

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	<i>Ознайомлення із завданням. Уточнення ТЗ.</i>	<i>21.11</i>	ВИК
2	<i>Підбір літератури за темою роботи.</i>	<i>21.11-28.11</i>	ВИК
3	<i>Виконання розділу 1</i>	<i>29.11-04.12</i>	ВИК
4	<i>Виконання розділу 2</i>	<i>05.12-11.12</i>	ВИК
5	<i>Виконання розділу 3</i>	<i>12.12-28.12</i>	ВИК
7	<i>Оформлення презентаційного матеріалу, підготовка до захисту у ЕК</i>	<i>28.12-02.01</i>	ВИК

Дата видачі завдання 28 жовтня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

доц. Кривенко С.А
(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	6
Вступ.....	8
1 Аналіз сучасної архітектури	11
1.1 Сценарій.....	11
1.2 Фаза 1: Створення тісно пов'язаної програми	14
1.3 Фаза 2: Створення програми з відокремленою архітектурою.....	20
1.4 Висновки до першого розділу	30
2 Модель безсерверної архітектури	32
2.1 Опис моделі	32
2.2 Лямбда-функції для завантаження даних.....	33
2.3 Налаштування події Amazon S3	34
2.4 Тестування процесу завантаження.....	36
2.5 Налаштування сповіщень.....	38
2.6 Лямбда-функції для надсилання сповіщень.....	39
2.7 Тестування системи	42
2.8 Висновки до другого розділу.....	42
3 Результати створення системи обліку запасів.....	43
3.1 Вхідні данні	43
3.2 Вихідний код.....	45
3.3 Видобування даних.....	45
3.4 Видобування даних.....	49
3.5 Тема листування	50

3.6	Підписка на тему листування	51
3.7	Тестування інфраструктури.....	52
3.8	Щоденний звіт.....	54
3.9	Висновки до третього розділу	55
	Висновки	57
	Перелік джерел	59
	Додаток А. Слайди презентації.....	61
	Додаток Б. Код.....	70

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 – аббревіатура 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/підписки.

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 stop на екземплярі веб сервера, який надсилає ці щоденні повідомлення електронної пошти зі звітами про замовлення власникам. Однак завдання stop 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 stop 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. Створювалась ця архітектура за допомогою заздалегідь підготованого коду за для швидшого 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.

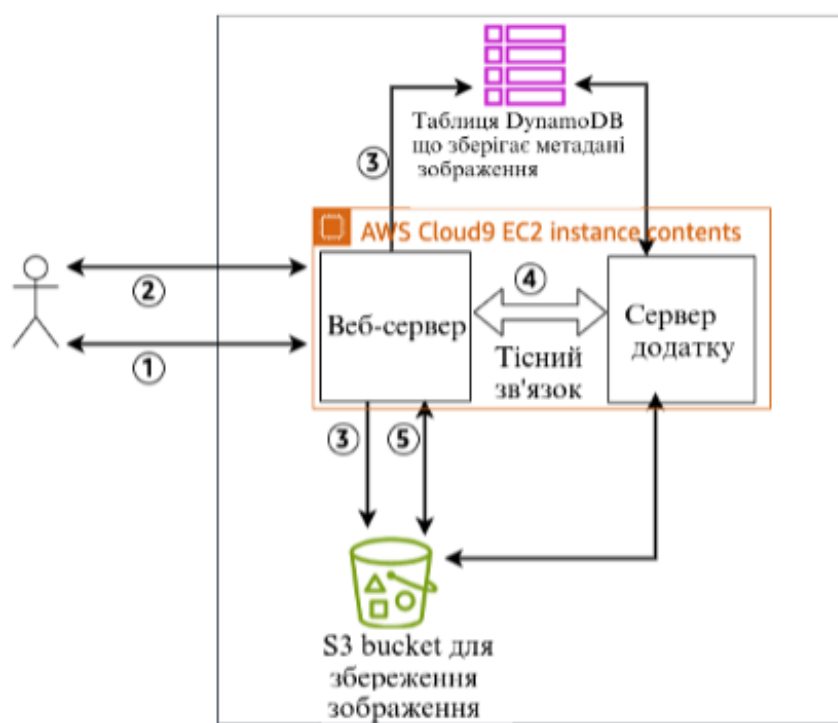


Рисунок 1.1 – Тісно пов'язана архітектура веб застосунку

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.

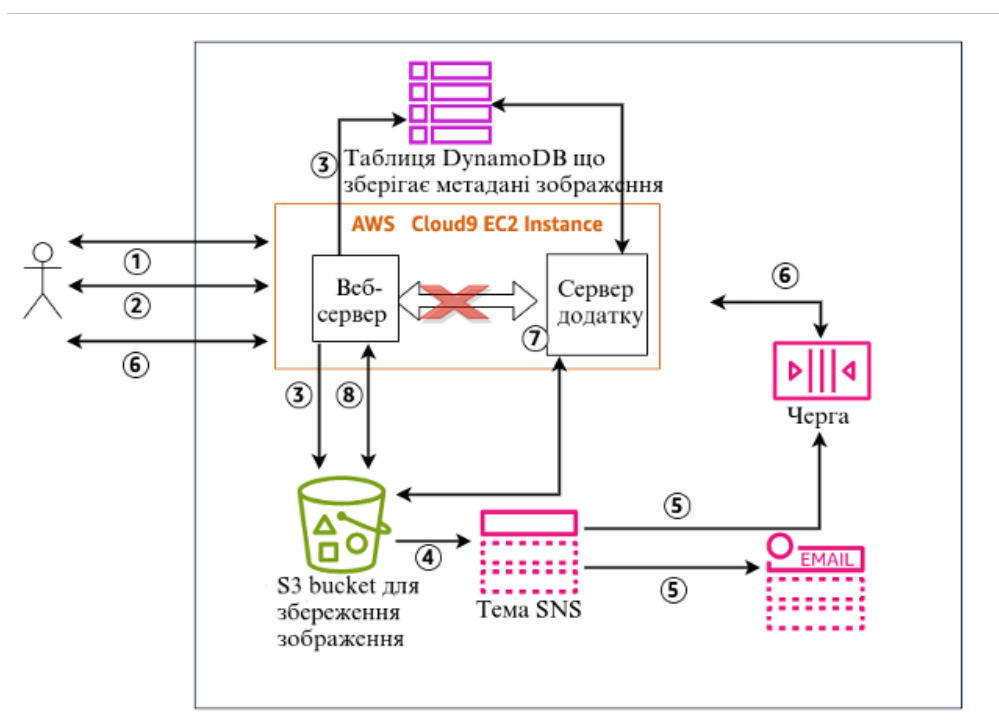


Рисунок 1.2 – Роз’єднана архітектура веб-застосунку

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

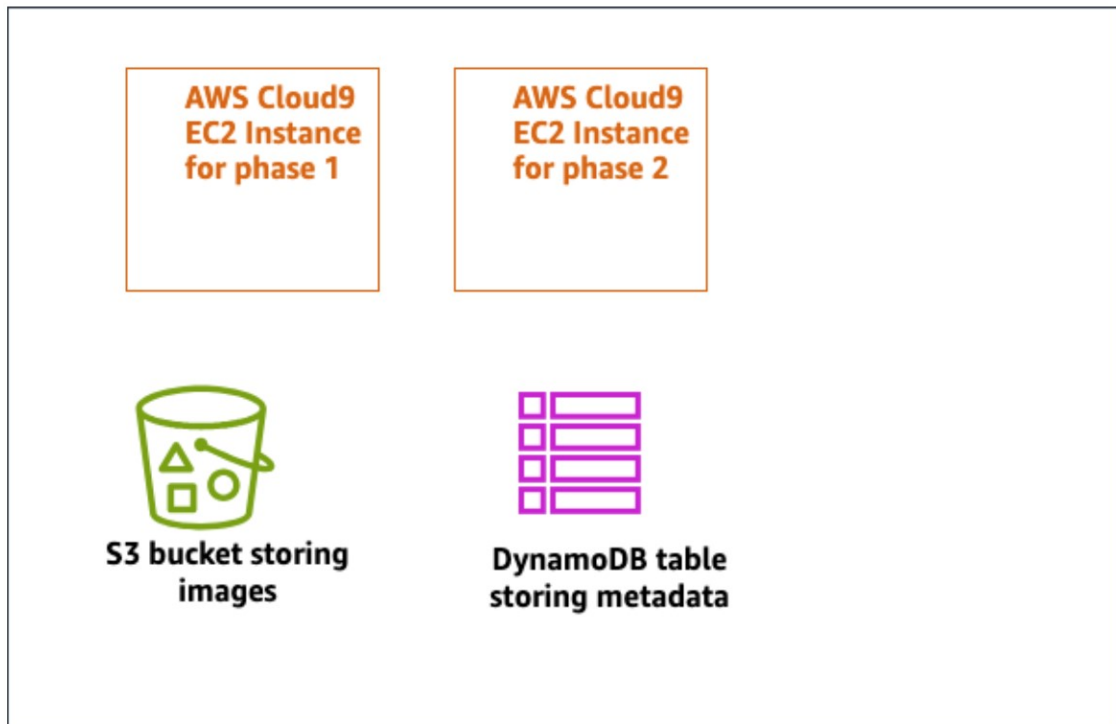


Рисунок 1.3 – Базова архітектура

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).

