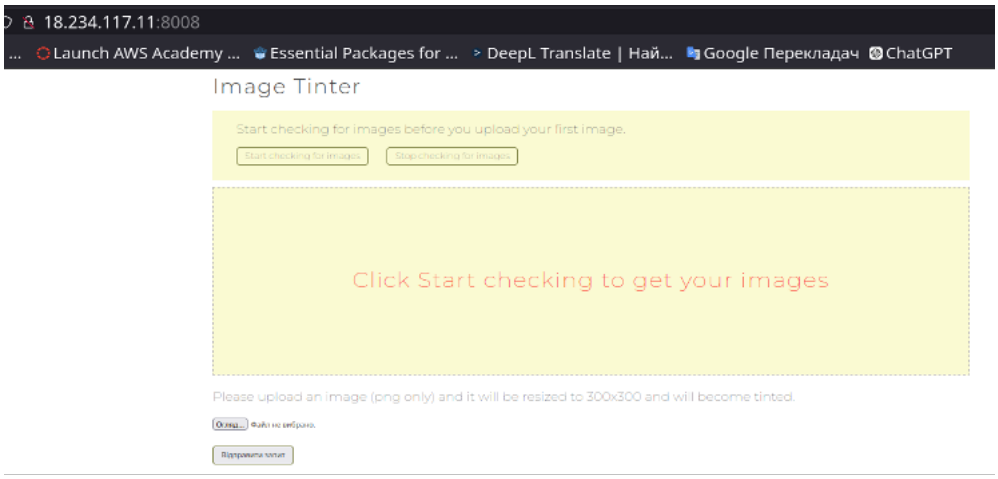

Рисунок 1.13 – Працюючий застосунок з тісним зв’язком

Для тестування програми, виконано команду «Start checking for images» та завантажено зображення. Під час обробки у фоновому режимі «Choose File», обирано файл, який підлягає завантаженню, і виконано його відправку «Submit». Оброблене зображення відображається, коли воно стає доступним.

Архітектура цього застосунку працювала наступним чином: коли користувач завантажував зображення через веб-додаток, воно спочатку зберігалося в AWS S3, а веб-сервер відразу оновлював метадані цього файлу (наприклад, статус завантаження або обробки) в таблиці Amazon DynamoDB. Після цього веб-сервер ініціював обробку зображення на сервері додатку, підтримуючи тісний синхронний зв'язок із сервером додатка. Весь процес був побудований на тісній взаємодії компонентів: сервер додатка забирав зображення із сегменту S3, виконуючи необхідну обробку, наприклад, застосування фільтрів або трансформацій. Після завершення обробки він оновлював статус в DynamoDB, щоб відобразити, що зображення готове, і знову зберігав результат до S3, звідки користувач завантажував вже змінене, оброблене зображення.

Однак цей процес мав істотні недоліки. Оскільки між веб-сервером і сервером додатків існував тісний синхронний зв'язок, кожен із них був залежним один від одного. Веб-сервер чекав завершення обробки, не продовжуючи виконувати інші задачі. У разі, якщо один із компонентів виходив з ладу — чи то веб-сервер, чи сервер додатка, — це призводило до збоїв у всій системі. Така архітектура була вразливою до несправності і потребувала високої надійності всіх частин. Крім того, синхронний характер обміну між компонентами обмежував масштабованість системи, оскільки веб-сервер не міг обробляти інші запити, доки тривав процес обробки зображення.

Одним із можливих рішень цієї проблеми є або впровадження асинхронної архітектури або використання архітектури роз'єднаних компонентів, що дає можливість серверу додатка і веб-серверу функціонувати незалежно один від одного, зменшуючи залежність системи від тісного зв'язку. Це дозволяє підвищити стійкість до збоїв, оскільки вихід з ладу одного з компонентів не впливав би на роботу інших частин програми.

1.3 Фаза 2: Створення програми з відокремленою архітектурою

Крок 1. Попереднє налаштування AWS Cloud9 IDE

Перед початком другого етапу, було налаштовано середовище розробки Cloud9, тому було повторено минулі кроки, як на початку фази 1, бо для цього Cloud9 також роль IAM передбачає необхідні credentials (рис.1.14).

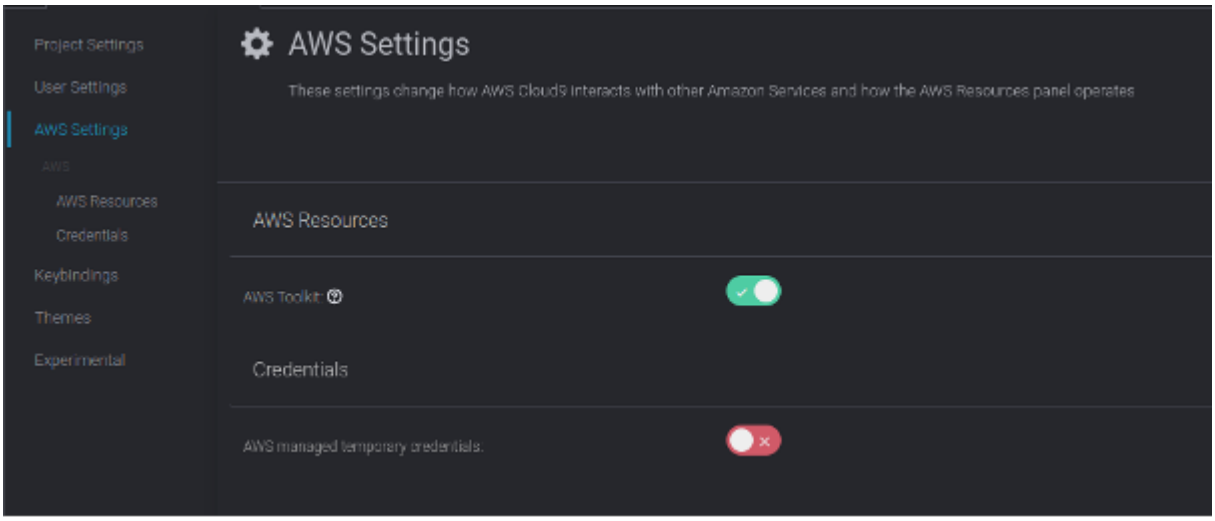

Рисунок 1.14 – Відключення управління тимчасовими обліковими данними.

Потім було встановлено необхідні файли, це виконано за допомогою команд (додаток Б):

wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CURTF-200ACACAD-3-113230/17-lab-mod13-guided-SQS/code.zip

Розпакування архіву; unzip code.zip

Потім виконано перехід до теки веб сервера де було встановлено «npm»; cd phase_2/web_server_2/ npm install

cd phase_2/app_server_2/ npm install

Крок 2. Налаштування Amazon SQS.

Потім налаштовано Simple Queue Service(SQS), це сервіс від AWS для передачі додатками повідомлень між різними компонентами або мікро сервісами. Була створена черга, з ім’ям “ImageApp”, усі інші налаштування залишено не торкнутим (рис. 1.15).

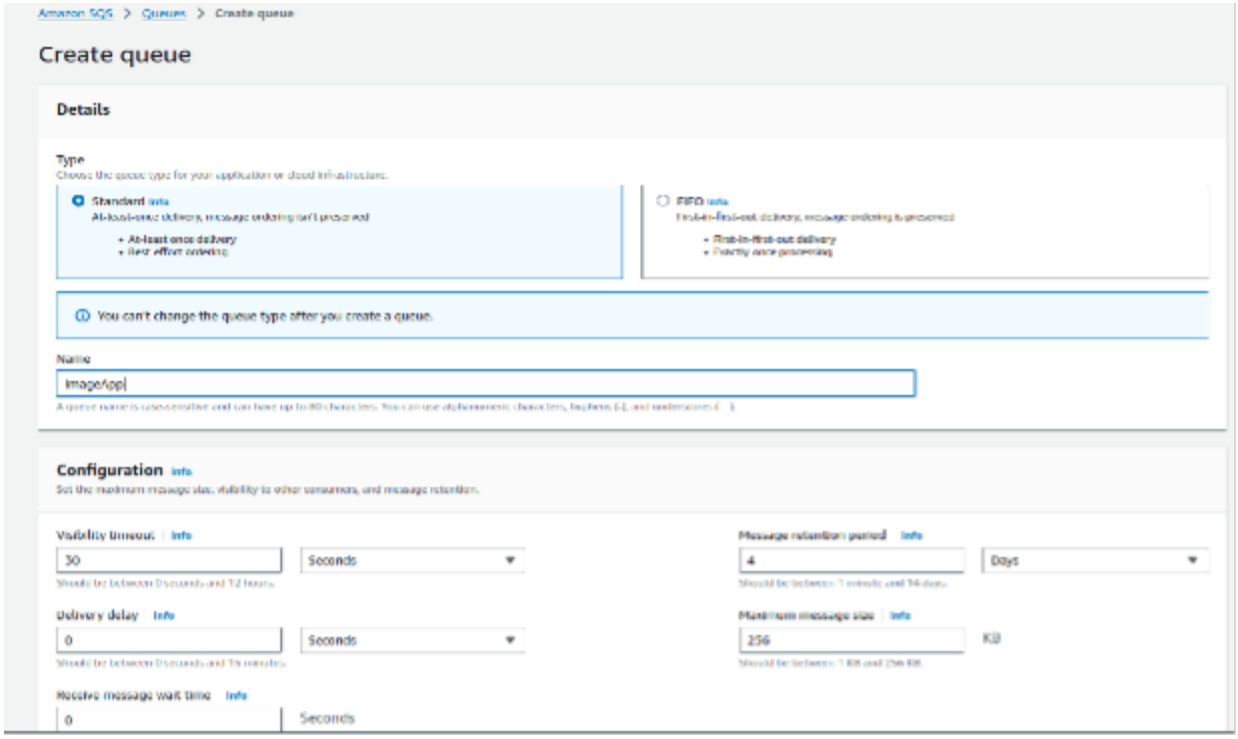

Рисунок 1.15– Створення нової черги в AWS SQS

Після створення черги скопійовано URL адресу створеної черги (рис. 1.16).

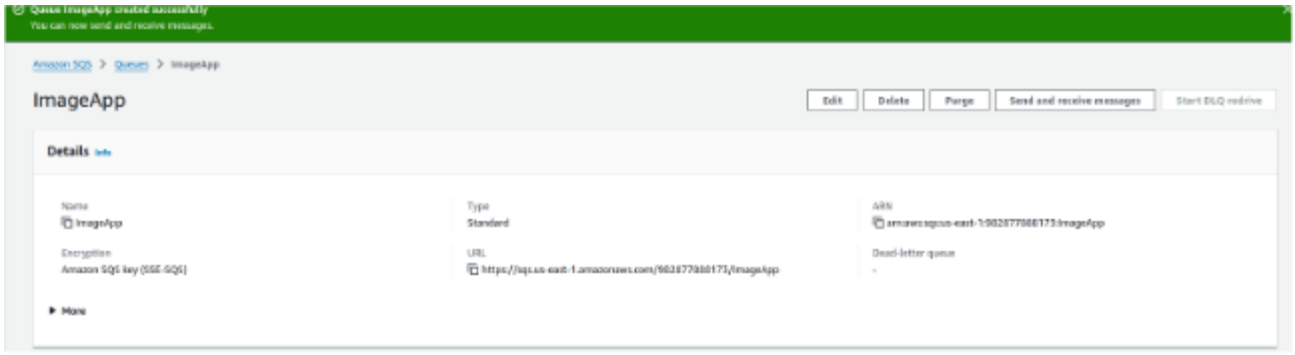

Рисунок 1.16 – Новостворена черга

Крок 3.Налаштування Amazon SNS.

Після налаштування SQS черги, було налаштовано Simple Notification Service(SNS), це керований сервіс від AWS для обміну повідомленнями (сповіщеннями) між додатками та сервісами. Він дозволяє розробникам відправляти повідомлення до різних кінцевих точок або підписників через моделі

публікації-підписки. В цій роботі було налаштовано цей сервіс, він служить для відправлення повідомлень на сервер застосунку та на особисту пошту вказану в розсилці повідомлень. Він відправляє повідомлення після якихось змін на сервері додатку.

Спочатку був здійснений перехід до сервісу SNS для налаштування (рис.1.17).

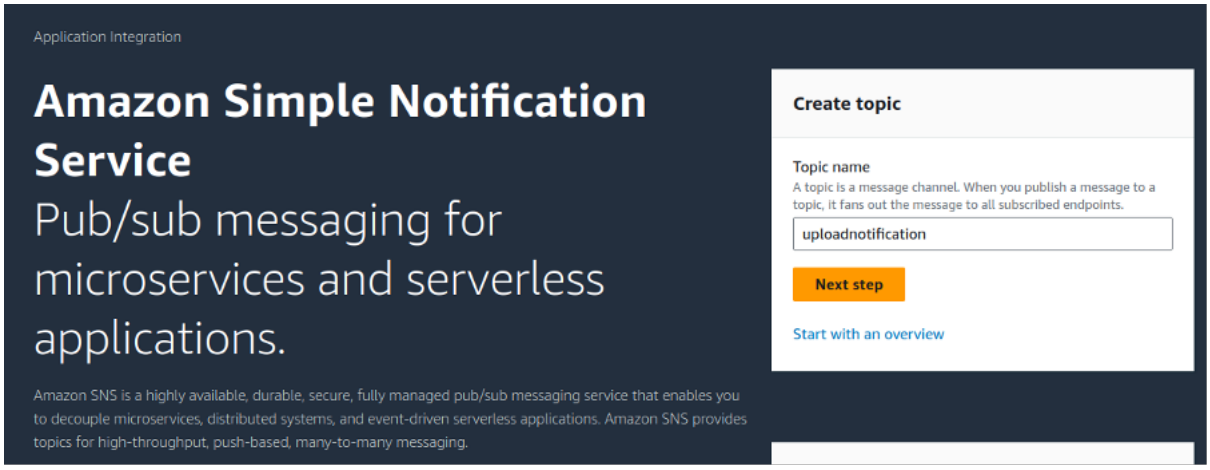

Рисунок 1.17 – Начальне вікно для створення нового «topic»

Після створення теми «Topic», під назвою «uploadnotification», було налаштовано його політику «Access policy», для цього обрано спосіб налаштування «advanced», і заповнений JSON файл (рис. 1.18) .

```
{
    "Version": "2012-10-17",
    "Id": "S3UploadNotification",
    "Statement": [
        {
            "Sid": "S3 SNS topic policy",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": [
                "SNS:Publish"
            ],
            "Resource": "arn:aws:sns:us-east-1:982877888173:uploadnotification",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:*:*:c129718a3287201l7657521t1w98287788817-phase2bucket-pqmyuxkkwxkd"
                },
                "StringEquals": {
                    "aws:SourceAccount": "982877888173"
                }
            }
        }
    ]
}
```

Рисунок 1.18 – Налаштування політики доступу в SNS

Крок 4. Налаштування дозволів Amazon S3 і сповіщень про події.

Після налаштування політики доступу було налаштовано S3 bucket куди потрапляють та зберігаються зображення, для налаштування S3 bucket для другої фази, було використано ті самі налаштування, що і для сегмента bucket в першій фазі.

Перед налаштуванням bucket policy, обрано сегмент де написано Phase2 в меню вибору сегментів в S3, і в розділі «permission» прибрано заборону на Block public access (bucket settings).

Після цього налаштовано в JSON файлі наступні політики (рис 1.19) (додаток Б).