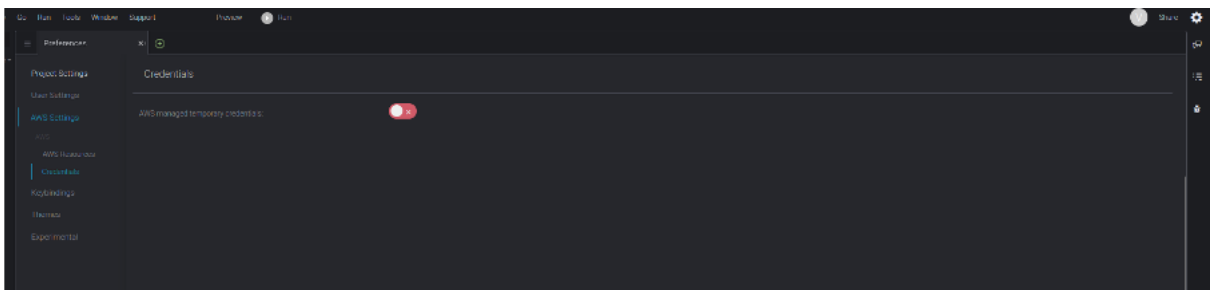


Рисунок 1.4 – Налаштування Cloud9

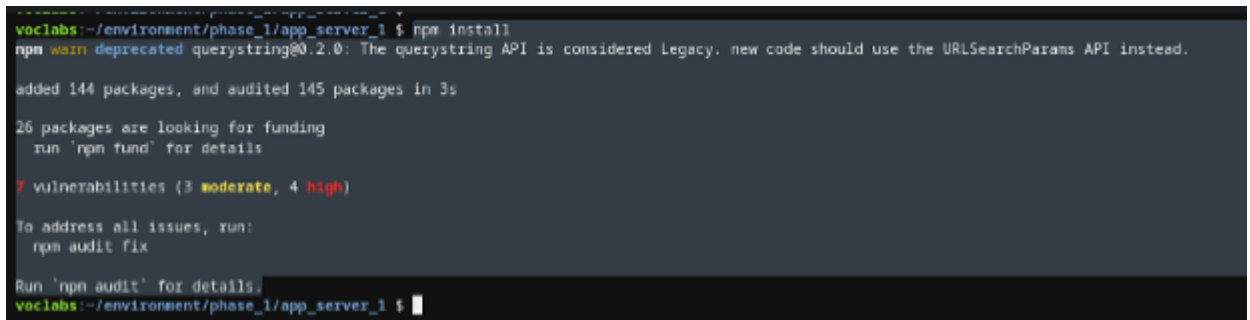
Крок 2. Завантаження файлів і встановлення програми

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).



```
voclabs:~/environment/phase_1/app_server_1 $ npm install
npm warn deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchParams API instead.

added 144 packages, and audited 145 packages in 3s

26 packages are looking for funding
  run 'npm fund' for details

7 vulnerabilities (3 moderate, 4 high)

To address all issues, run:
  npm audit fix

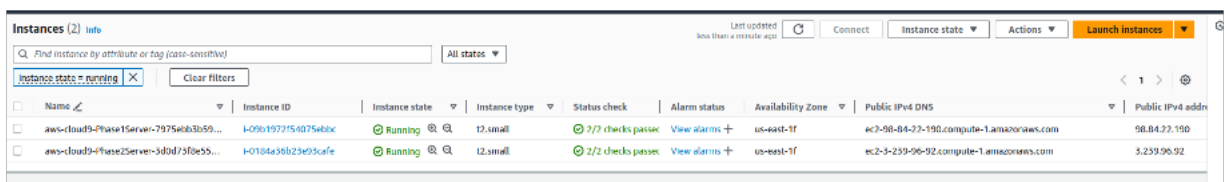
Run 'npm audit' for details
voclabs:~/environment/phase_1/app_server_1 $
```

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).



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
aws-cloud9-Phase1Server-7975a6b3b50...	i-0961977f54075abbc	Running	t2.small	2/2 checks passed	View alarms	us-east-1f	ec2-98-84-23-190.compute-1.amazonaws.com 98.84.23.190
aws-cloud9-Phase2Server-3d3d73f0e55...	i-0184a38b23493cafe	Running	t2.small	2/2 checks passed	View alarms	us-east-1f	ec2-3-239-96-92.compute-1.amazonaws.com 3.239.96.92

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) обрано «Anywhere-Ipv4» (рис. 1.7).

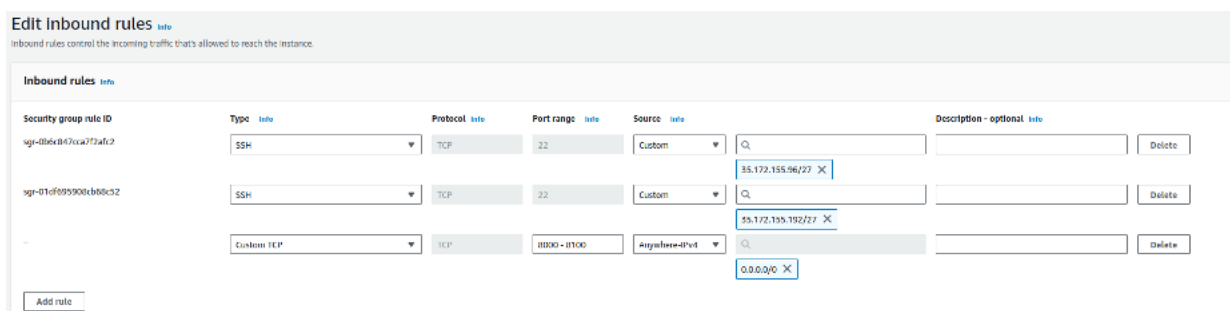


Рисунок 1.7 – Налаштування безпекових політик для Phase 1

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.

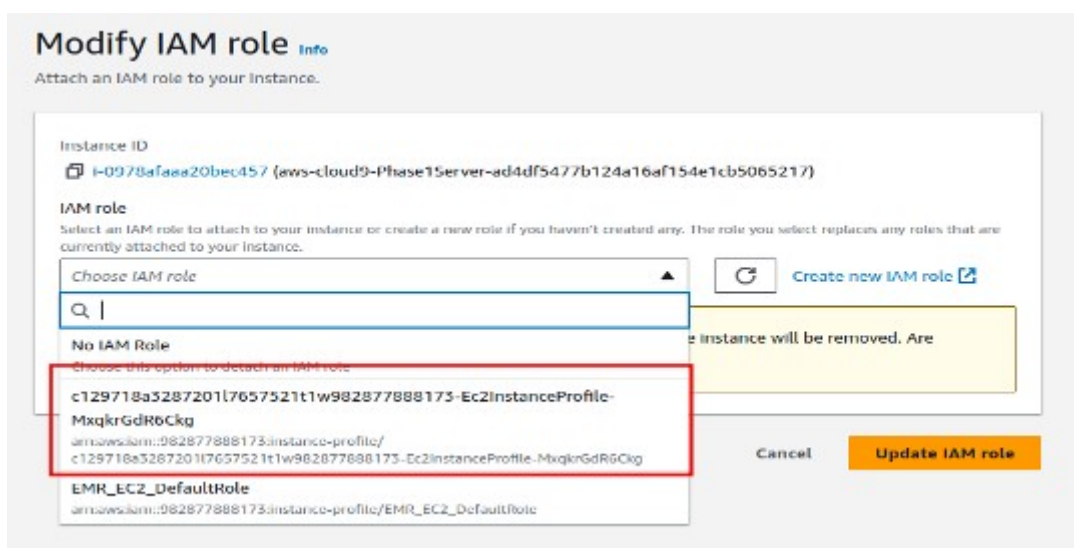


Рисунок 1.8 – Налаштування IAM Role

Крок 4. Налаштування дозволів сегмента S3

Після завершення налаштувань EC2 налаштовано 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).

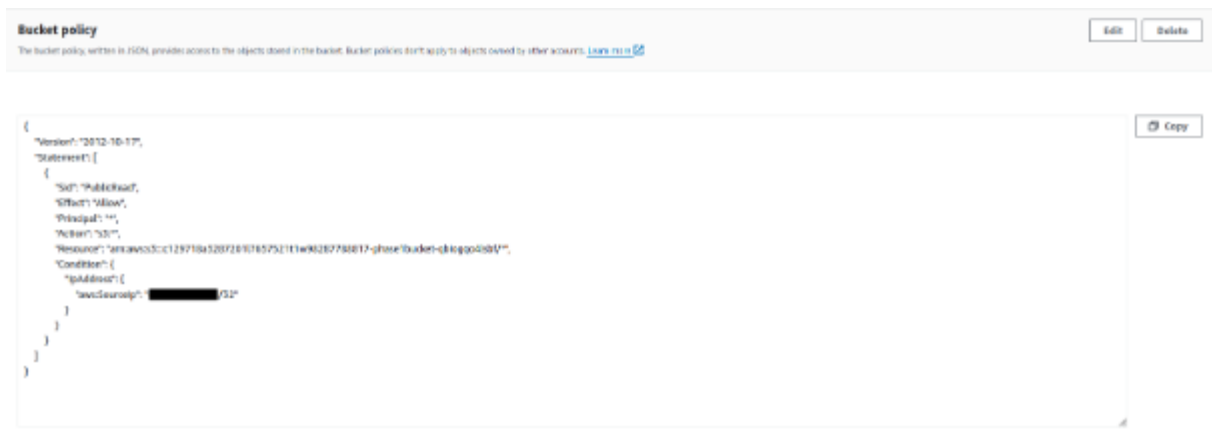


Рисунок 1.9 – Налаштування політик S3 bucket

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).

```

2 Do not replace any other values except those mentioned in the lab guide you are following
3 */
4
5 //GLOBAL
6 CONFIG = {};
7
8 CONFIG.STATUS_STR_ARR = [
9   "Got original image",
10  "Saved original image to s3",
11  "App server received image url",
12  "App server got original image buffer",
13  "Processed image",
14  "Saved adjusted to s3",
15  "Complete"
16 ];
17
18 CONFIG.FULL_URL_STR = "http://18.234.117.11:8008/upload";
19 CONFIG.FULL_IMAGE_PATH_STR = "https://arn:aws:s3:::c129718a328720117657521t1w98287788817-phaselbucket-qbiogqo4lsbf.s3.amazonaws.com";
20 CONFIG.FULL_IMAGE_URL_STR = "http://18.234.117.11:8008/get-image-urls";
21

```

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).

```

6  CONFIG = {};
7
8  CONFIG.SECRET_STR = "secret_squirrel";
9
10 CONFIG.DYNAMO = {};
11 CONFIG.DYNAMO.API_VERSION_STR = "2012-08-10";
12 CONFIG.DYNAMO.REGION_STR = "us-east-1";
13
14 CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase1";
15
16 CONFIG.PROCESSING_URL_STR = "http://18.234.117.11:8009/process";
17
18 CONFIG.S3 = {};
19 CONFIG.S3.API_VERSION_STR = "2006-03-01";
20 CONFIG.S3.REGION_STR = "us-east-1";
21 CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a328720117657521t1w98287788817-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 = "c129718a328720117657521t1w98287788817-phase1bucket-qbiogqo4lsbf";

```

Рисунок 1.12 – Налаштування веб застосунку

Після всіх налаштувань виконано запуск веб-застосунку, 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).

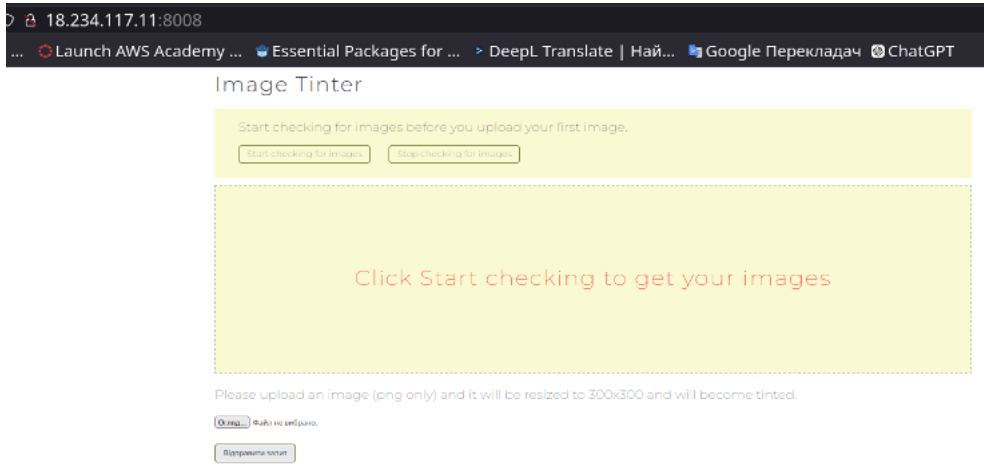


Рисунок 1.13 – Працюючий застосунок з тісним зв'язком

Для тестування програми, виконано команду «Start checking for images» та завантажено зображення. Під час обробки у фоновому режимі «Choose File», обирали 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).

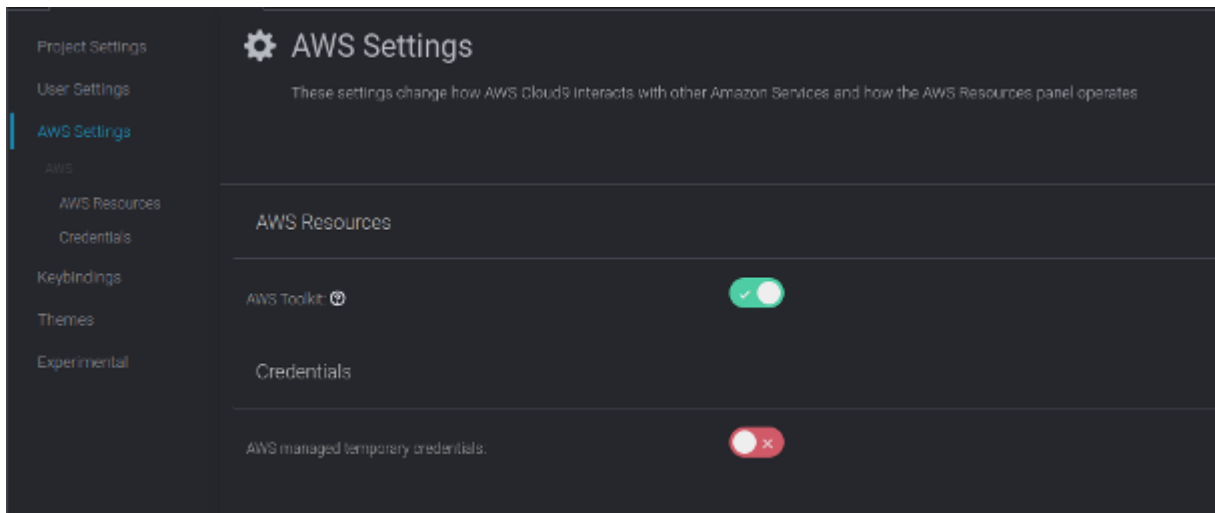


Рисунок 1.14 – Відключення управління тимчасовими обліковими даними.

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).

Amazon SQS > Queues > Create queue

Create queue

Details

Type
Choose the queue type for your application or cloud infrastructure.

Standard queue
At-least-once delivery, message ordering not guaranteed
+ At-least-once delivery
+ Best effort ordering

FIFO queue
First-in-first-out delivery, message ordering guaranteed
+ First-in-first-out delivery
+ Exactly-once processing

You can't change the queue type after you create a queue.

Name
ImageApp

A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

Configuration [info](#)
Set the maximum message size, visibility to other consumers, and message retention.

Visibility timeout [info](#)
30 Seconds
Should be between 0 seconds and 12 hours.

Message retention period [info](#)
4 Days
Should be between 1 minute and 14 days.

Delivery delay [info](#)
0 Seconds
Should be between 0 seconds and 15 minutes.

Maximum message size [info](#)
256 KB
Should be between 1 KB and 256 KB.

Receive message wait time [info](#)
0 Seconds

Рисунок 1.15– Створення нової черги в AWS SQS

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

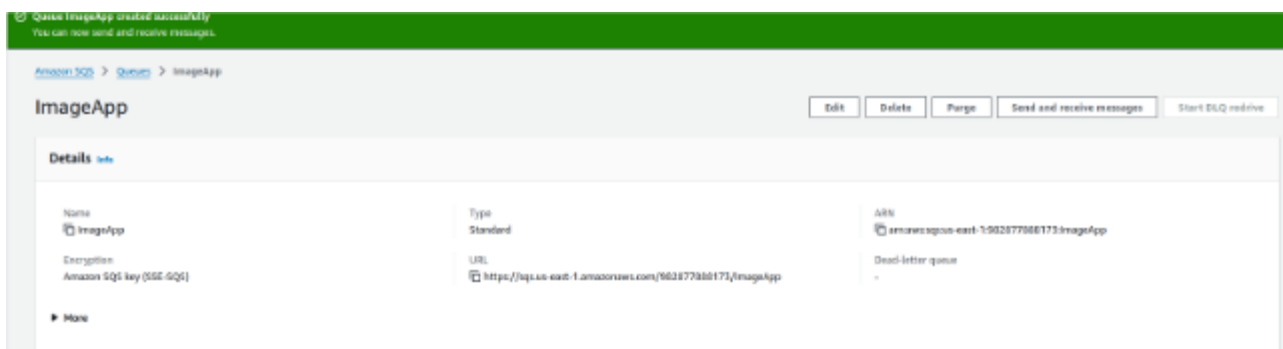


Рисунок 1.16 – Новостворена черга

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).