Рисунок 1.19 – Налаштування політик доступу S3 bucket

В розділі «Properties», в підрозділі «Event notifications» було створено нове повідомлення про подію:

Назва події: «SendtoSns»;

Object creation: «Select All object create events» (рис. 1.20);

Destination: «SNS topic»; SNS Topic : «uploadnotification» (рис. 1.21).

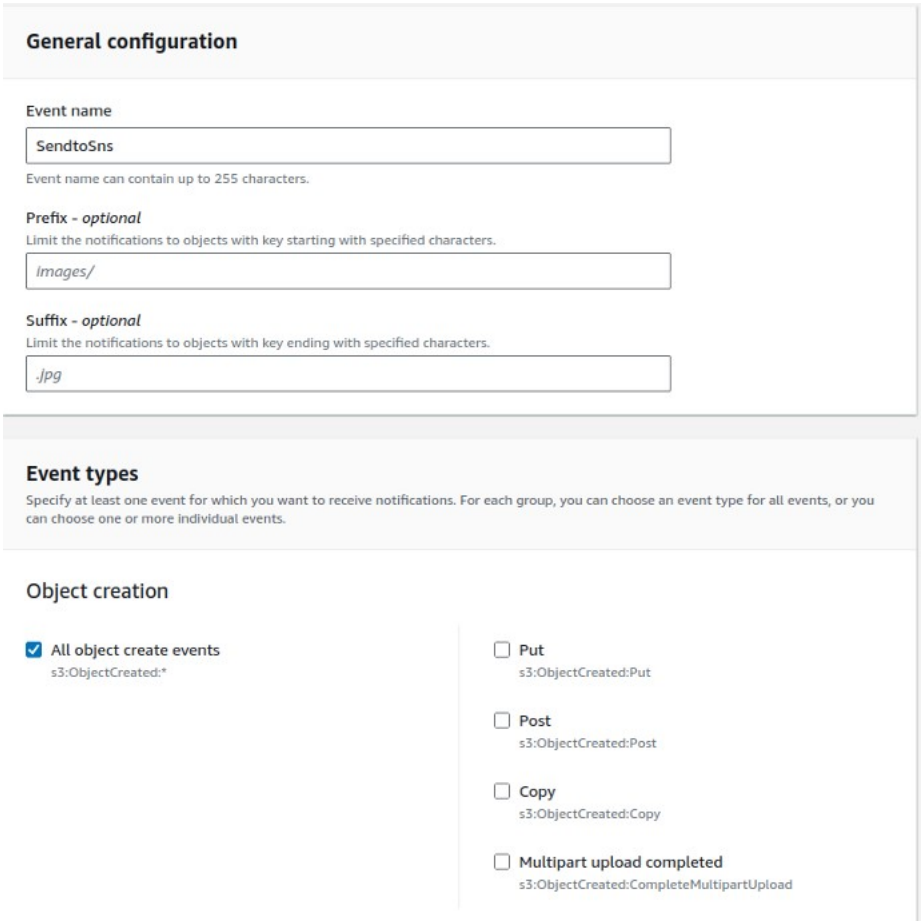

Рисунок 1.20 – Створення повідомлення про події

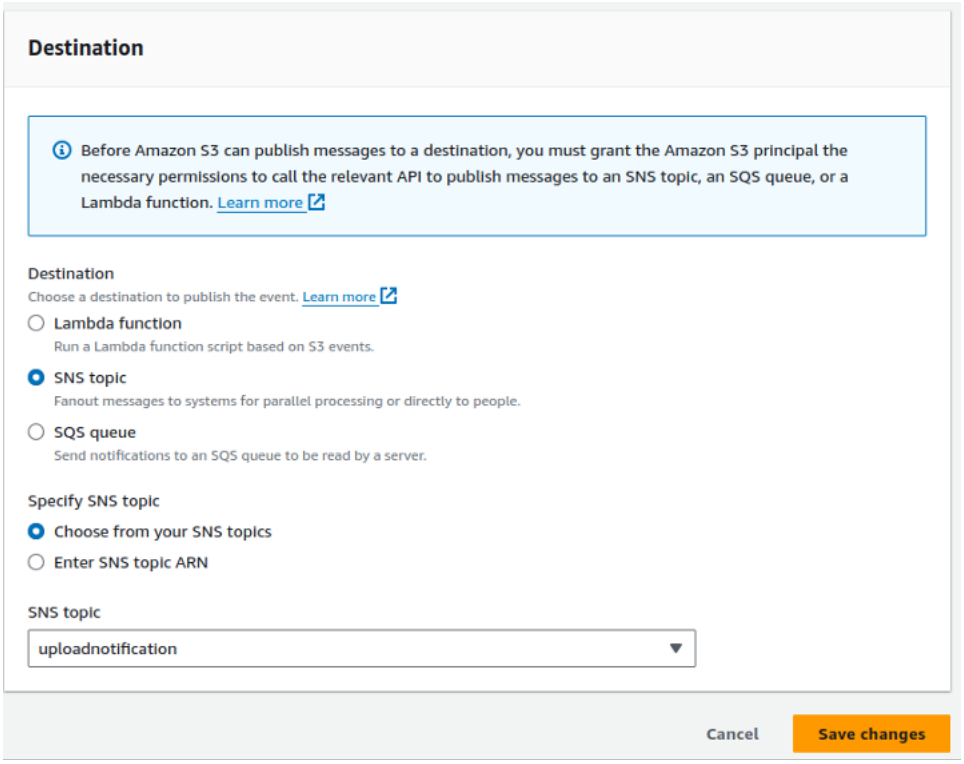

Рисунок 1.21 – Встановлення пункту призначення повідомлення

Крок 5. Створення підписок на Amazon SNS.

Після створення сповіщення про подію в Amazon S3 для надсилання сповіщення в Amazon SNS, була створена підписка на тему SNS для надсилання повідомлення в чергу. Також налаштована підписка на електронну пошту.

В розділі SQS було обирано створену раніше «ImageApp». Після чого в Actions обрано "Subscribe to Amazon SNS topic" на сторінці теми у списку обрано тему «uploadnotification» (рис.1.22).

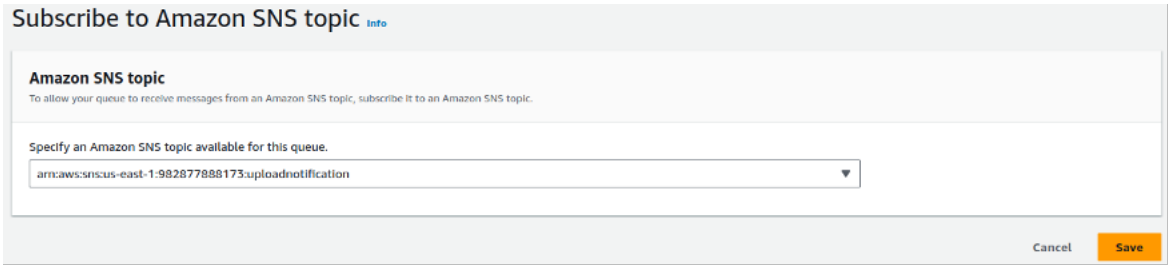

Рисунок 1.22 – Налаштування підписки на SNS

Далі налаштовано «SNS email subscription» в "uploadnotification", для цього в «Subscriptions» виконано команду «Create subscription» (рис. 1.23).

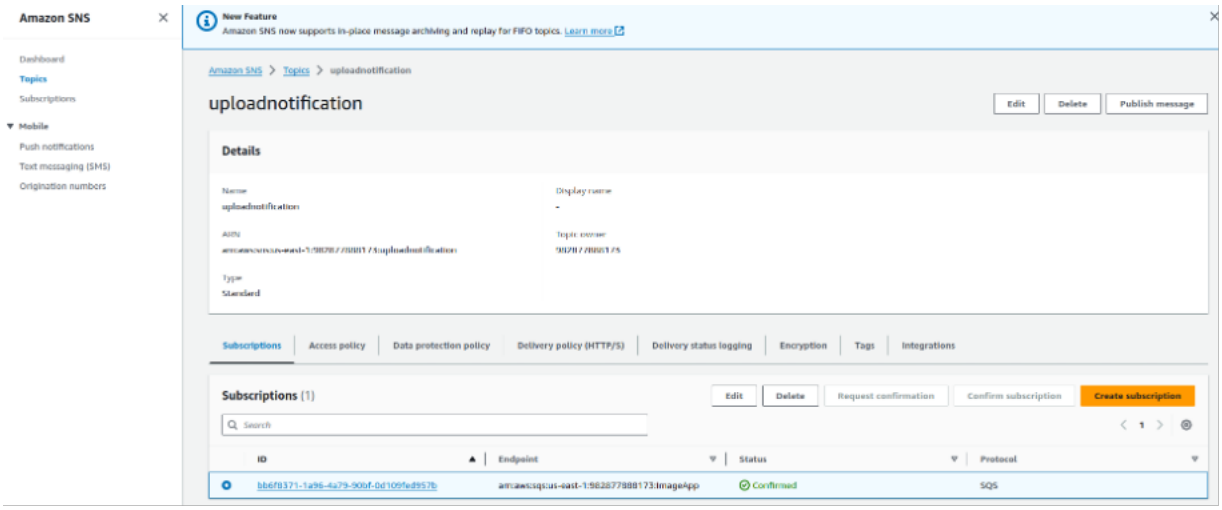

Рисунок 1.23 – Опис SNS topic "uploadnotification"

В новій підписці вказано пошту для отримання сповіщень (рис. 1.24) та отримання листа для підтвердження підписки на SQS пошти (рис. 1.25).

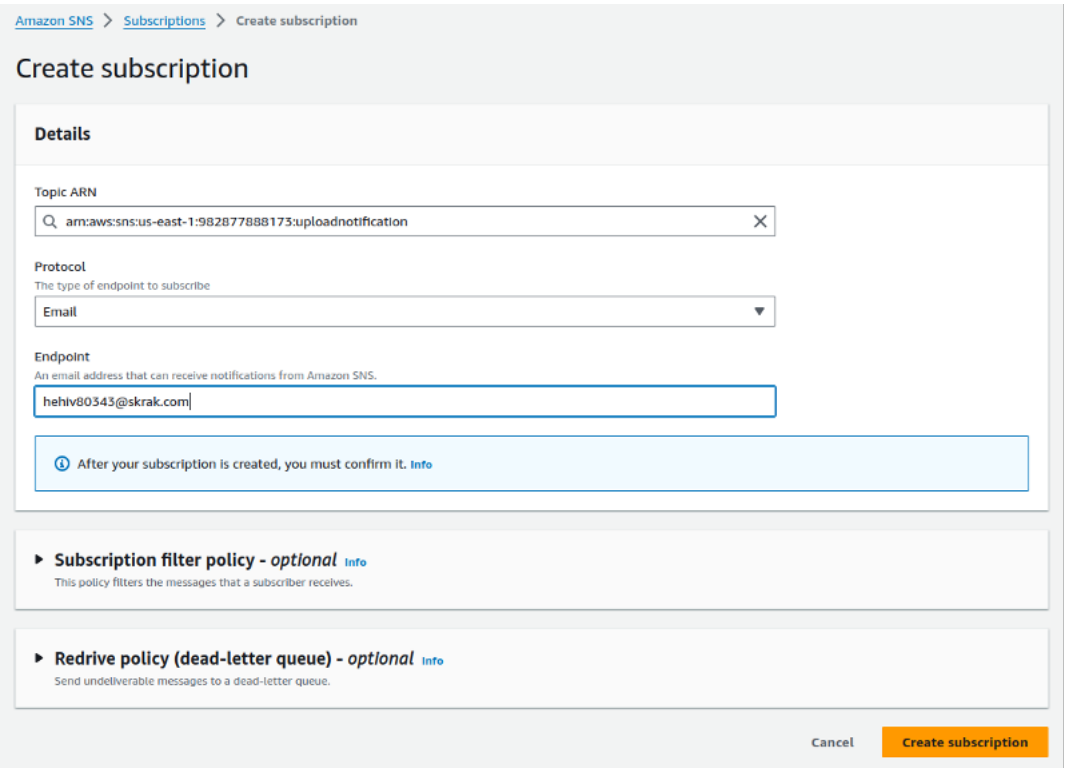

Рисунок 1.24 – Додавання пошти для повідомлень від SQS

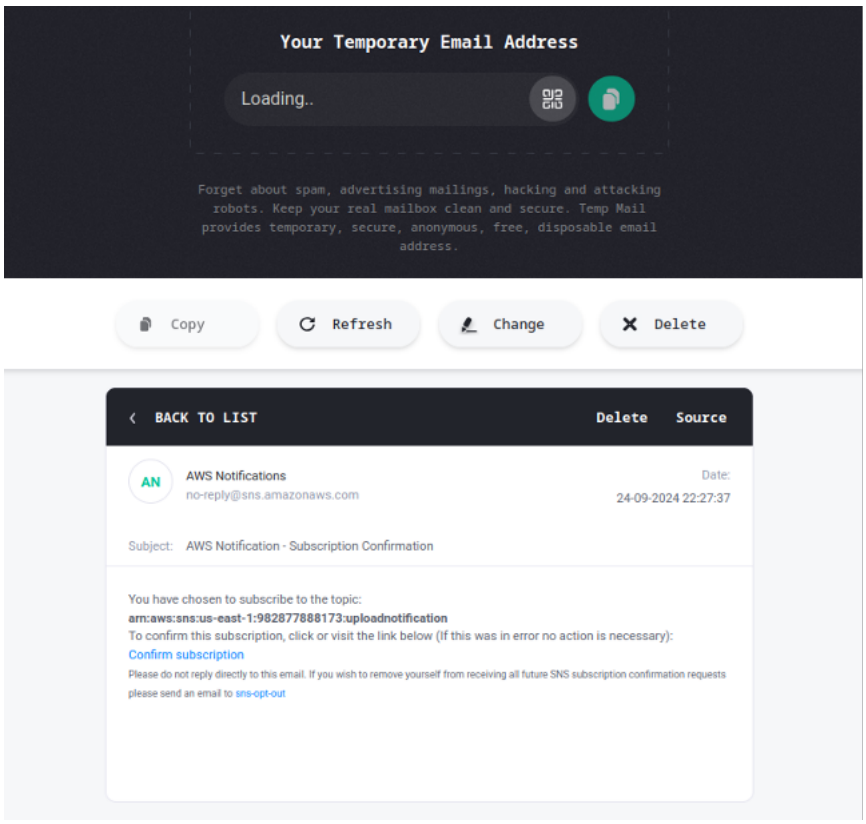

Рисунок 1.25 – Підтвердження на пошті підписки на SNS

Крок 6. Налаштування параметрів і запуск програми.