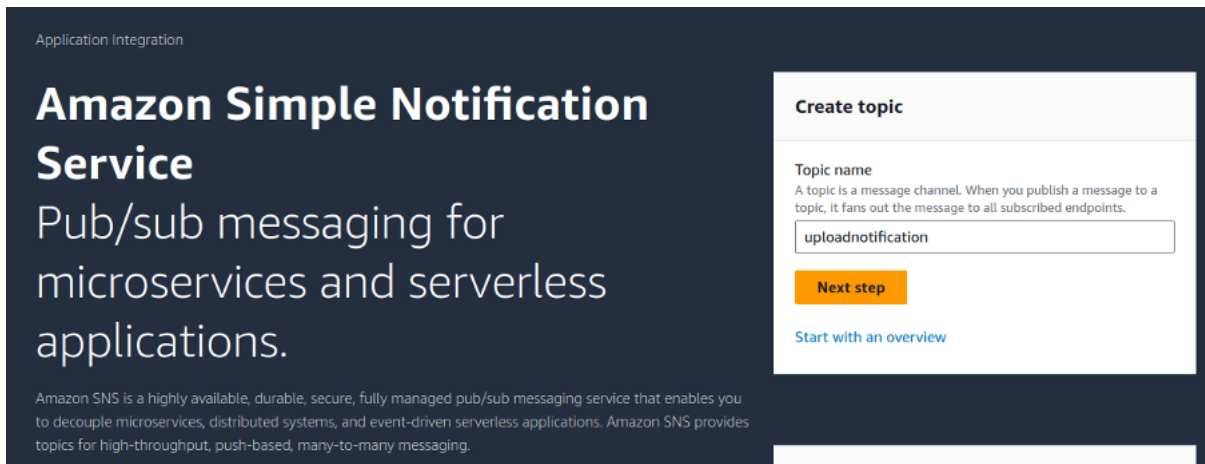


Рисунок 1.17 – Начальне вікно для створення нового «topic»

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:*:*:c129718a3287201176575211w98287788817-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).

General configuration

Event name

 Event name can contain up to 255 characters.

Prefix - optional
 Limit the notifications to objects with key starting with specified characters.

Suffix - optional
 Limit the notifications to objects with key ending with specified characters.

Event types

Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

Object creation

All object create events
 s3:ObjectCreated:*

Put
 s3:ObjectCreated:Put

Post
 s3:ObjectCreated:Post

Copy
 s3:ObjectCreated:Copy

Multipart upload completed
 s3:ObjectCreated:CompleteMultipartUpload

Рисунок 1.20 – Створення повідомлення про події

Destination

i Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

Destination
 Choose a destination to publish the event. [Learn more](#)

Lambda function
 Run a Lambda function script based on S3 events.

SNS topic
 Fanout messages to systems for parallel processing or directly to people.

SQS queue
 Send notifications to an SQS queue to be read by a server.

Specify SNS topic

Choose from your SNS topics

Enter SNS topic ARN

SNS topic

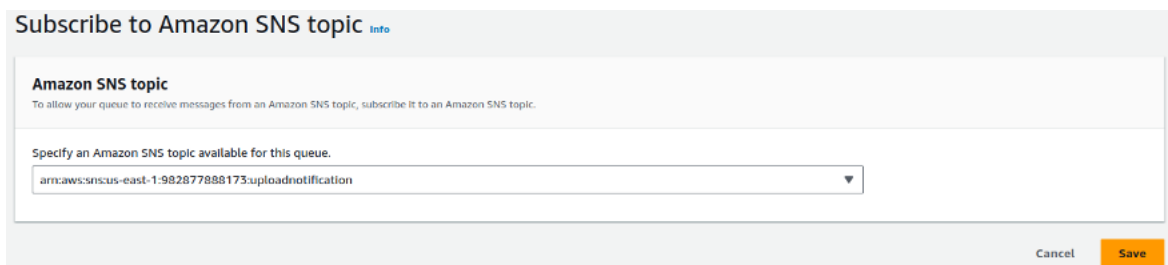
[Cancel](#) [Save changes](#)

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



Subscribe to Amazon SNS topic [Info](#)

Amazon SNS topic
To allow your queue to receive messages from an Amazon SNS topic, subscribe it to an Amazon SNS topic.

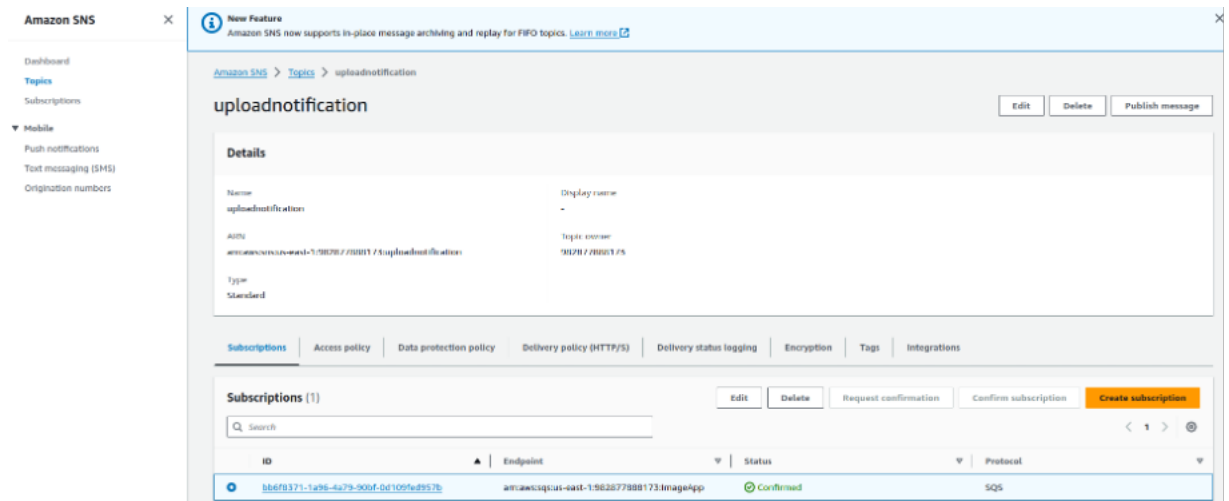
Specify an Amazon SNS topic available for this queue.

arn:aws:sns:us-east-1:982877888173:uploadnotification

Cancel Save

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).



Amazon SNS

Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)
Origination numbers

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Amazon SNS > Topics > uploadnotification

uploadnotification Edit Delete Publish message

Details

Name	uploadnotification	Display name	-
ARN	arn:aws:sns:us-east-1:982877888173:uploadnotification	Topic owner	982877888173
Type	Standard		

Subscriptions | Access policy | Data protection policy | Delivery policy (HTTP/S) | Delivery status logging | Encryption | Tags | Integrations

Subscriptions (1) Edit Delete Request confirmation Confirm subscription Create subscription

Search

ID	Endpoint	Status	Protocol
bb6f8371-1a96-4a79-900f-0c105fa89576	arn:aws:sqs:us-east-1:982877888173:imageapp	Confirmed	SQS

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).

Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN

Protocol
 The type of endpoint to subscribe

Endpoint
 An email address that can receive notifications from Amazon SNS.

After your subscription is created, you must confirm it. [Info](#)

► **Subscription filter policy - optional** [Info](#)
 This policy filters the messages that a subscriber receives.

► **Redrive policy (dead-letter queue) - optional** [Info](#)
 Send undeliverable messages to a dead-letter queue.

Cancel **Create subscription**

Рисунок 1.24 – Додавання пошти для повідомлень від SQS

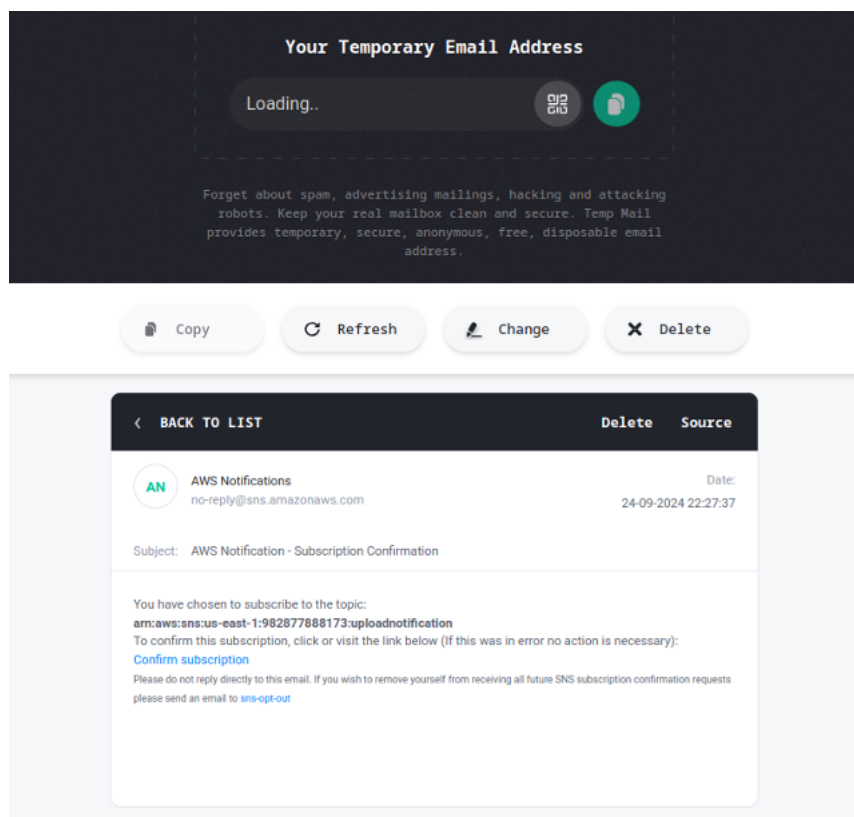


Рисунок 1.25 – Підтвердження на пошті підписки на SNS

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).

```

1 CONFIG = {};
2
3 //PHASE 2 CONFIG
4
5 CONFIG.FULL_IMAGE_URL_STR = "http://3.81.22.152:8007/get-image-urls";
6 CONFIG.FULL_IMAGE_PATH_STR = "https://c129718a328720117657521t1w98287788817-phase2bucket-pqmyuxkkwxkd.s3.amazonaws.com";
7 CONFIG.ARN_TRUNC_URL_STR = "c129718a328720117657521t1w98287788817-phase2bucket-pqmyuxkkwxkd";
8 CONFIG.WEB_SERVER_URL_STR = "http://3.81.22.152:8007";
9 CONFIG.APP_SERVER_URL_STR = "http://3.81.22.152:8010";
10
11 CONFIG.STATUS_STR_ARR = [
12   "Sent to S3",
13   "App server received image url",
14   "App server got original image buffer",
15   "Processed image",
16   "Saved adjusted to s3",
17   "Complete"
18 ];