Після всіх налаштувань в сервісах AWS, в Cloud9 IDE налаштовано параметри в файлі для веб застосунку та веб сервера. На відміну від минулої фази, в цій більше загальних налаштувань, які були прописані, бо застосунок складався з більшої кількості частин, прописувались вони в файлі «web_server_2/static/js/config.js» (рис. 1.26).

```
CONFIG = {};

//PHASE 2 CONFIG

CONFIG.FULL_IMAGE_URL_STR = "http://3.81.22.152:8007/get-image-urls";
CONFIG.FULL_IMAGE_PATH_STR = "https://c129718a3287201l7657521t1w98287788817-phase2bucket-pqmyuxkkwxkd.s3.amazonaws.com";
CONFIG.ARN_TRUNC_URL_STR = "c129718a3287201l7657521t1w98287788817-phase2bucket-pqmyuxkkwxkd";
CONFIG.WEB_SERVER_URL_STR = "http://3.81.22.152:8007";
CONFIG.APP_SERVER_URL_STR = "http://3.81.22.152:8010";

CONFIG.STATUS_STR_ARR = [
    "Sent to S3",
    "App server received image url",
    "App server got original image buffer",
    "Processed image",
    "Saved adjusted to s3",
    "Complete"
];
```

Рисунок 1.26– Загальне налаштування веб сервера

Потім були налаштовані параметри в файлі «web_server_2/libs/config.js» де були прописані назва S3 bucket до якого звертається веб сервер (рис. 1.27).

```
bash - "ip-10-16-0-178"  ×  bash - "ip-10-16-0-178"  ×  JS config.js  ×  JS config.js

//GLOBAL
CONFIG = {};

CONFIG.SECRET_STR = "secret_squirrel";

CONFIG.DYNAMO = {};
CONFIG.DYNAMO.API_VERSION_STR = "2012-08-10";
CONFIG.DYNAMO.REGION_STR = "us-east-1";

//we use the same DDB table for phase 2
CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase2";

CONFIG.S3 = {};
CONFIG.S3.API_VERSION_STR = "2006-03-01";
CONFIG.S3.REGION_STR = "us-east-1";
CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a3287201l7657521t1w98287788817-phase2bucket-pqmyuxkkwxkd";

// (Ignore the commented code below)
//CONFIG.DYNAMO.PROFILE_STR = "default";
// CONFIG.S3.PROFILE_STR = "default";
```

Рисунок 1.27 – Налаштування веб сервера

І останнім було налаштовано параметри в файлі "app_server_2/libs/config.js" додававши назву bucket до якого виконувалось звернення та URL адресу AWS SQS, яка була і скопіювана раніше (рис. 1.28).

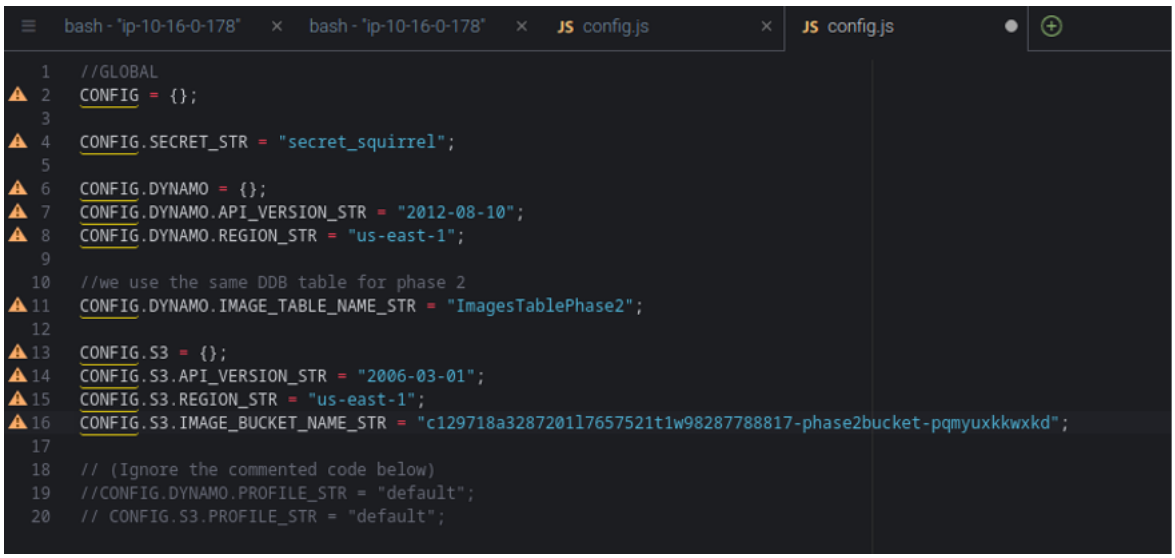

Рисунок 1.28 – Налаштування веб застосунку

Для запуску веб сервера, і сервера застосунку, була використана команда npm install, після переходу в теки в яких знаходяться веб-сервер «phase_2/web_server_2/» та сервер застосунку «phase_2/app_server_2/», і виконання команди cd з прописаним повним шляхом.

Крок 7.Тестування програми

Для перевірки програми, була скопіювана IP адреса, EC2 instance «Phase2» та додано до неї «:8007» і вставлена в пошукову строку. Таким чином виконано перехід на головну сторінку застосунку, перевірено його завантаживши декілька зображень для того щоб додаток відредагував їх(рис. 1.29).

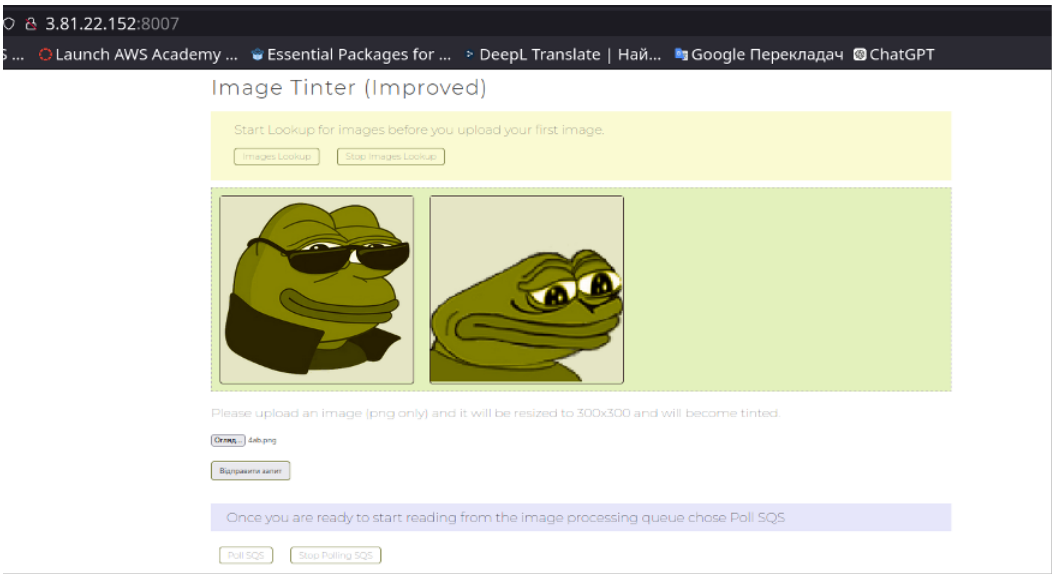

Рисунок 1.29 – Виведення застосунку при перевірці роботи

А також було перевірено пошту куди надійшов лист про використання застосунку, і з якої IP адреси це було зроблено(рис.1.30).

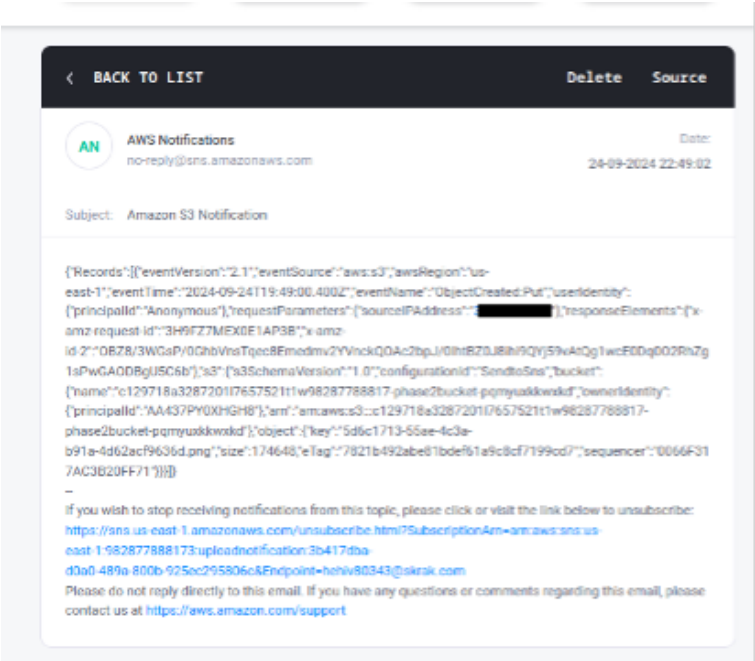

Рисунок 1.30 – Сповіщення на пошту про дії користувача у застосунку

1.4 Висновки до першого розділу

Архітектура з тісним зв'язком має низку недоліків. Синхронна взаємодія між компонентами призводить до того, що один процес очікує завершення іншого, що знижує загальну ефективність [4]. Така залежність компонентів створює вразливість, оскільки збій у роботі одного з них може призвести до зупинки всієї системи. Масштабування таких рішень є складним, оскільки зростання кількості запитів потребує збільшення ресурсів для всіх пов'язаних компонентів одночасно, що може бути неефективним з точки зору витрат.

Розділена архітектура, розглянута в розділі, має кілька суттєвих переваг. Вона дозволяє незалежним компонентам веб-застосунку функціонувати окремо, що забезпечує гнучкість та масштабованість системи [5]. У такій архітектурі кожен елемент може працювати незалежно, що значно підвищує стійкість до збоїв: навіть якщо один з компонентів виходить з ладу, це не призводить до загальної втрати

працездатності. Окрім цього, розділення функцій покращує продуктивність, адже компоненти не блокують один одного під час виконання складних операцій. Застосування AWS-сервісів, таких як SQS та SNS, забезпечує асинхронну обробку запитів, що дозволяє краще обслуговувати великі обсяги даних та обробляти їх швидше й ефективніше [6].

Однак власники хочуть отримувати щоденні звіти електронною поштою про всі замовлення, розміщені на веб-сайті. Один власник хоче передбачити попит, щоб у майбутньому він міг спекти правильну кількість десертів (зменшуючи відходи). Інший власник хоче виявити будь-які закономірності в бізнесі кафе (аналітика). В роз’єднаній архітектурі налаштовано завдання cron на екземплярі вебсервера, який надсилає ці щоденні повідомлення електронної пошти зі звітами про замовлення власникам. Однак завдання cron вимагає ресурсів і знижує продуктивність веб-сервера.

2 МОДЕЛЬ БЕЗСЕРВЕРНОЇ АРХІТЕКТУРИ

Традиційно програми працюють на серверах. Ці сервери можуть бути фізичними або віртуальними середовищами, які працюють поверх фізичних серверів. Однак ці сервери потрібно придбати та підготувати всі ці типи серверів, а також керувати їх потужністю [7]. На відміну від цього, можна запускати свій код на AWS Lambda без необхідності попереднього розподілу серверів [8]. Для Lambda потрібно надати лише код і визначити тригер. Функція Lambda може запускатися, коли це необхідно, чи то раз на тиждень, чи то сотні разів щосекунди. Плата береться лише за те, що використовується(за кожну відпрацьовану хвилину функції lambda).

2.1 Опис моделі

При розширені бізнеса, виникає і потреба в більш централізованому, а в ідеалі автоматизованому контролі, за необхідною сировиною, чи іншими елементами, необхідними для бізнесу.

В рамках цієї моделі викликається функція Lambda, коли файл завантажується до Amazon Simple Storage Service (Amazon S3). Файл завантажується в таблицю Amazon DynamoDB [9]. Дані доступні для перегляду на сторінці інформаційної панелі, яка отримує дані безпосередньо з DynamoDB. Це рішення не використовує Amazon Elastic Compute Cloud (Amazon EC2). Це безсерверне рішення, яке автоматично масштабується під час використання. Це рішення також вимагає невеликих витрат, коли воно використовується. Коли воно неактивно, витрати практично відсутні, оскільки рахунок виставляється лише за зберігання даних.

Завдяки цьому створюється система відстеження запасів. Крамниці з усього світу завантажують файл інвентаризації на Amazon S3 [10]. Щоб власники могли

мати можливість переглядати рівні запасів і надсилати сповіщення, коли рівень запасів низький.

2.2 Лямбда-функції для завантаження даних

Перша лямбда-функція створена для обробки файлу інвентаризації. Функція читає файл і вставляє інформацію в таблицю DynamoDB [11]. Blueprints — це шаблони коду для написання лямбда-функцій. Надаються креслення для стандартних лямбда-тригерів, таких як створення навичок Amazon Alexa та обробка потоків Amazon Kinesis Data Firehose. Застосовується попередньо написана функцію Lambda, тому використовувалась опція Author from scratch .

Створена нова лямбда функцію з наступними параметрами(рис. 2.1), компілятор коду Python 3.9, назва “Load-Inventory” і роль " Lambda-Load-Inventory-Role.”

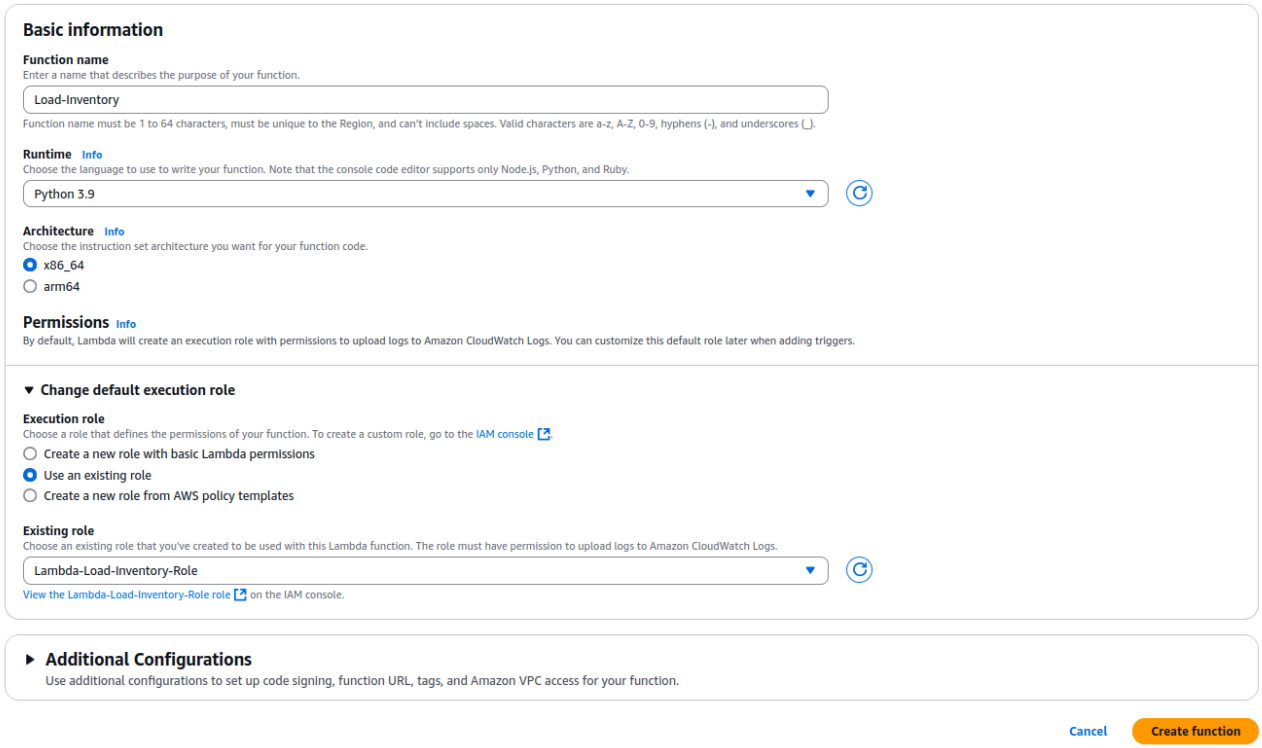

Рисунок 2.1 - Створення lambda функції

Ця роль надає дозволи функції Lambda, щоб вона могла отримати доступ до Amazon S3 і DynamoDB.

Після створення lambda функції прописано код функціоналу (Додаток Б) та збережено оновлений код. Код виконує такі дії: завантажує файл із Amazon S3, який ініціював подію; переглядає кожен рядок у файлі; вставляє дані в таблицю Inventory інвентаризації DynamoDB.

В наступному підрозділі описано налаштування Amazon S3 для запуску функції Lambda під час завантаження файлу.

2.3 Налаштування події Amazon S3

Крамниці з усього світу надають файли інвентаризації для завантаження в систему відстеження інвентаризації. Замість того, щоб завантажувати свої файли через FTP, крамниці можуть завантажувати їх безпосередньо в Amazon S3. Вони можуть завантажувати файли через веб-сторінку, сценарій або як частину програми. Коли файл отримано, він запускає функцію Lambda. Потім ця лямбда-функція завантажує інвентар у таблицю DynamoDB.

У цьому підрозділі описано створення сегменту S3 і налаштування його для запуску функції Lambda [12].

Кожен сегмент має мати унікальне ім’я, тому до назви сегмента додано випадкове число (рис.2.2)

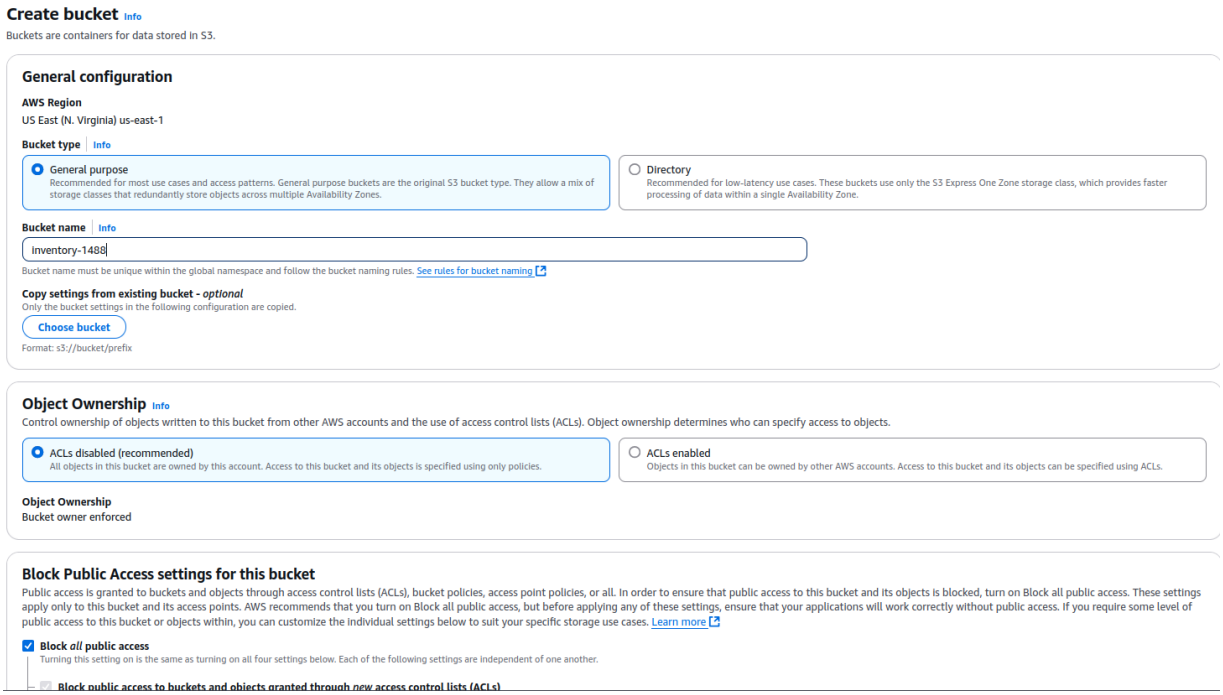

Рисунок 2.2 - Створення S3 bucket для завантаження інвентаризації

В цей сегмент S3 bucket надсилаються звіти про інвентаризацію.

Сегмент було налаштовано для автоматичного запуску функції лямбда (Lambda) під час завантаження файлу (рис.3.3) [13].

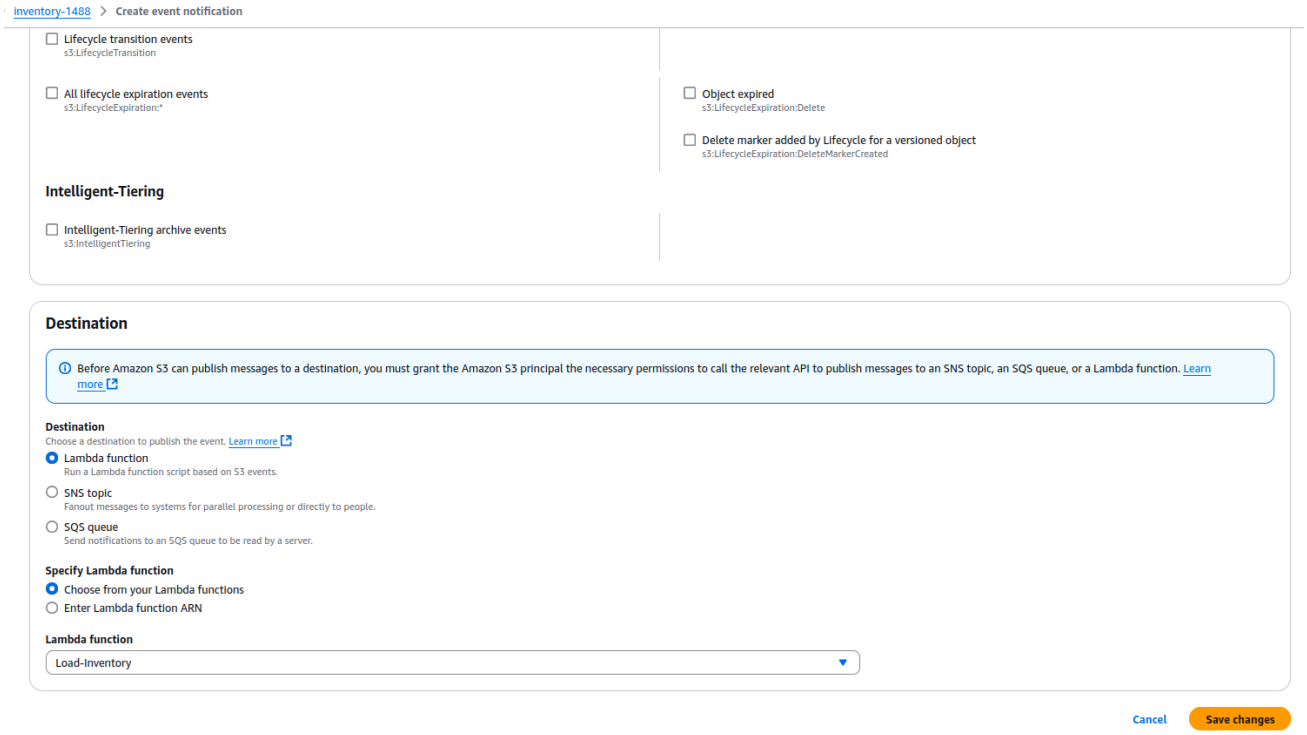

Рисунок 2.3 - Створення сповіщення про подію для lambda функції в S3

На сторінці «Створити сповіщення про подію» налаштовані ці параметри:

- Name: Load-Inventory
- Event types: All object create events o Destination: Lambda Function
- Lambda function: Load-Inventory

Коли об’єкт створюється у сегменті, ця конфігурація повідомляє Amazon S3 запустити функцію Load-Inventory Lambda, яка була створена раніше. Після цього сегмент готовий для отримання файлів інвентаризації!

2.4 Тестування процесу завантаження

Далі були протестовані процеси завантаження. Було завантажено файли інвентаризації, а потім перевірено успішність завантаження.

Були завантажені наступні файли інвентаризації: inventory-berlin.csv; inventory-calcutta.csv; inventory-karachi.csv ;inventory-pusan.csv; inventory-shanghai.csv; inventory-springfield.csv.

Ці файли є файлами інвентаризації, які можна використовувати для перевірки системи. Це файли зі значеннями, розділеними комами CSV. В додатку Б показано вміст цих файлів.

Було завантажено один з тестових звітів, а саме inventory-berlin.csv (рис.2.4).

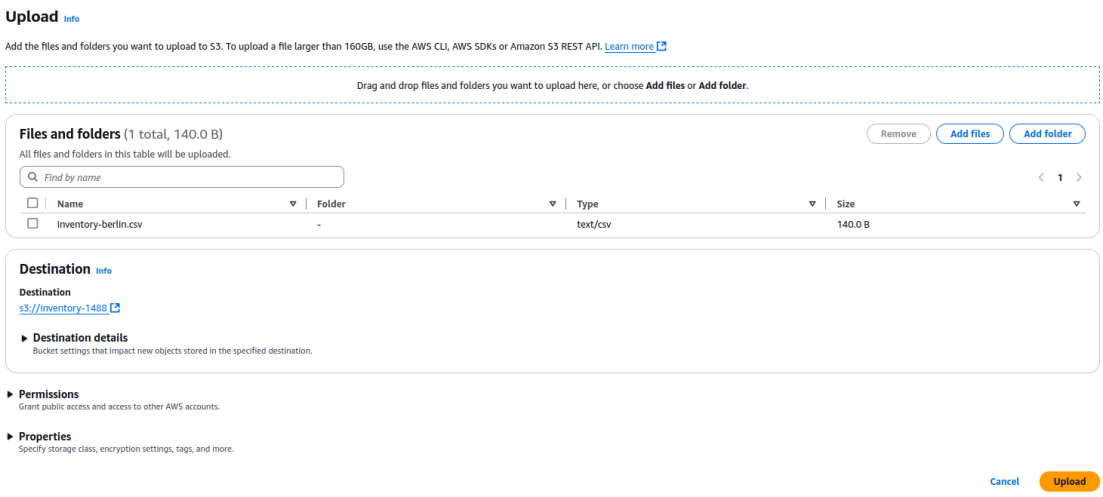

Рисунок 2.4 - Завантаження тестового звіту про інвентаризацію

Amazon S3 автоматично запустив функцію Lambda, яка завантажила дані в таблицю DynamoDB.

Для перегляду результатів застосовано безсерверну програму Dashboard(рис.2.5).

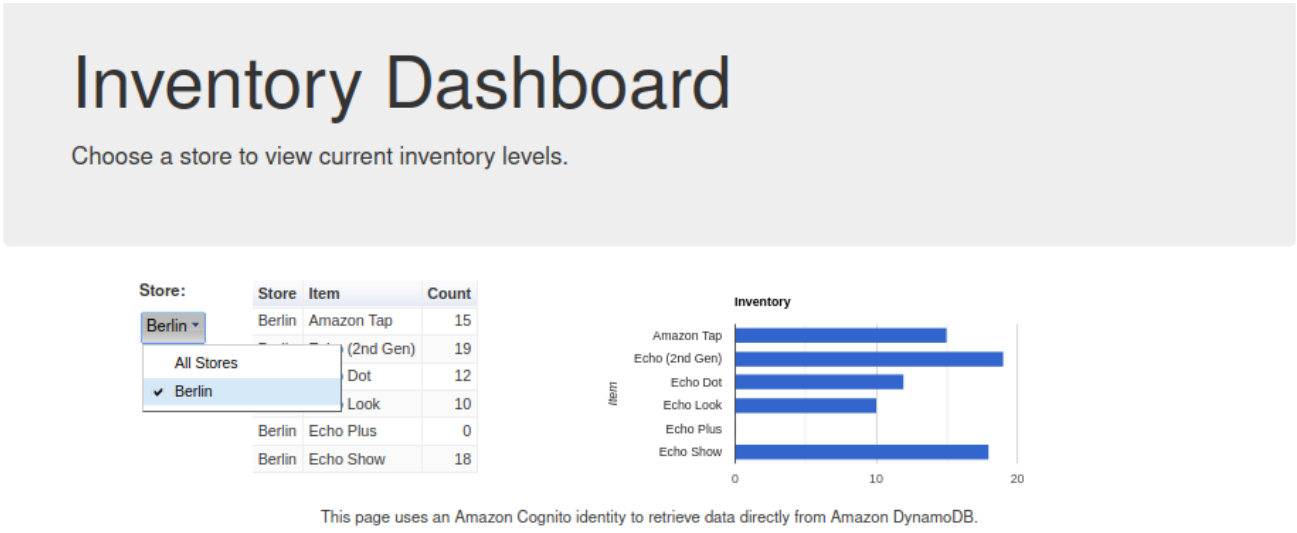

Рисунок 2.5 - Звіт про інвентаризацію в dashboard

Для цього була відкрита нова вкладка веб-браузера з відповідною URL-адресою.

Відкрилася програма інформаційної панелі, що відобразила дані інвентаризації, які були завантажені у сегмент. Дані отримані з DynamoDB, що доводить, що завантаження успішно запустило функцію Lambda.

Програма інформаційної панелі подається як статична веб-сторінка з Amazon S3. Перевірка автентичності інформаційної панелі виконується через Amazon Cognito як анонімний користувач anonymous user, що надає достатні дозволи для інформаційної панелі для отримання даних із DynamoDB.

Також була переглянута безпосередньо таблиця DynamoDB. Було вибрано таблицю Inventory вкладку Items. Відобразилися дані з файлу інвентаризації. Вони показують магазин Store, товар Item і кількість інвентарю Count.

2.5 Налаштування сповіщень

Метою створення моделі є повідомити персонал управління запасами, коли в крамниці закінчяться запаси товару. Для цієї функції безсерверного сповіщення був використаний сервіс Amazon SNS. Amazon SNS — це гнучка, повністю керована служба обміну повідомленнями та мобільними сповіщеннями для публікації/підписки [14]. Вона доставляє повідомлення кінцевим точкам і клієнтам, які підписалися. За допомогою Amazon SNS можна розповсюджувати повідомлення великій кількості передплатників, включаючи розподілені системи та служби та мобільні пристрої.

Була створена тема NoStock (рис.2.6).

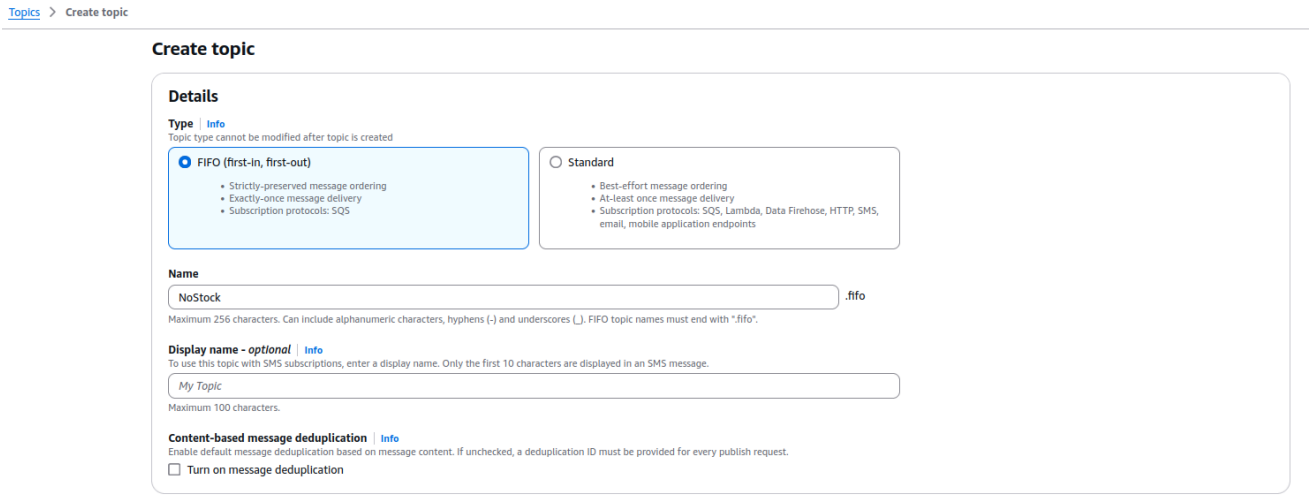

Рисунок 2.6 - Створення нового SNS topic

Щоб отримувати повідомлення була оформлена підписка на цю тему. Можна отримувати сповіщення декількома способами, наприклад SMS або електронною поштою (рис.2.7).