```

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).

```

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
10 //we use the same DDB table for phase 2
11 CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase2";
12
13 CONFIG.S3 = {};
14 CONFIG.S3.API_VERSION_STR = "2006-03-01";
15 CONFIG.S3.REGION_STR = "us-east-1";
16 CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a328720117657521t1w98287788817-phase2bucket-pqmyuxkkwxkd";
17
18 // (Ignore the commented code below)
19 //CONFIG.DYNAMO.PROFILE_STR = "default";
20 // CONFIG.S3.PROFILE_STR = "default";

```

Рисунок 1.27 – Налаштування веб сервера

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

```

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
10 //we use the same DDB table for phase 2
11 CONFIG.DYNAMO.IMAGE_TABLE_NAME_STR = "ImagesTablePhase2";
12
13 CONFIG.S3 = {};
14 CONFIG.S3.API_VERSION_STR = "2006-03-01";
15 CONFIG.S3.REGION_STR = "us-east-1";
16 CONFIG.S3.IMAGE_BUCKET_NAME_STR = "c129718a328720117657521t1w98287788817-phase2bucket-pqmyuxkkwxkd";
17
18 // (Ignore the commented code below)
19 //CONFIG.DYNAMO.PROFILE_STR = "default";
20 // CONFIG.S3.PROFILE_STR = "default";

```

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 скопійована 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).

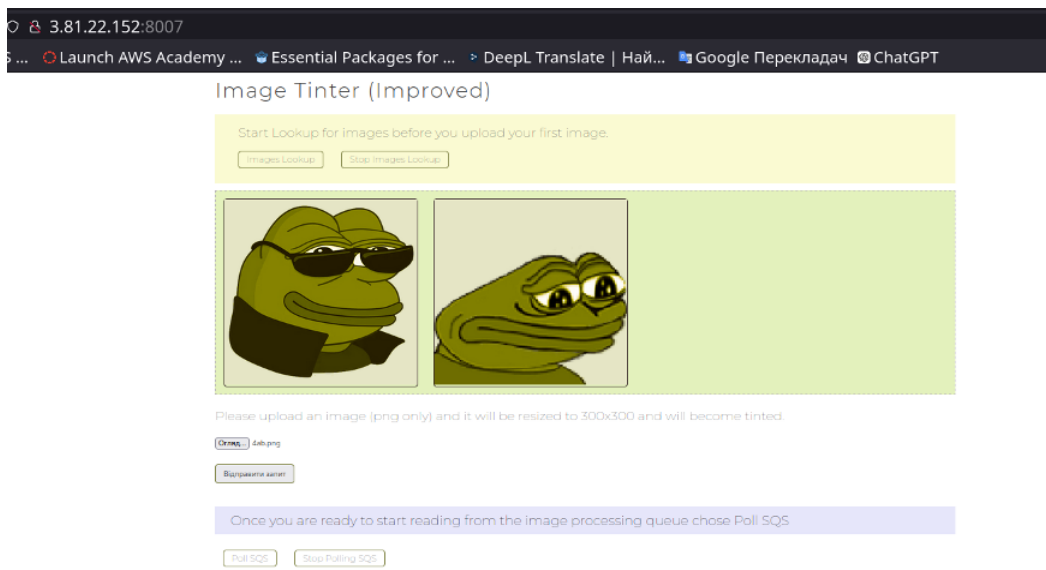


Рисунок 1.29 – Виведення застосунку при перевірці роботи

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

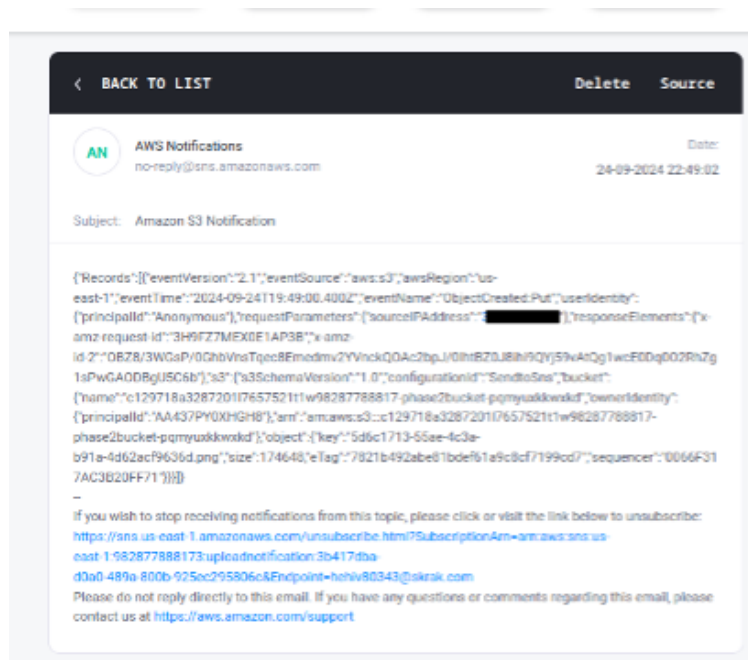


Рисунок 1.30 – Сповідження на пошту про дії користувача у застосунку

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 кожен відпрацьовану хвилину функції lambda).

2.1 Опис моделі

При розширенні бізнесу, 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.”

Basic information

Function name
Enter a name that describes the purpose of your function.
Load-Inventory
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).
 Create a new role with basic Lambda permissions
 Use an existing role
 Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
Lambda-Load-Inventory-Role
[View the Lambda-Load-Inventory-Role role](#) on the IAM console.

► **Additional Configurations**
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Cancel Create function

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)

Create bucket [Info](#)

Buckets are containers for data stored in S3.

General configuration

AWS Region
US East (N. Virginia) us-east-1

Bucket type [Info](#)

General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name [Info](#)
Inventory-1488

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Block public access to buckets and objects granted through new access control lists (ACLs)

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].

inventory-1488 > Create event notification

Lifecycle transition events
s3:LifecycleTransition

All lifecycle expiration events
s3:LifecycleExpiration*

Object expired
s3:LifecycleExpiration.Delete

Delete marker added by Lifecycle for a versioned object
s3:LifecycleExpiration.DeleteMarkerCreated

Intelligent-Tiering

Intelligent-Tiering archive events
s3:IntelligentTiering

Destination

[Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. \[Learn more\]\(#\)](#)

Destination
Choose a destination to publish the event. [Learn more](#)

Lambda function
Run a Lambda function script based on S3 events.

SNS topic
Fanout messages to systems for parallel processing or directly to people.

SQS queue
Send notifications to an SQS queue to be read by a server.

Specify Lambda function

Choose from your Lambda functions

Enter Lambda function ARN

Lambda function
Load-Inventory

[Cancel](#) [Save changes](#)

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).