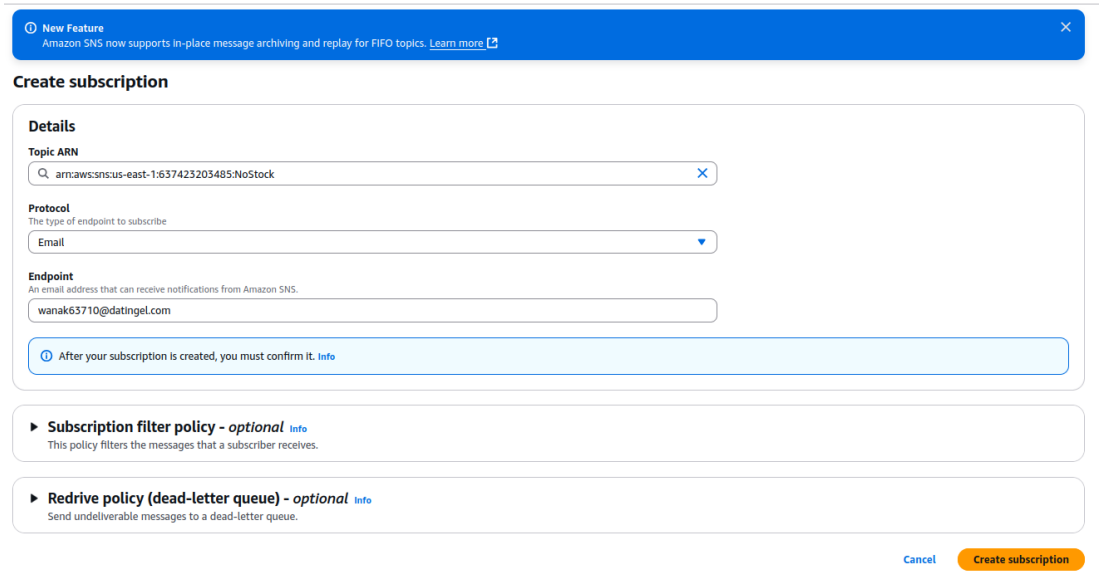

Рисунок 2.7 - Створення підписки на AWS SNS

Був обраний протокол email.

Після створення підписки електронною поштою, було отримано повідомлення електронної пошти з підтвердженням.

Було відкрито повідомлення та вибрано посилання Confirm subscription (Підтвердити підписку).

Інші повідомлення, надіслані в тему SNS, були перенаправленні на вказану електронну адресу.

2.6 Лямбда-функції для надсилання сповіщень

Можна змінити існуючу функцію Load-Inventory Lambda, щоб перевіряти рівні запасів під час завантаження файлу. Однак така конфігурація не є хорошою архітектурною практикою. Замість того, щоб перевантажувати функцію Load-Inventory бізнес-логікою, було створено іншу функцію Lambda, яка запускається, коли дані завантажуються в таблицю DynamoDB. Ця функція запускається потоком DynamoDB stream.

Такий архітектурний підхід має кілька переваг:

- кожна лямбда-функція виконує одну певну функцію. Ця практика робить код простішим і зручнішим у супроводі;
- додаткову бізнес-логіку можна додати, створивши додаткові функції Lambda. Кожна функція працює незалежно, тому це не впливає на наявні функції.

У цьому підрозділі описана ще одна функція Lambda, яка переглядає інвентар під час завантаження в таблицю DynamoDB. Якщо функція Lambda помічає, що товару немає в наявності, вона надсилає сповіщення через тему SNS, яка була створена раніше.

Налаштовано ці параметри (рис.2.8):

- Function name: Check-Stock
- Runtime: Python 3.8

Розгорнуто Choose or create an execution role.

- Execution role: Use an existing role
- Existing role: Lambda-Check-Stock-Role

Цю роль було налаштовано з дозволами надсилати сповіщення до Amazon SNS.

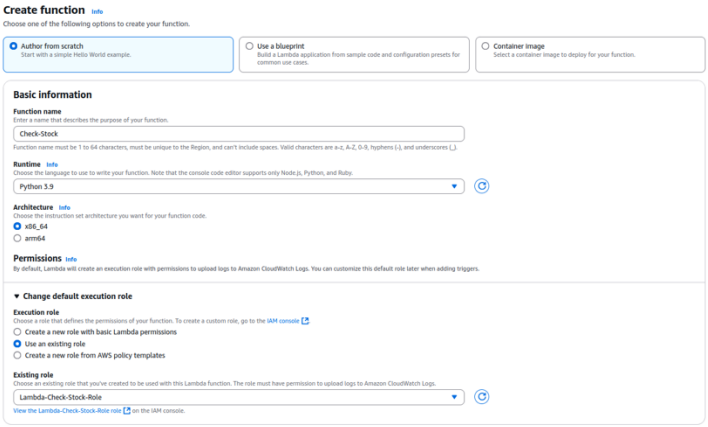

Рисунок 2.8 - Створення додаткової функції для полегшення безсерверної архітектури

Новий код (Додаток Б) виконує такі дії: переглядає вхідні записи; якщо кількість запасів дорівнює нулю, надсилає повідомлення в тему NoStock SNS.

Далі було налаштовано функцію (рис.2.9), щоб вона запускалася, коли дані додаються до таблиці Inventory у DynamoDB.

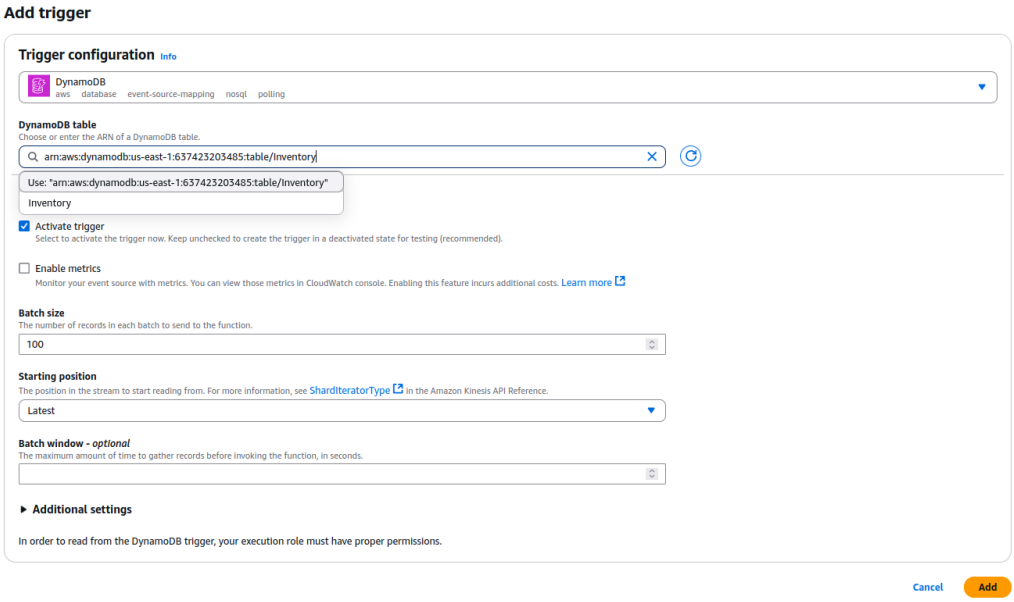

Рисунок 2.9 - Додавання тригеру події до lambda функції

Тобто налаштовані ці параметри:

- Select a trigger: DynamoDB;
- DynamoDB Table: Inventory.

Далі система (рис. 2.10.) була протестована.

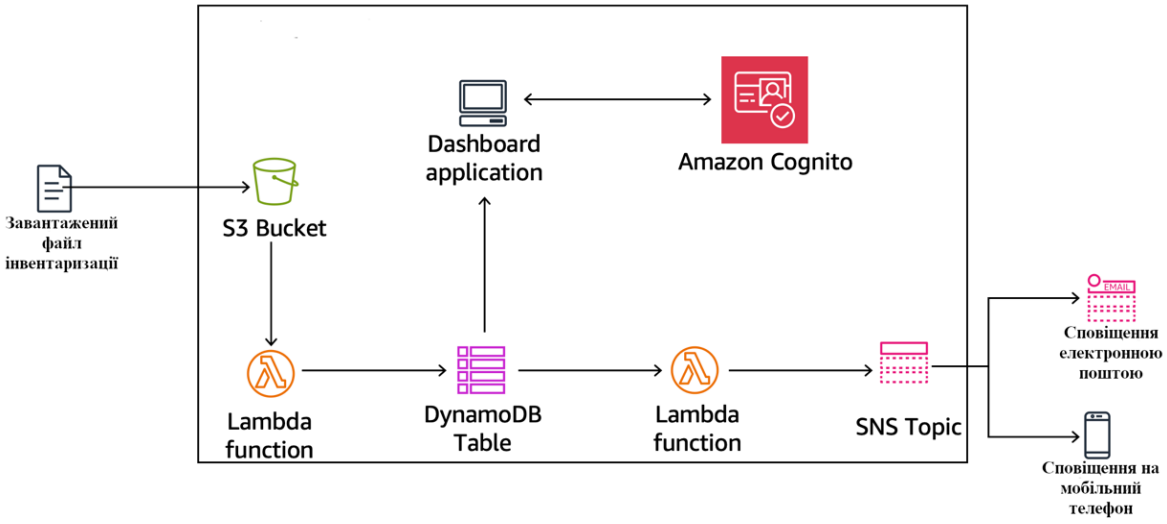

Рисунок 2.10 - Фінальна архітектура цього розділу

2.7 Тестування системи

Був завантажений файл інвентаризації в Amazon S3, який запустив оригінальну функцію Load-Inventory. Ця функція завантажила дані в DynamoDB, яка потім запустила нову функцію Check-Stock Lambda. Функція Lambda виявила товар із нульовим запасом і надіслала повідомлення до Amazon SNS. Тоді Amazon SNS сповістила через електронну пошту.

Тепер можна використовувати меню All Stores «Крамниці» для перегляду інвентарю в обох крамницях.

Крім того, отримані сповіщення через електронну пошту про те, що в крамниці немає в наявності товару (у кожному файлі інвентаризації є один товар, якого немає в наявності).

Сповіщення приходять через кілька хвилин і можна завантажувати інший файл інвентаризації. Активація тригера DynamoDB іноді може тривати кілька хвилин.

2.8 Висновки до другого розділу

Запропоновано модель без серверної архітектури на AWS .

Реалізовано виклик функції Lambda з Amazon S3 і Amazon DynamoDB.

Налаштовано Amazon SNS для надсилання сповіщень.

3 РЕЗУЛЬТАТИ СТВОРЕННЯ СИСТЕМИ ОБЛІКУ ЗАПАСІВ

Бізнес кафе процвітає. Власники бізнесу хочуть отримувати щоденні звіти про продажі продуктів, які продаються на веб-сайті кафе. Вони використовуватимуть цей звіт для планування замовлень інгредієнтів і моніторингу впливу рекламних акцій продукту.

Початкова ідея менеджера полягає в тому, щоб використовувати один із екземплярів веб-сервера Amazon Elastic Compute Cloud (Amazon EC2) для створення звіту. Менеджер встановлює завдання cron на примірнику веб-сервера, який надсилає повідомлення електронної пошти зі звітами про щоденні продажі. Однак завдання cron знижує продуктивність веб-сервера, оскільки воно потребує ресурсів.

Менеджер розповідає експерту AWS про роботу cron і те, як вона знижує продуктивність веб-додатку. Експерт радить менеджеру відокремити неважливі для бізнесу завдання звітності від робочого екземпляра веб-сервера. Після того, як менеджер розглянув переваги та недоліки свого поточного підходу, він вирішив, що не хоче уповільнювати веб-сервер. Він також розглядає можливість запуску окремого екземпляра EC2, але його турбує вартість цілодобового запуску екземпляра, коли він потрібен лише на короткий час щодня.

Менеджер вирішує, що запуск коду генерації звіту як функції AWS Lambda буде працювати, і це також знизить витрати [15]. Сам звіт можна надіслати на електронну адресу власників бізнесу через службу простих повідомлень Amazon SNS. У цьому розділі досліджується роль менеджера, щоб реалізувати код щоденного звіту як лямбда-функцію.

3.1 Вхідні данні

Початкова архітектура проекту наведена на рис. 3.1.

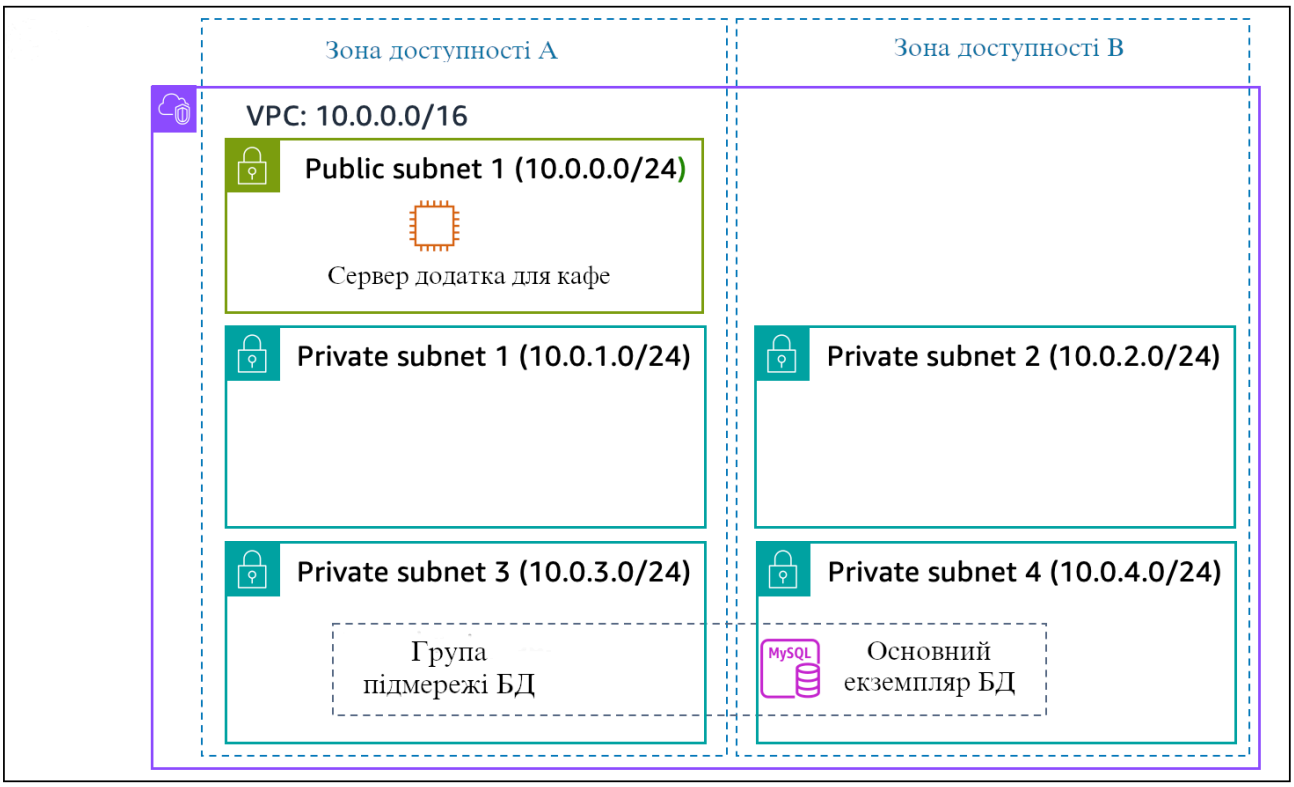

Рисунок 3.1 - Початкова архітектура проекту

Проект побудовано на базі архітектури, яка має декілька зон доступності, між якими розподілені декілька підмереж, в одній знаходиться база даних, а в іншій група підмережи, яка пов’язана з цією базою даних.