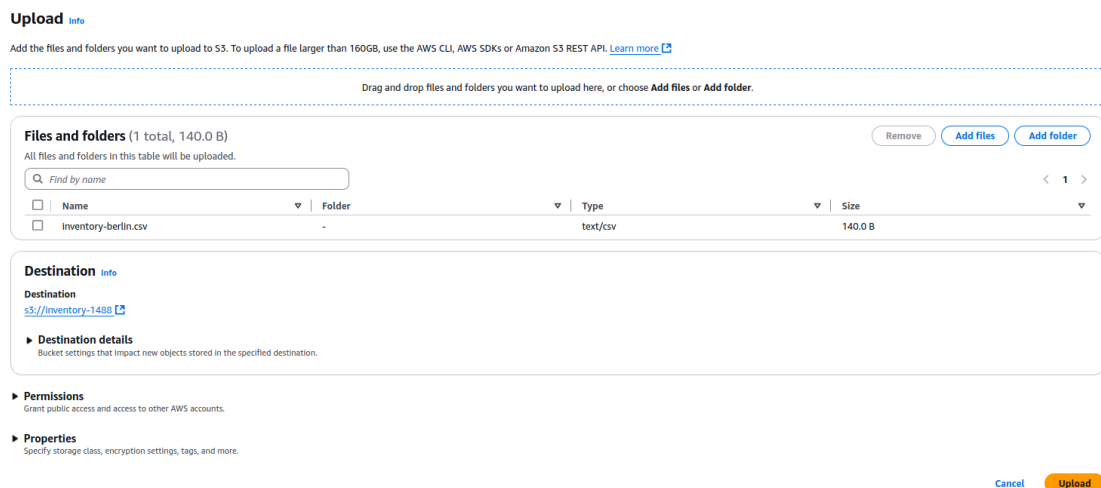


Рисунок 2.4 - Завантаження тестового звіту про інвентаризацію

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

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

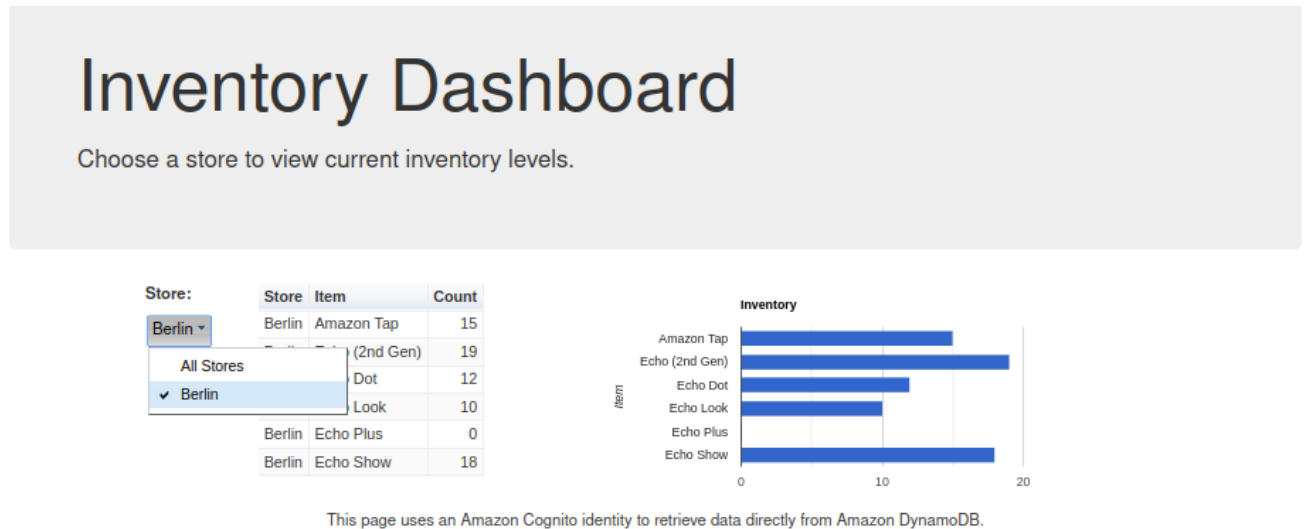


Рисунок 2.5 - Звіт про інвентаризацію в dashboard

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 вкладки 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 закінчаться 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).

Topics > Create topic

Create topic

Details

Type | Info
Topic type cannot be modified after topic is created

FIFO (first-in, first-out)

- Strictly-preserved message ordering
- Exactly-once message delivery
- Subscription protocols: SQS

Standard

- Best-effort message ordering
- At-least once message delivery
- Subscription protocols: SQS, Lambda, Data Firehose, HTTP, SMS, email, mobile application endpoints

Name

NoStock .fifo

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_). FIFO topic names must end with ".fifo".

Display name - optional | Info
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message.

My Topic

Maximum 100 characters.

Content-based message deduplication | Info
Enable default message deduplication based on message content. If unchecked, a deduplication ID must be provided for every publish request.

Turn on message deduplication

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).

New Feature
Amazon SNS now supports in-place message archiving and replay for FIFO topics. [Learn more](#)

Create subscription

Details

Topic ARN
arn:aws:sns:us-east-1:637423203485:NoStock

Protocol
The type of endpoint to subscribe
Email

Endpoint
An email address that can receive notifications from Amazon SNS.
wanak63710@datangel.com

After your subscription is created, you must confirm it. [Info](#)

Subscription filter policy - optional [Info](#)
This policy filters the messages that a subscriber receives.

Redrive policy (dead-letter queue) - optional [Info](#)
Send undeliverable messages to a dead-letter queue.

[Cancel](#) [Create subscription](#)

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 перенаправлені 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.

The screenshot shows the 'Create function' page in the AWS Lambda console. At the top, there are three options for creating a function: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below this is the 'Basic information' section with the following fields:

- Function name:** 'Check-Stock' (with a note: 'Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).')
- Runtime:** 'Python 3.8' (with a note: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.')
- Architecture:** 'x86_64' (with a note: 'Choose the instruction set architecture you want for your function code.')
- Permissions:** A section titled 'Change default execution role' with the following options:
 - Execution role:** 'Use an existing role' (selected), with a link to 'IAM console'.
 - Existing role:** 'Lambda-Check-Stock-Role' (selected), with a link to 'View the Lambda-Check-Stock-Role role on the IAM console'.

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.

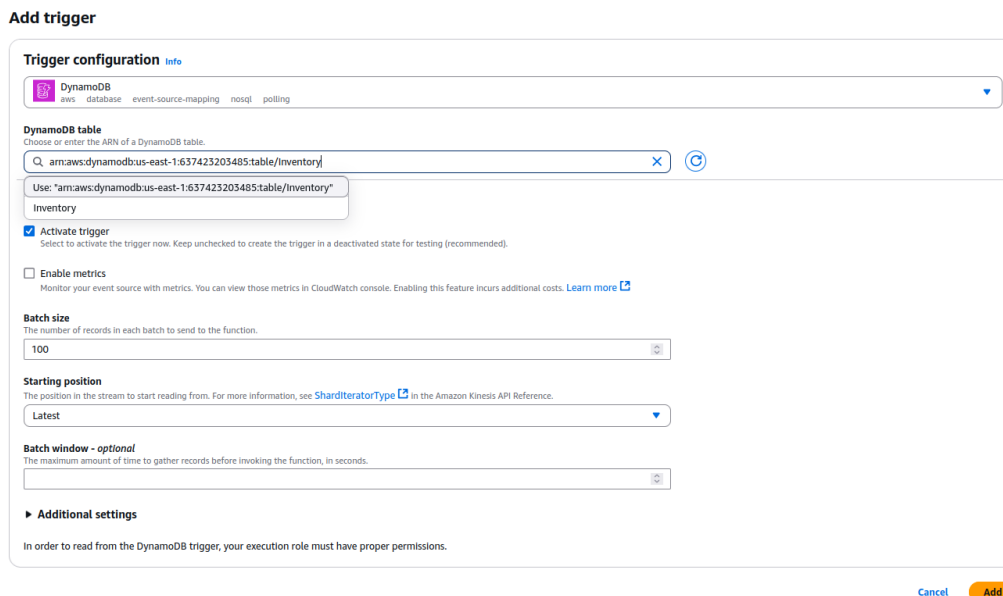


Рисунок 2.9 - Додавання триггеру події до lambda функції

Тобто налаштовані ці параметри:

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

Далі система (рис. 2.10.) була протестована.

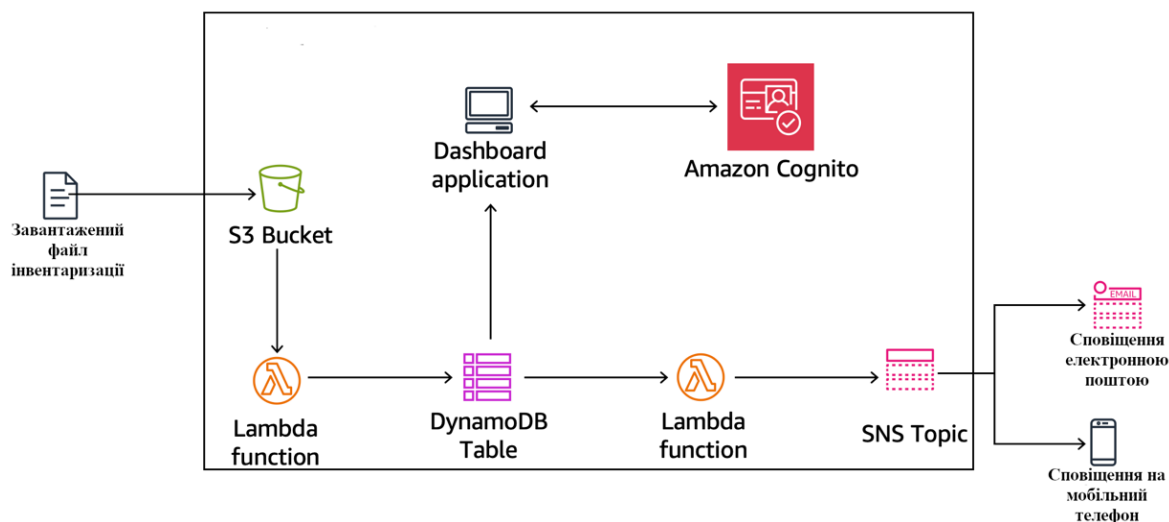


Рисунок 2.10 - Фінальна архітектура цього розділу

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 stop на примірнику веб-сервера, який надсилає повідомлення електронної пошти зі звітами про щоденні продажі. Однак завдання stop знижує продуктивність веб-сервера, оскільки воно потребує ресурсів.

Менеджер розповідає експерту AWS про роботу stop 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.