Досліджується архітектура (рис. 3.2) з наступними елементами: функція Lambda у віртуальній приватній хмарі VPC, яка підключається до бази даних (Amazon Relational Database Service - Amazon RDS) із даними про продажі в кафе; функція Lambda, яка створює та виконує звіт про продажі; запланована подія, яка щоп’ятниці, о 17:00 ініціює лямбда-функцію звіту про продажі[16].

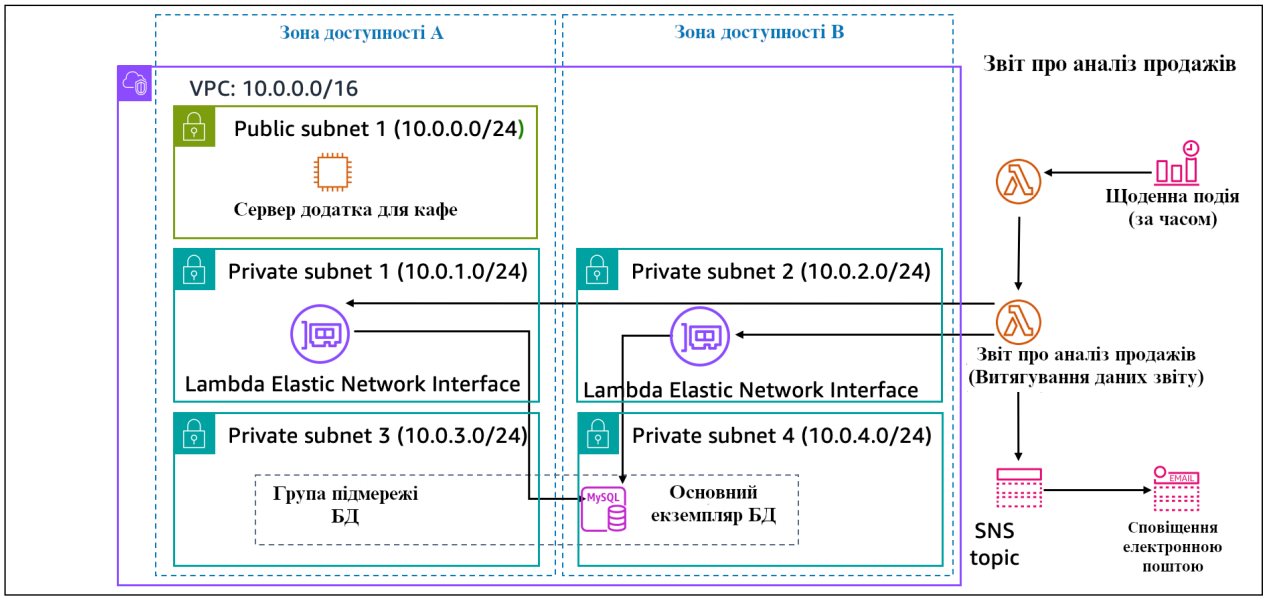

Рисунок 3.2 - Фінальна архітектура проекту

3.2 Вихідний код

Код завантажено з двох архівів “salesAnalysisReport” та “salesAnalysisReportDataExtractor”, з кодом Python, та додатковими теками, до нього, ці архіви отримані з документації AWS, в якості макету і потім перероблені під потреби дослідження, ці теки необхідні для побудови AWS Lambda.

3.3 Видобування даних

Перед налаштуванням функцію Lambda було надано доступ для неї до Amazon RDS, бо вони знаходяться в різних приватних підмережах, тому зв’язок між ними без додаткових налаштувань неможливий, це зроблено завдяки додаванню нової політики безпеки, до групи безпеки, щоб дозволити підключення, тому що lambda функція, яка потребує доступу до RDS, підключається до VPC через Elastic Network Interface ENI. ENI також асоціюється із власною групою безпеки.

Було створено нове правило для вихідного трафіку для Amazon RDS додано його як правило вихідного трафіку (outbound) до групи безпеки [17] (рис 3.3).

Створена група безпеки з наступними параметрами:

– Security group name:LambdaSG;
– VPC: Lab VPC
– Outbound Rules:
  - Type: All traffic
  - IP addresses: 0.0.0.0/0

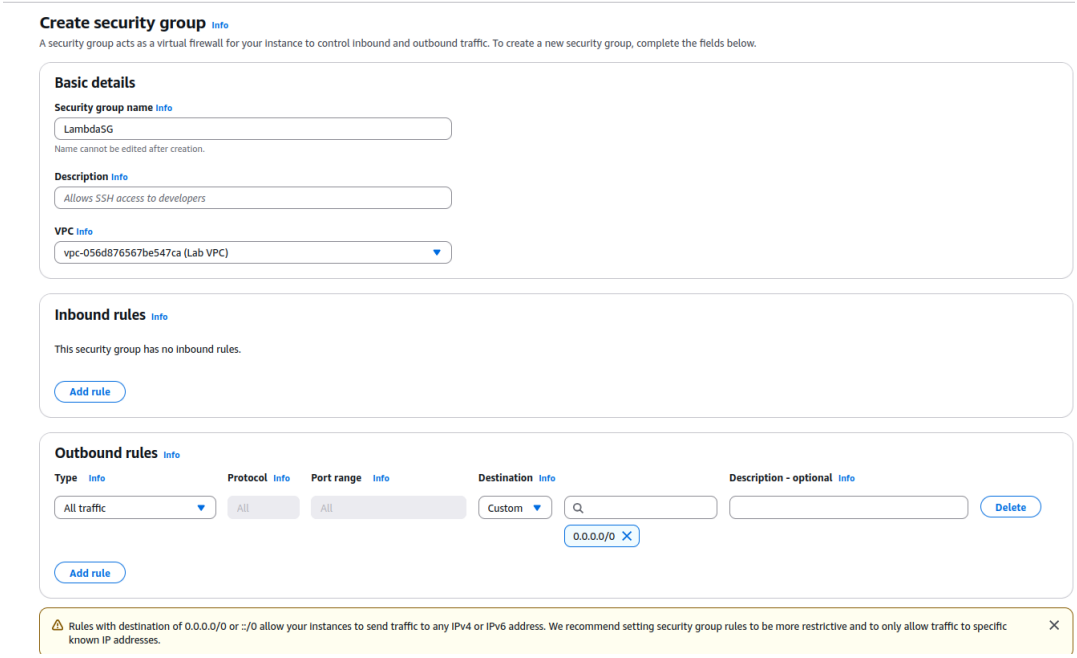

Рисунок 3.3 - Панель створення нової групи безпеки

Після створення нової групи безпеки налаштовано наявну групу безпеки бази даних шляхом додавання до неї нового правила для вхідного трафіку [17].

В розділі “Inbound rules”, обрано тип “MySQL/Aurora” та додано додаткове правило зі створеної до цього групи безпеки “LambdaSG”(рис. 3.4).

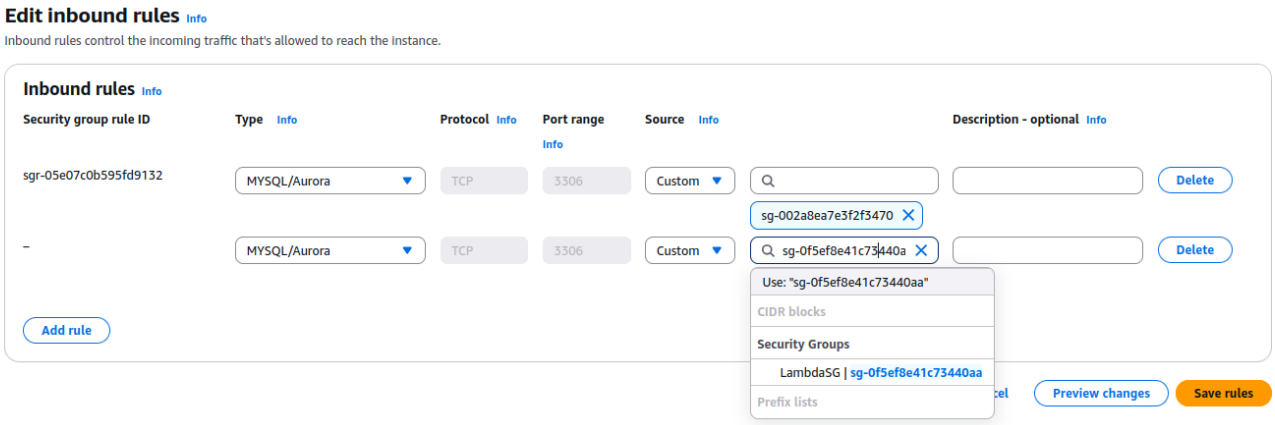

Рисунок 3.4 - Панель налаштувань вхідного трафіку

Після налаштування з'єднання між Amazon RDS і AWS Lambda, була створена нова Lambda функція (рис. 3.5), яка робить аналіз по продажам [18].

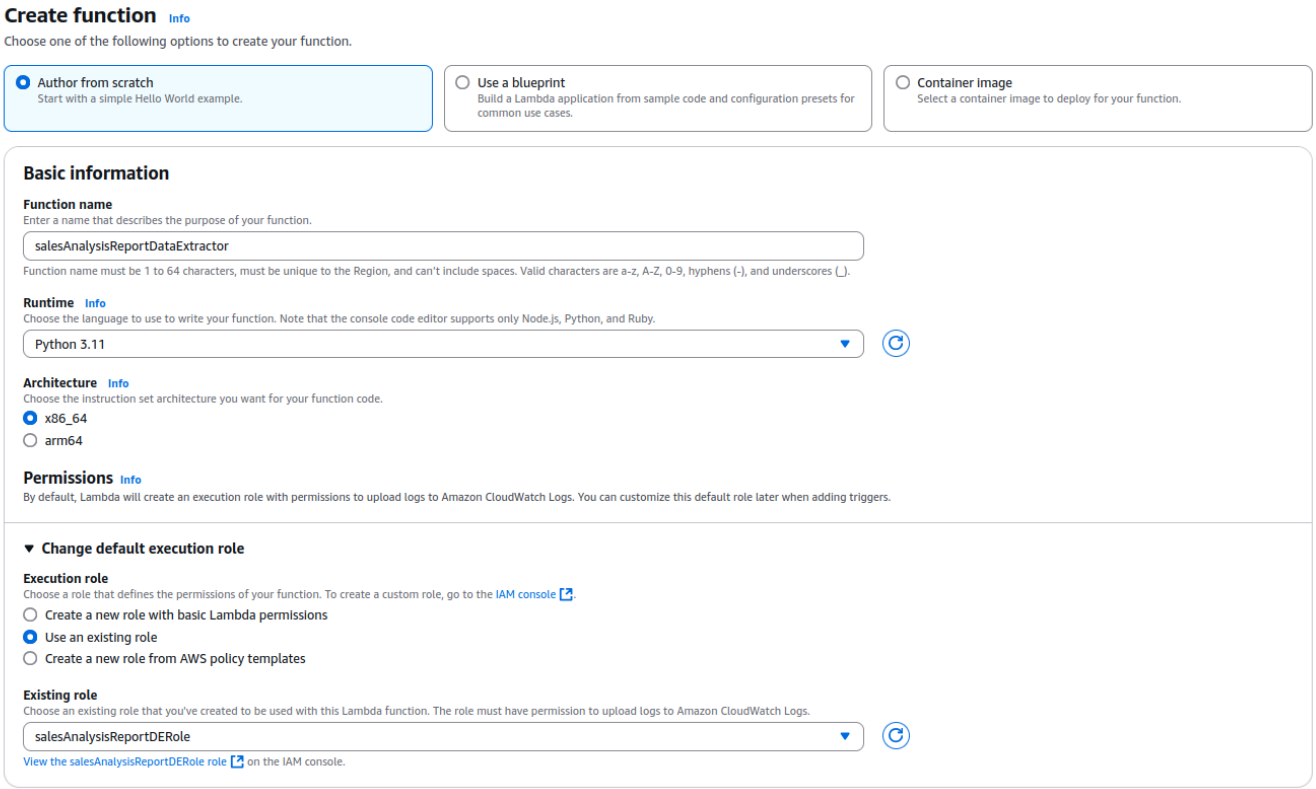

Рисунок 3.5 Створення lambda функції

При створенні lambda функції прописані наступні параметри:

- Function name: salesAnalysisReportDataExtractor
- Runtime: Python 3.11
- Роль: salesAnalysisReportDERole

Для VPC прописані наступні параметри:

– VPC: Lab VPC
– Підмережі: Private subnet 1 та Private subnet 2
– Security Group: LambdaSG

В підмережах були вказані private subnet 1 та private subnet 2 тому що передбачено ще створення додаткової лямбда функції, а так як вони знаходяться в різних зонах доступності заздалегідь прописано ці підмережі.

Після створення функції, було завантажено python код в вигляді архіву про який згадувалось на початку цього розділу, даний основний код знаходиться в додатку Б. Цей код працює наступним чином lambda-функція викликається через AWS, вхідні параметри (конфігурація бази даних) передаються через event, lambda підключається до MySQL бази даних у приватній підмережі виконує SQL-запит для збору даних про продажі, та повертає результат, який може бути використаний іншими сервісами AWS.

Після завантаження архіву, код та додаткові файли доступні для перегляду, після цього було ще додано опис до лямбда функції “Lambda function to extract data from database”, та пам’ять яка виділена на цей код в центральному процесорі в розмірі 128 МБ та timeout в розмірі 30 секунд (рис. 3.6).