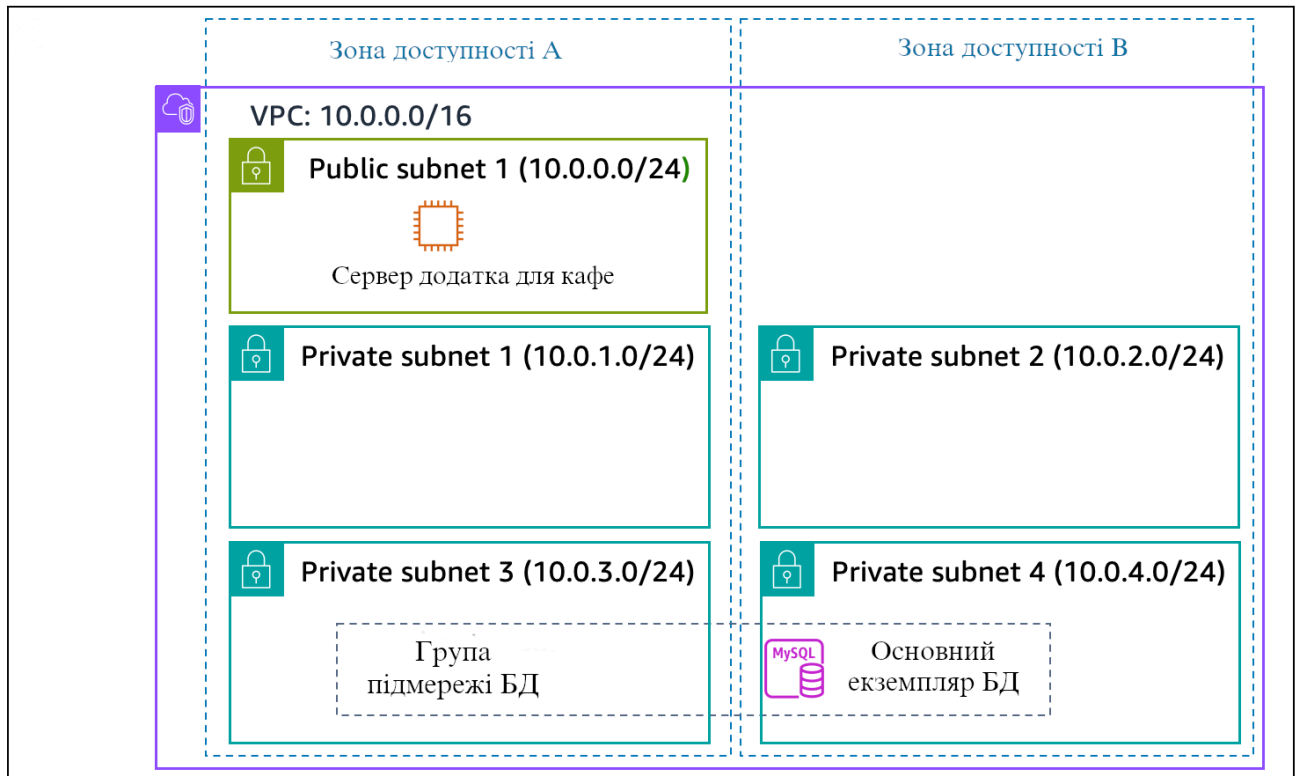


Рисунок 3.1 - Початкова архітектура проекту

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].

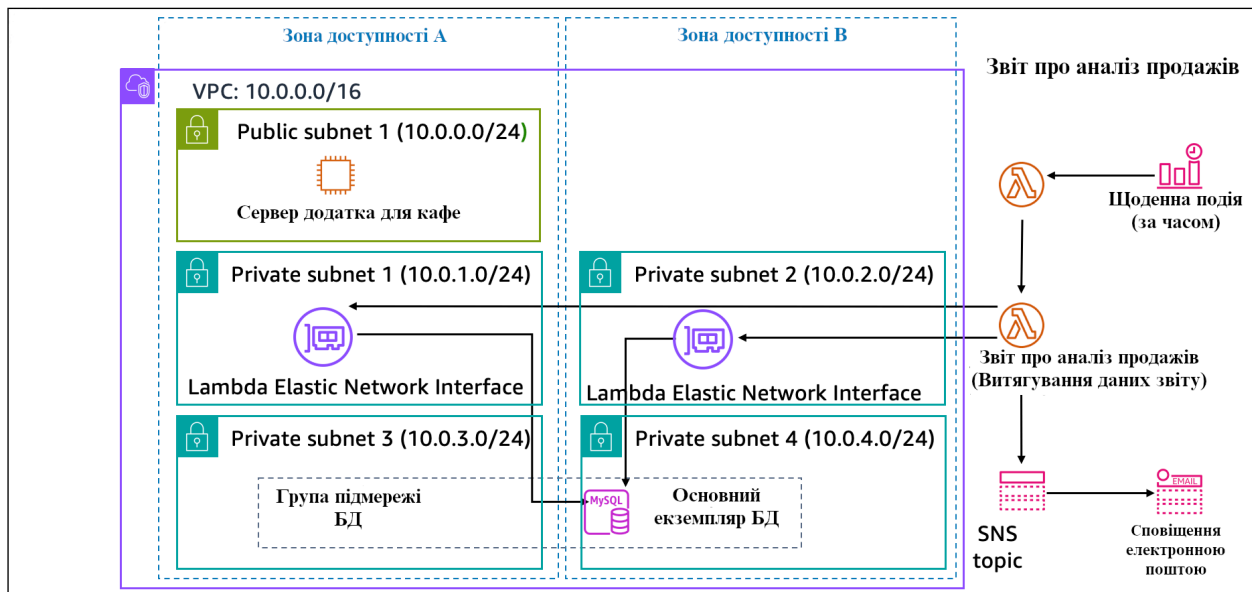


Рисунок 3.2 - Фінальна архітектура проекту

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

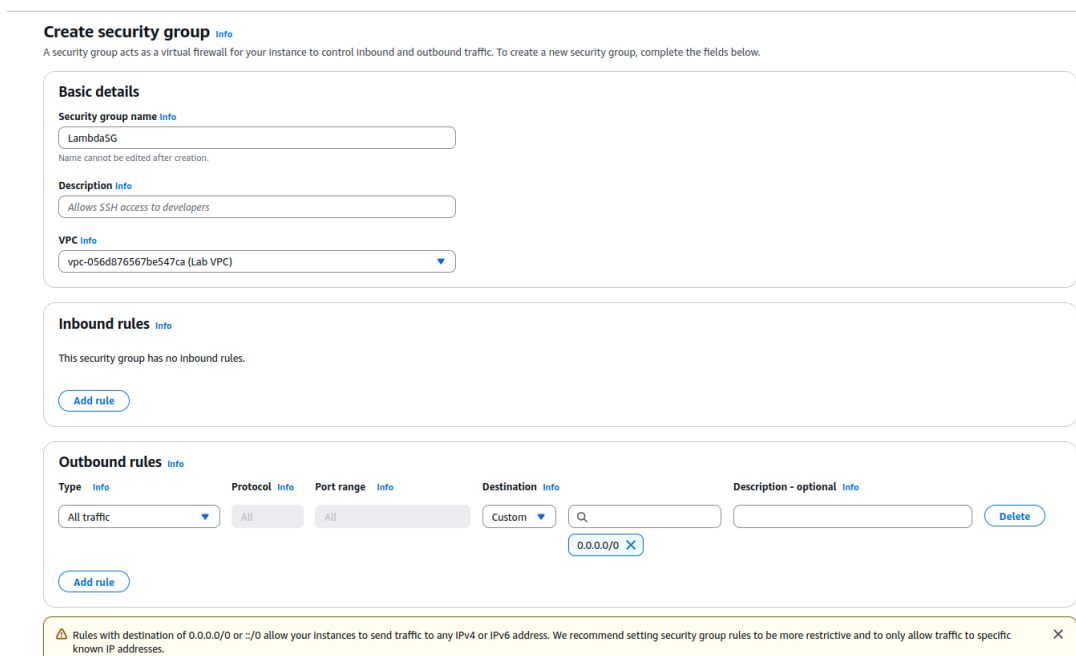


Рисунок 3.3 - Панель створення нової групи безпеки

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).

Edit inbound rules [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-05e07c0b595fd9132	MySQL/Aurora	TCP	3306	Custom	
-	MySQL/Aurora	TCP	3306	Custom	

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].

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch
 Start with a simple Hello World example.

Use a blueprint
 Build a Lambda application from sample code and configuration presets for common use cases.

Container image
 Select a container image to deploy for your function.

Basic information**Function name**

Enter a name that describes the purpose of your function.

salesAnalysisReportDataExtractor

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.11

Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

x86_64

arm64

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

Change default execution role**Execution role**

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role with basic Lambda permissions

Use an existing role

Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

salesAnalysisReportDERole

[View the salesAnalysisReportDERole role](#) on the IAM console.

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).

Edit basic settings

Basic settings [Info](#)

Description - optional

Lambda function to extract data from database

Memory [Info](#)

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Ephemeral storage [Info](#)

You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)

512 MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart [Info](#)

Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#). For Python and .NET runtimes, [view pricing](#)

None

Supported runtimes: .NET 8 (C#/.NET Core/PowerShell), Java 11, Java 17, Java 21, Python 3.12, Python 3.13.

Timeout

0 min 30 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#)

Use an existing role

Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

salesAnalysisReportDERole

[View the salesAnalysisReportDERole role](#) on the IAM console.

[Cancel](#) [Save](#)

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 А, завдяки додаванню обох використовуваних приватних підмереж.

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container Image
Select a container image to deploy for your function.

Basic information

Function name
 Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime [Info](#)
 Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
 [C](#)

Architecture [Info](#)
 Choose the instruction set architecture you want for your function code.
 x86_64
 arm64

Permissions [Info](#)
 By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role
 Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role with basic Lambda permissions
 Use an existing role
 Create a new role from AWS policy templates

Existing role
 Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
 [C](#)
[View the salesAnalysisReportRole role](#) on the IAM console.