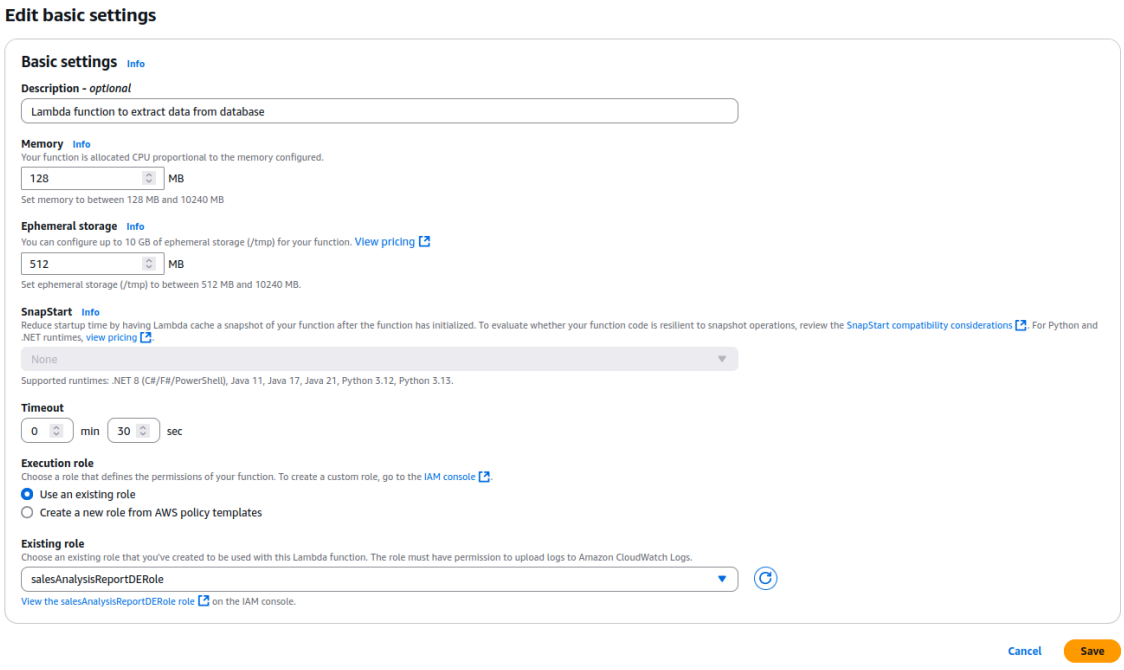

Рисунок 3.6 - Загальне налаштування lambda функції

3.4 Видобування даних

Далі після створення lambda функції яка робить звіт по продажам, було зроблена функція, яка за часом викликає її після чого передає цю функцію на SNS сервіс для відправки цього звіту на особисту пошту [1].

Для цього створена нова lambda функція з наступними параметрами (рис. 3.7):

– Function name: salesAnalysisReport
– Runtime: Python 3.11
– Роль: salesAnalysisReportRole

Як видно з вказаних параметрів вище в цей раз не додавались параметри VPC, тому що в першій створеній функції був прописаний доступ до зони доступності A, завдяки додаванню обох використовуваних приватних підмереж.

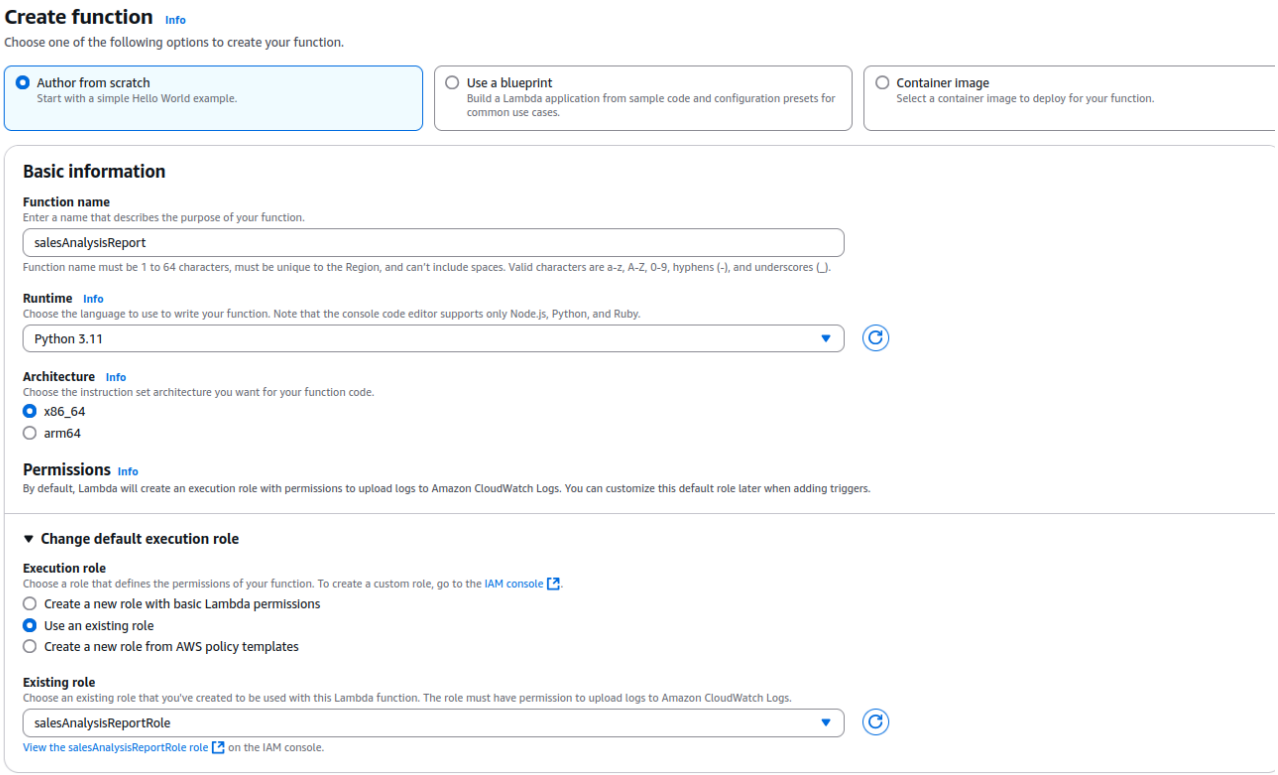

Рисунок 3.7 - Створення нової lambda функції

Після створення lambda функції було завантажено новий python код в вигляді архіву до неї, і так само як з попередньою функцією прописано опис “Lambda function to generate and send the daily sales report” та виділено пам’ять в такому ж розмірі 128МБ використання центрального процесора, та timeout в розмірі 30 секунд.

Після створення обох lambda функцій які будуть збирати данні з бази даних по продажам, та друга буде їх обробляти та надсилати, далі потрібно було додати до цього циклу сервіс який після всієї відпрацьованої роботи обох функції, надсилає зручним способом, цей звіт, для цього використовувався як і в минулому розділі SNS topic, для надсилання звіту було обрано спосіб надсилання на електрону пошту.

3.5 Тема листування

Було створено новий SNS topic, на сторінці (SNS-Simple Notification Service) в полі “Create topic” вписана назва “Simple Notification Service” та як “Display Name” вказано “Sales Report Topic”. Після створення теми SNS topic, було в налаштуваннях lambda функції “salesAnalysisReport” в розділі “Configuration” в підрозділі “Environment variables” додано нову змінну середовища, з ім’ям “topicARN” та значенням створеного SNS topic.(рис. 3.8)

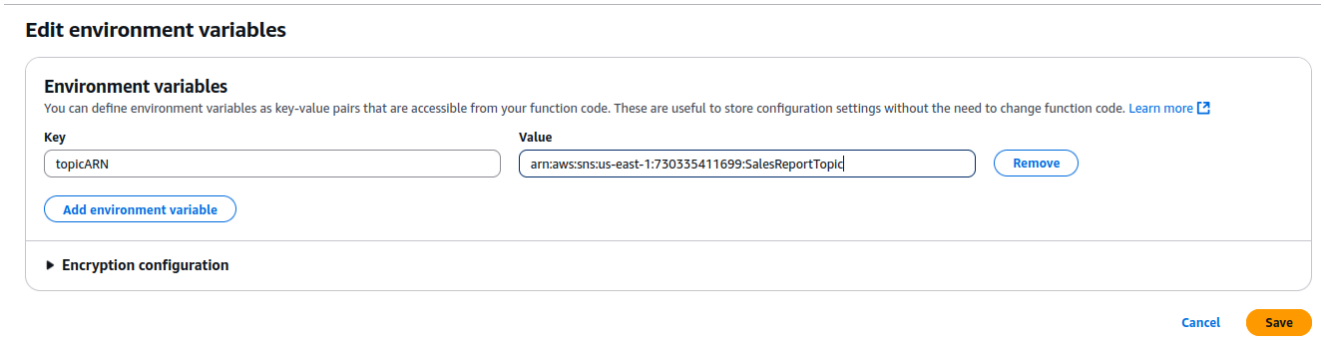

Рисунок 3.8 - Прописування змінних середовища

3.6 Підписка на тему листування

Після закінчення прив’язання lambda функції до SNS topic, була створена підписка на SNS електронної пошти. Тому в створеній темі SNS topic створена нова підписка для пошти куди надсилаються сповіщення (рис. 3.9).

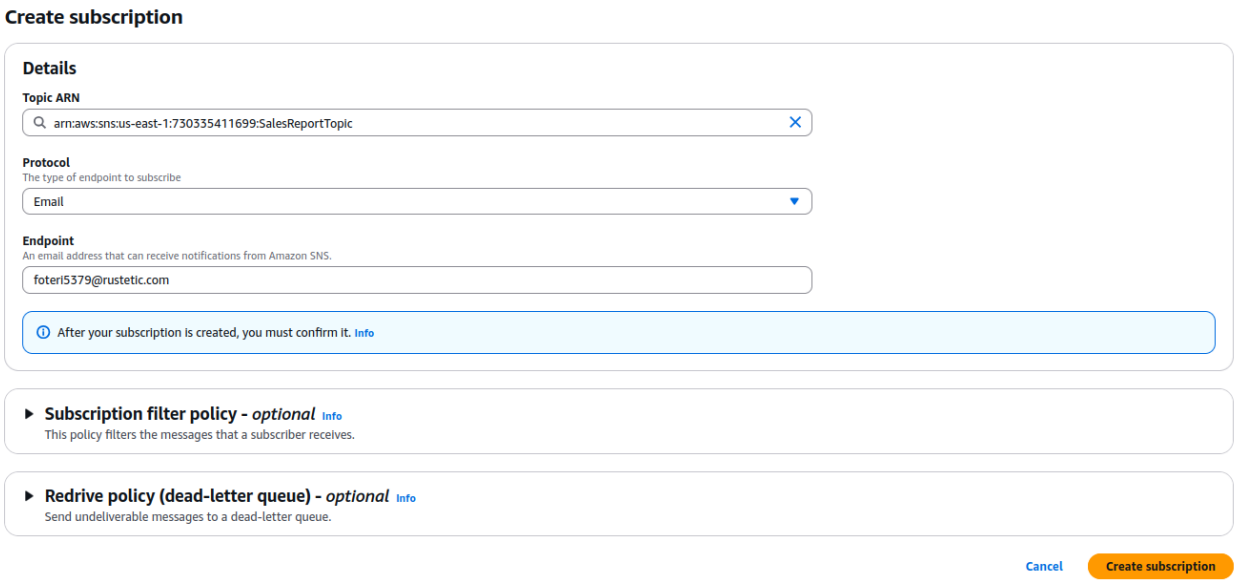

Рисунок 3.9 - Створення підписки на SNS

На поштову скриньку яку вказана в підписці надходить лист для підтвердження оформлення підписки, його було підтверджено. На цьому процес підписки на SNS сервіс був завершений (рис. 3.10).

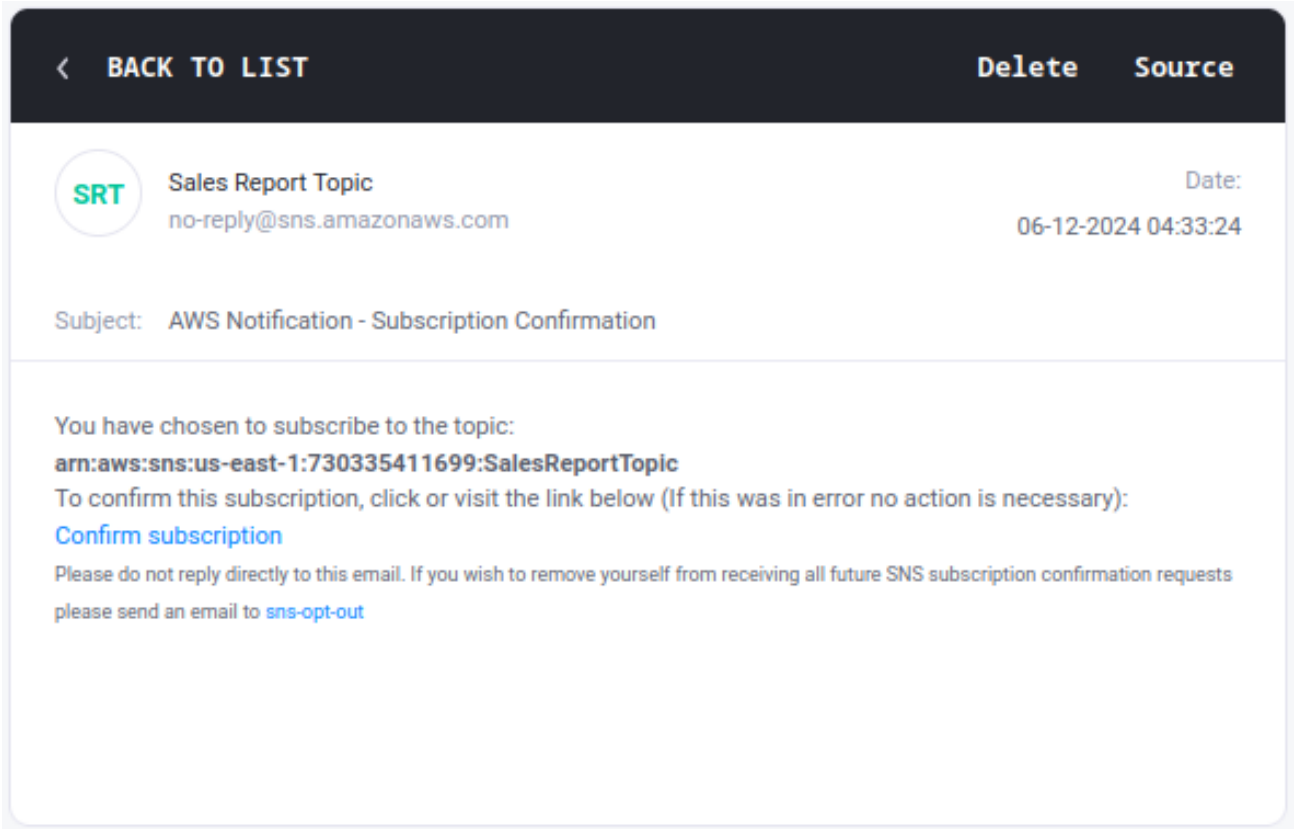

Рисунок 3.10 - Лист для підтвердження підписки

3.7 Тестування інфраструктури

Після всіх цих дій було перевірено повністю роботу всієї інфраструктури створення звітів. Це було зроблено в lambda функції “salesAnalysisReport” в розділі “Test”, і без параметрів ім’я просто натиснувши “Test”, і як видно з зображення нижче, тест відпрацював без збоїв, і закінчився успішно(рис. 3.11).

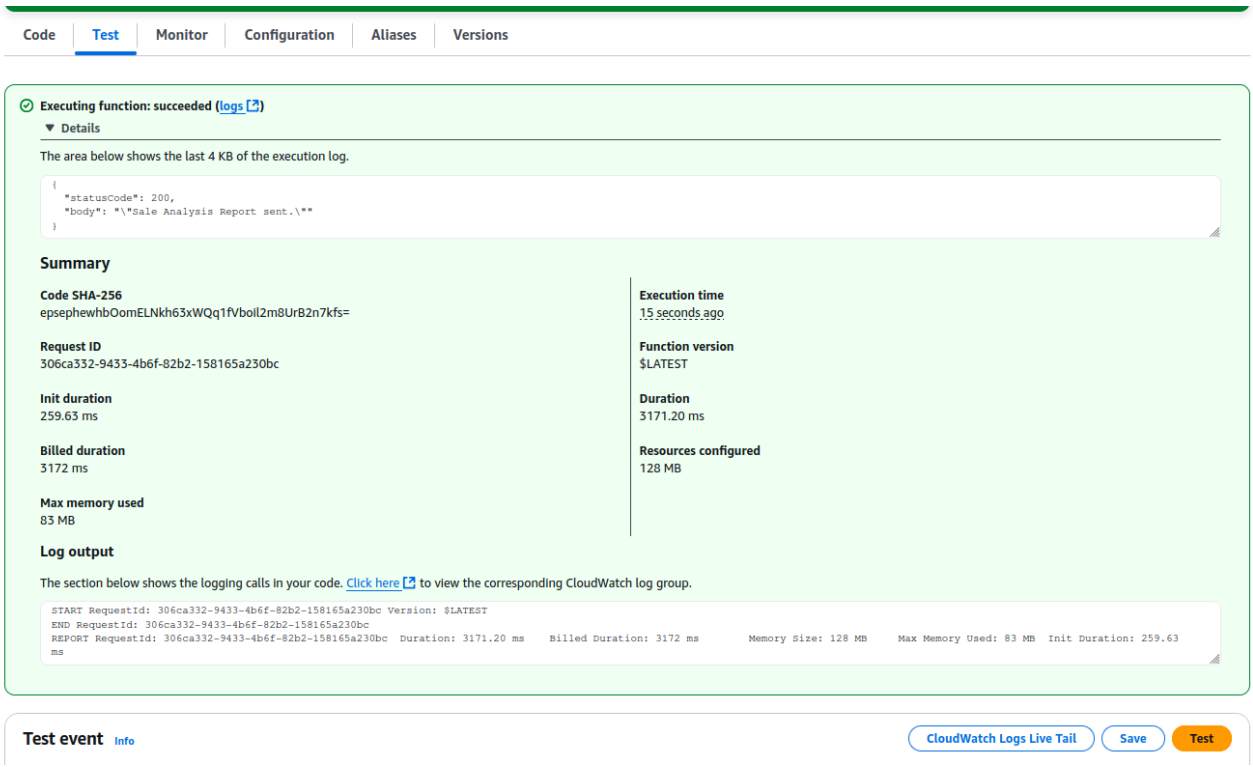

Рисунок 3.11 - Журнал події тестування звіту по продажам

Також перевіряючи поштову скриньку вказану для надсилання звітів, можна завіритись що все працює належним чином і звіт в читабельному зручному вигляді надсилається на пошту(рис. 3.12).

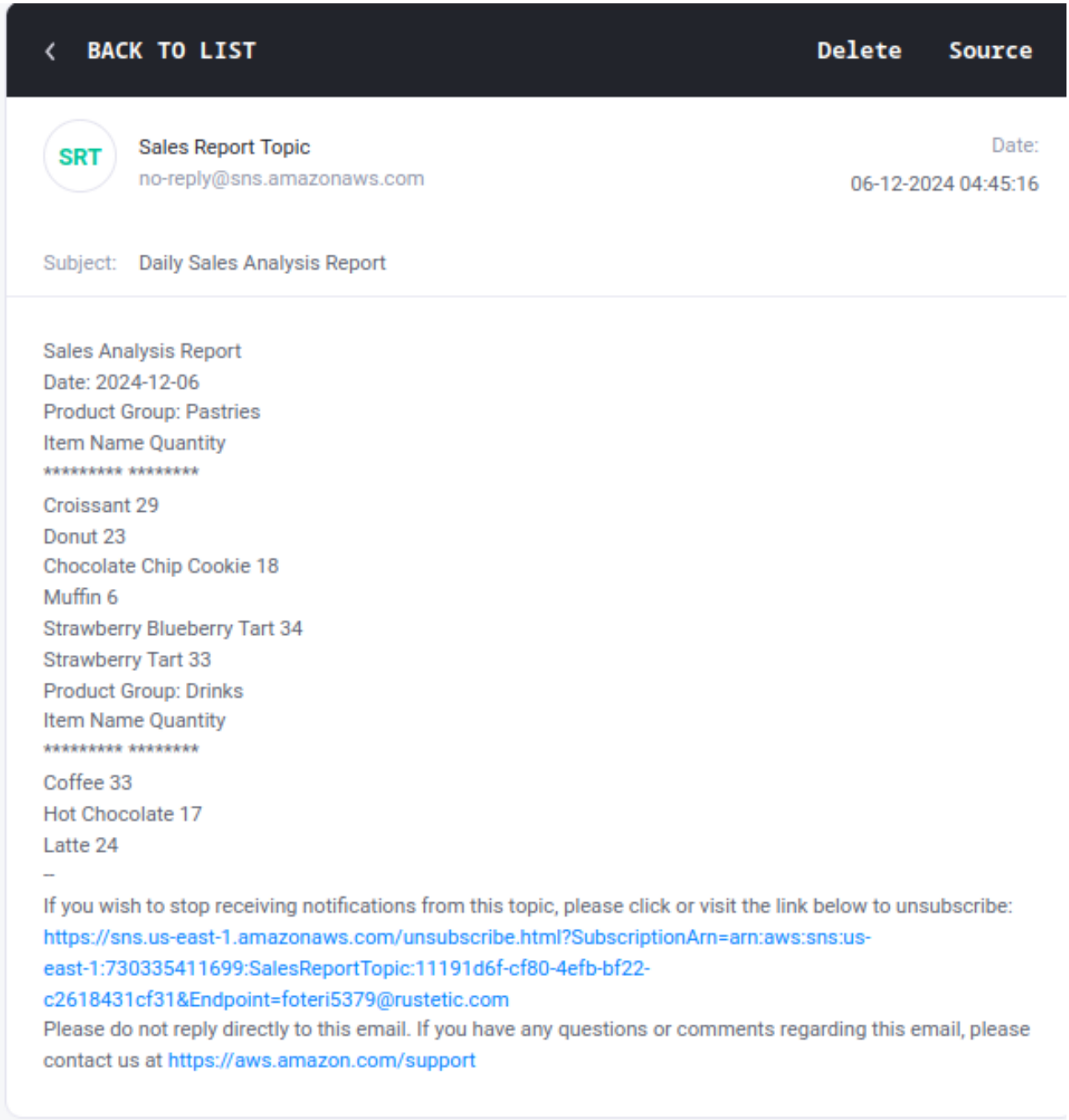

Рисунок 3.12 - Надісланий на пошту звіт

3.8 Щоденний звіт

Після підтвердження, що все працює як треба, було додано ще cron в якості тригера події, щоб ця lambda функція відпрацьовувала в один і той самий визначений час.

Для цього в lambda функції в розділі “Add trigger” додано “Event bridge” в якості тригера (рис 3.13) (додаток Б).

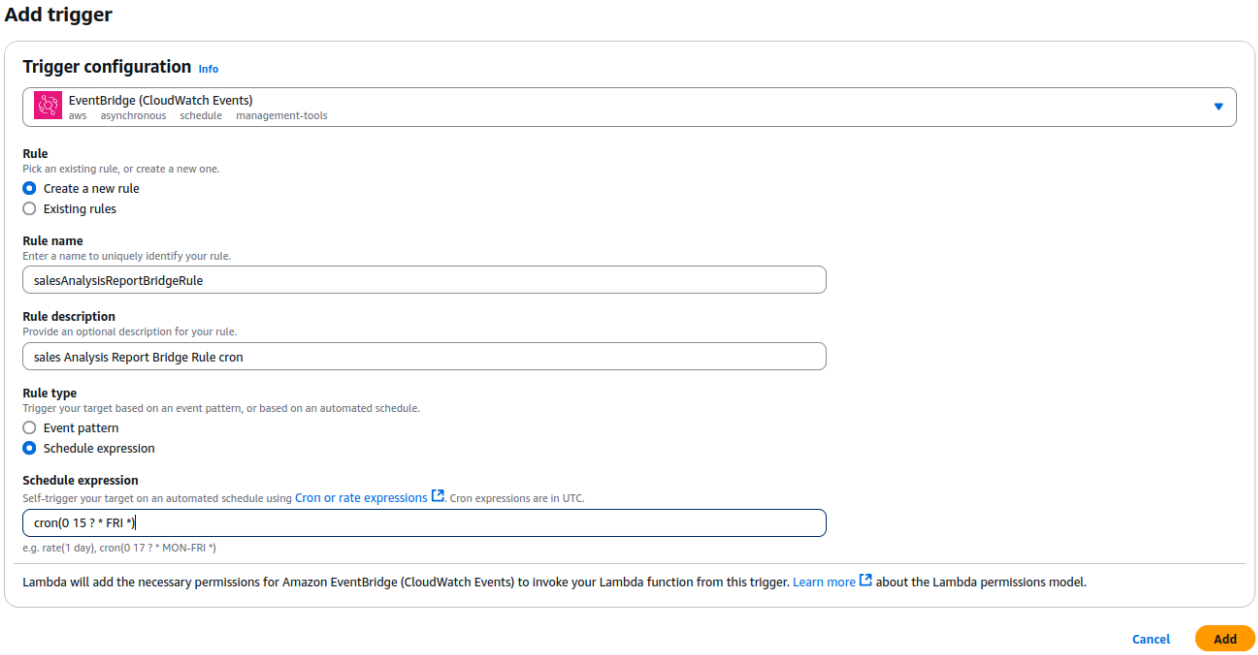

Рисунок 3.13 - Створення cron завдання

В cron, іде “0” хвилин тобто звіт приходить рівно в нуль хвилин, “15” це у котрій годині, і позначено 15, а не 17 як просив клієнт бо час там відлічується по UTC, і по Києву це UTC+2, за для того щоб звіти приходили рівно о 17, знак “?” позначає будь який день, знак “*” позначає будь який місяць, “FRI” позначає те що звіти будуть приходити кожної п’ятниці, а остання “*” позначає будь який рік.

3.9 Висновки до третього розділу

Коли менеджер завершує тестування звіту, він створює підписку на електронну пошту для власників бізнесу. Власники бізнесу раді отримати перший щоденний звіт від безсерверного рішення.

Менеджер задоволений тим, що він автоматизував звіти про продажі для кафе, що й надалі допомагатиме власникам бізнесу аналізувати щоденні продажі та планувати інвентаризацію кафе. Він також щасливий, що успішно навчився

використовувати AWS Lambda, Amazon SNS і Amazon EventBridge. Фактично, менеджер планує запровадити більше безсерверних і автоматизованих функцій звітності у веб-додатку кафе, щоб допомогти кафе розвиватися та керувати своїм бізнесом.

ВИСНОВКИ

У ході дослідження було реалізовано роз’єднану безсерверну архітектуру для автоматизації створення щоденних звітів про продажі, що вирішило низку проблем, характерних для тісно пов’язаних систем. Основний веб-сервер більше не виконує завдання, пов’язані зі звітністю, що дозволило зменшити затримки та підвищити загальну продуктивність системи. Завдяки перенесенню обробки даних у безсерверне середовище, зокрема через використання AWS Lambda, Amazon SQS і Amazon SNS[20], вдалося забезпечити асинхронну обробку запитів, яка ефективно справляється з великими обсягами даних без втрати швидкодії.

Однією з ключових проблем, яку вирішила запропонована архітектура, стало використання cron-завдань. У початковій архітектурі завдання cron працювали на основному веб-сервері, що створювало додаткове навантаження на його ресурси та знижувало продуктивність. Це особливо проявлялося під час пікових навантажень, коли одночасно оброблялися клієнтські запити та генерувалися звіти. Завдяки впровадженню AWS Lambda і Amazon EventBridge cron-завдання було перенесено у безсерверне середовище. Це дозволило виконувати їх в автоматизованому режимі у визначений час без навантаження на основний сервер.

Автоматизація cron-завдань забезпечила стабільну генерацію щоденних звітів про продажі, які надсилаються власникам кафе через Amazon SNS. Ці звіти допомагають власникам аналізувати щоденні продажі, планувати інвентаризацію та оцінювати ефективність маркетингових кампаній. Використання EventBridge дозволило налаштувати точний розклад виконання завдань, гарантуючи своєчасність доставки звітів.

Запропонована архітектура не тільки підвищила продуктивність, але й забезпечила значну гнучкість. Інтеграція хмарних сервісів дала змогу легко адаптувати систему до нових бізнес-вимог. Наприклад, власники кафе тепер мають можливість отримувати щоденні звіти про продажі, аналізувати попит на

продукцію та виявляти закономірності в поведінці клієнтів. Це дозволяє ефективно планувати інвентаризацію, зменшувати відходи та приймати більш обґрунтовані бізнес-рішення.

Важливою перевагою стало зниження витрат на підтримку інфраструктури. Безсерверна архітектура забезпечує оплату лише за фактичне використання ресурсів, що особливо актуально для завдань, які виконуються періодично. Крім того, система демонструє високу надійність: дані звітів залишаються доступними навіть під час технічного обслуговування основних серверів.

Результати роботи показали, що перехід на роз’єднану архітектуру дозволяє не лише вирішити існуючі проблеми, але й створити основу для подальшого вдосконалення. Реалізоване рішення демонструє, як сучасні хмарні технології можуть бути ефективно використані для автоматизації бізнес-процесів, підвищення продуктивності та оптимізації витрат. Це відкриває нові можливості для розвитку бізнесу та покращення взаємодії з клієнтами.

ПЕРЕЛІК ДЖЕРЕЛ

[1] Gupta, S. "Serverless Architectures on AWS." – Packt Publishing, 2017. – 250 с.

[2] Amazon Web Services. Amazon Simple Queue Service Developer Guide.[Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html.

[3] Wicherski, M. "Building Event-Driven Microservices: Leveraging Organizational Data at Scale." – O'Reilly Media, 2021. – 300 с.

[4] Fowler, M. "Patterns of Enterprise Application Architecture." – Addison-Wesley Professional, 2002. – 560 с.

[5] Villamizar, M., et al. "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud." – 2015 IEEE 8th International Conference on Cloud Computing. – pp. 931-938.

[6] Adzic, G., Chatley, R. "Serverless Computing: Economic and Architectural Impact." – ACM Digital Library, 2017. – 5(2): 409-415.

[7] Sharma, S. "Serverless Computing: A Comprehensive Guide to Architecting Serverless Applications." – Springer, 2020. – 230 с.

[8] Amazon Web Services. AWS Lambda Developer Guide. [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[9] Amazon Web Services. Amazon DynamoDB Developer Guide. [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html.

[10] Eyk, E. van, et al. "A Serverless Ecosystem for Edge Computing." – IEEE Internet Computing, 2018. – 22(4): 16-22.

[11] Roberts, M., Chapin, J. "Programming AWS Lambda." – O'Reilly Media, 2020. – 250 с.

[12] Amazon Web Services. AWS Well-Architected Framework. [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/architecture/well-architected/.

[13] Amazon Web Services. Amazon S3 User Guide. [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[14] Amazon Web Services. Amazon SNS Developer Guide. [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/sns/latest/dg/welcome.html.

[15] Glikson, A., Nellis, S. "AWS for Solutions Architects: Design and implement AWS solutions for enterprise applications." – Packt Publishing, 2021. – 400 с.

[16] Behrendt, M., et al. "AWS Lambda in Action: Event-Driven Serverless Applications." – Manning Publications, 2019. – 320 с.

[17] Bass, L., Weber, I., Zhu, L. "DevOps: A Software Architect's Perspective." – Addison-Wesley Professional, 2015. – 240 с.

[18] Amazon Web Services. Amazon RDS User Guide. [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html.

[19] Kumar, P. "Mastering AWS Lambda: Learn how to build and deploy serverless applications." – Packt Publishing, 2020. – 320 с.

[20] Parikh, J. "Implementing Serverless Architectures with AWS: Harnessing AWS for Scalable Cloud Applications."– Packt Publishing, 2021. – 350 с.