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 функцій, 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)

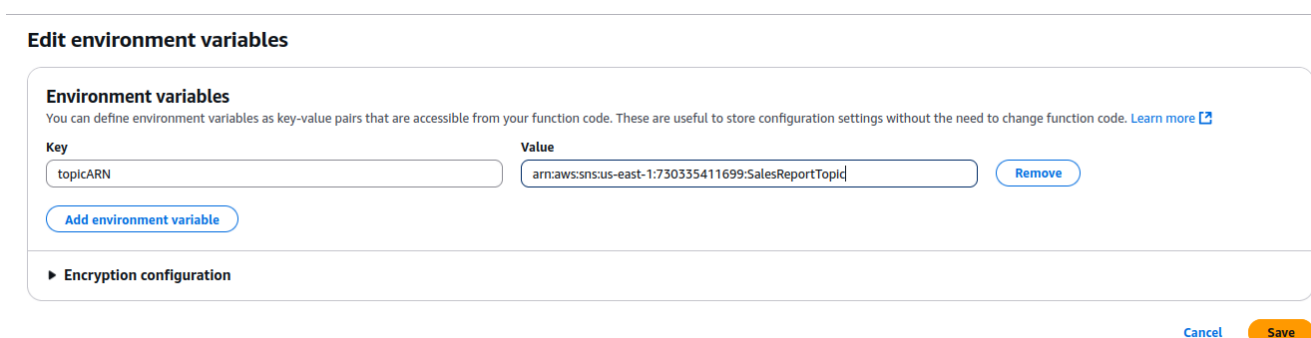
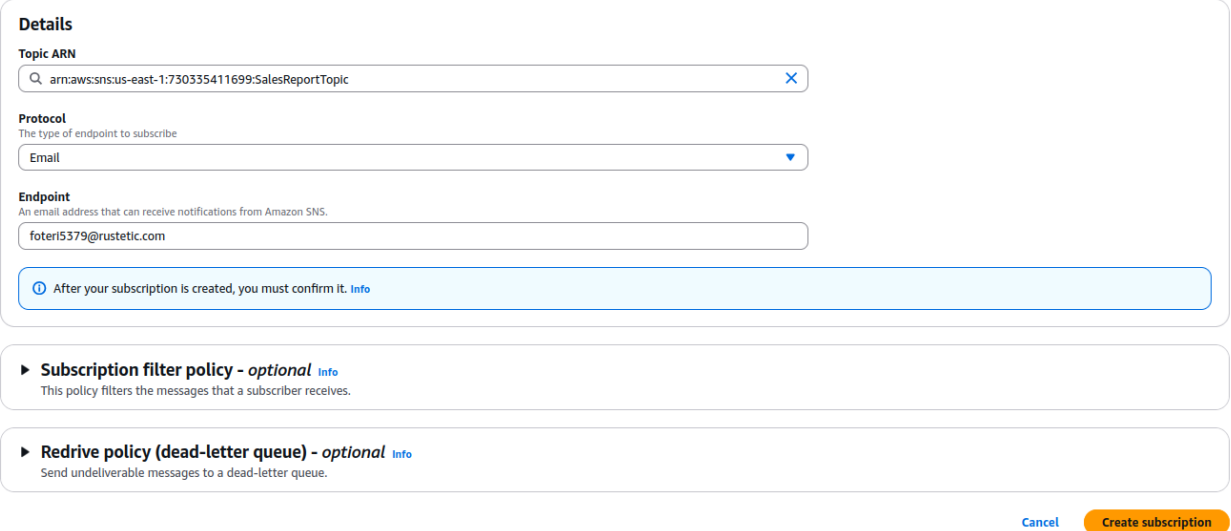


Рисунок 3.8 - Прописування змінних середовища

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).



The screenshot shows the 'Create subscription' form in the AWS console. It is titled 'Create subscription' and contains the following fields and sections:

- Details**
 - Topic ARN**: A text input field containing 'arn:aws:sns:us-east-1:730335411699:SalesReportTopic' with a search icon on the left and a close icon on the right.
 - Protocol**: A dropdown menu with 'Email' selected. Below it is the text 'The type of endpoint to subscribe'.
 - Endpoint**: A text input field containing 'foteri5379@rustetic.com'. Below it is the text 'An email address that can receive notifications from Amazon SNS.'
- A light blue informational box with a circular icon containing an 'i' and the text: 'After your subscription is created, you must confirm it. [Info](#)'
- Subscription filter policy - optional [Info](#)**: A section with a right-pointing triangle icon. Below the title is the text: 'This policy filters the messages that a subscriber receives.'
- Redrive policy (dead-letter queue) - optional [Info](#)**: A section with a right-pointing triangle icon. Below the title is the text: 'Send undeliverable messages to a dead-letter queue.'
- At the bottom right, there are two buttons: a blue 'Cancel' button and an orange 'Create subscription' button.

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).

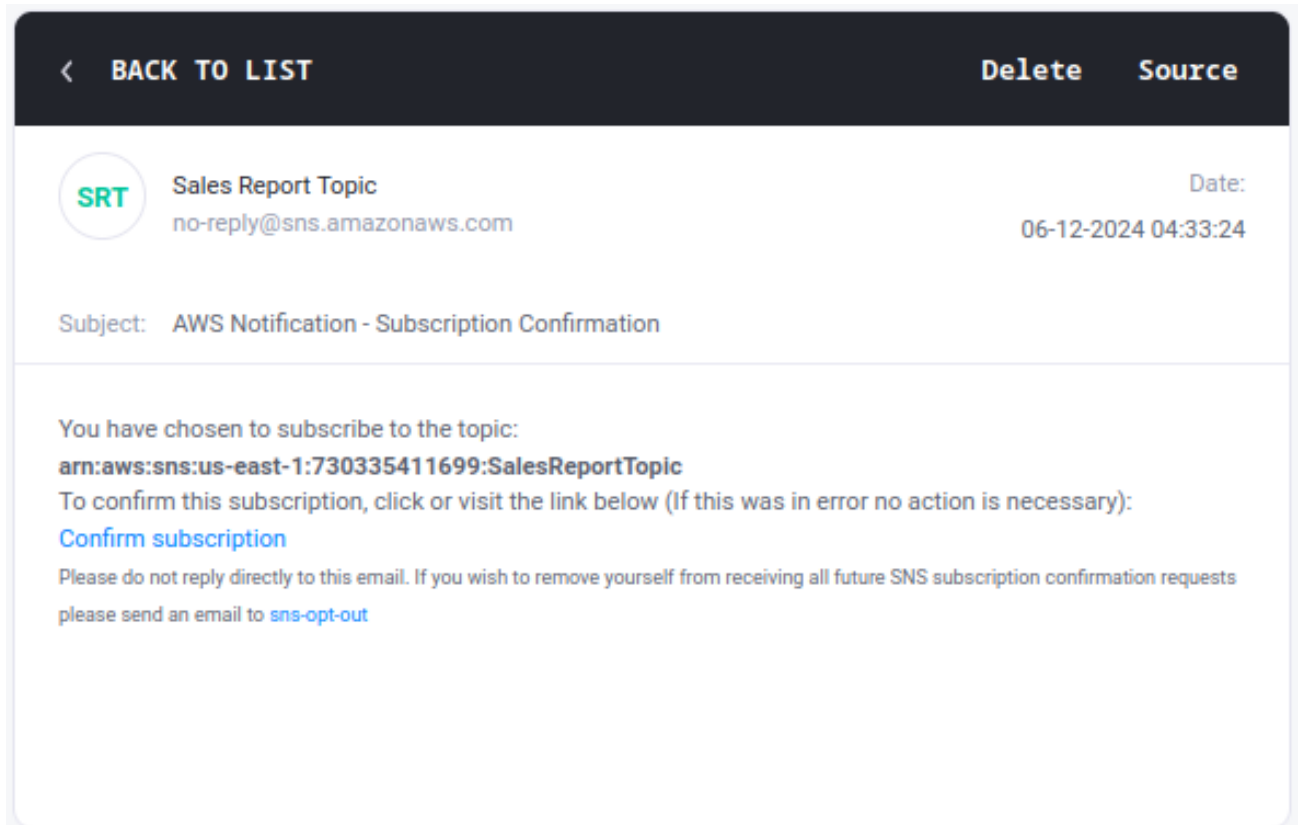


Рисунок 3.10 - Лист для підтвердження підписки

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).

Code **Test** Monitor Configuration Aliases Versions

✔ Executing function: succeeded ([logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 200,
  "body": "\\Sale Analysis Report sent.\\}"
```

Summary

Code SHA-256 epsephewhbOomELNkh63xWQq1fvboil2m8UrB2n7kfs=	Execution time 15 seconds ago
Request ID 306ca332-9433-4b6f-82b2-158165a230bc	Function version \$LATEST
Init duration 259.63 ms	Duration 3171.20 ms
Billed duration 3172 ms	Resources configured 128 MB
Max memory used 83 MB	

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: 306ca332-9433-4b6f-82b2-158165a230bc Version: $LATEST
END RequestId: 306ca332-9433-4b6f-82b2-158165a230bc
REPORT RequestId: 306ca332-9433-4b6f-82b2-158165a230bc Duration: 3171.20 ms Billed Duration: 3172 ms Memory Size: 128 MB Max Memory Used: 83 MB Init Duration: 259.63 ms
```

Test event [Info](#) [CloudWatch Logs Live Tail](#) [Save](#) [Test](#)

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).

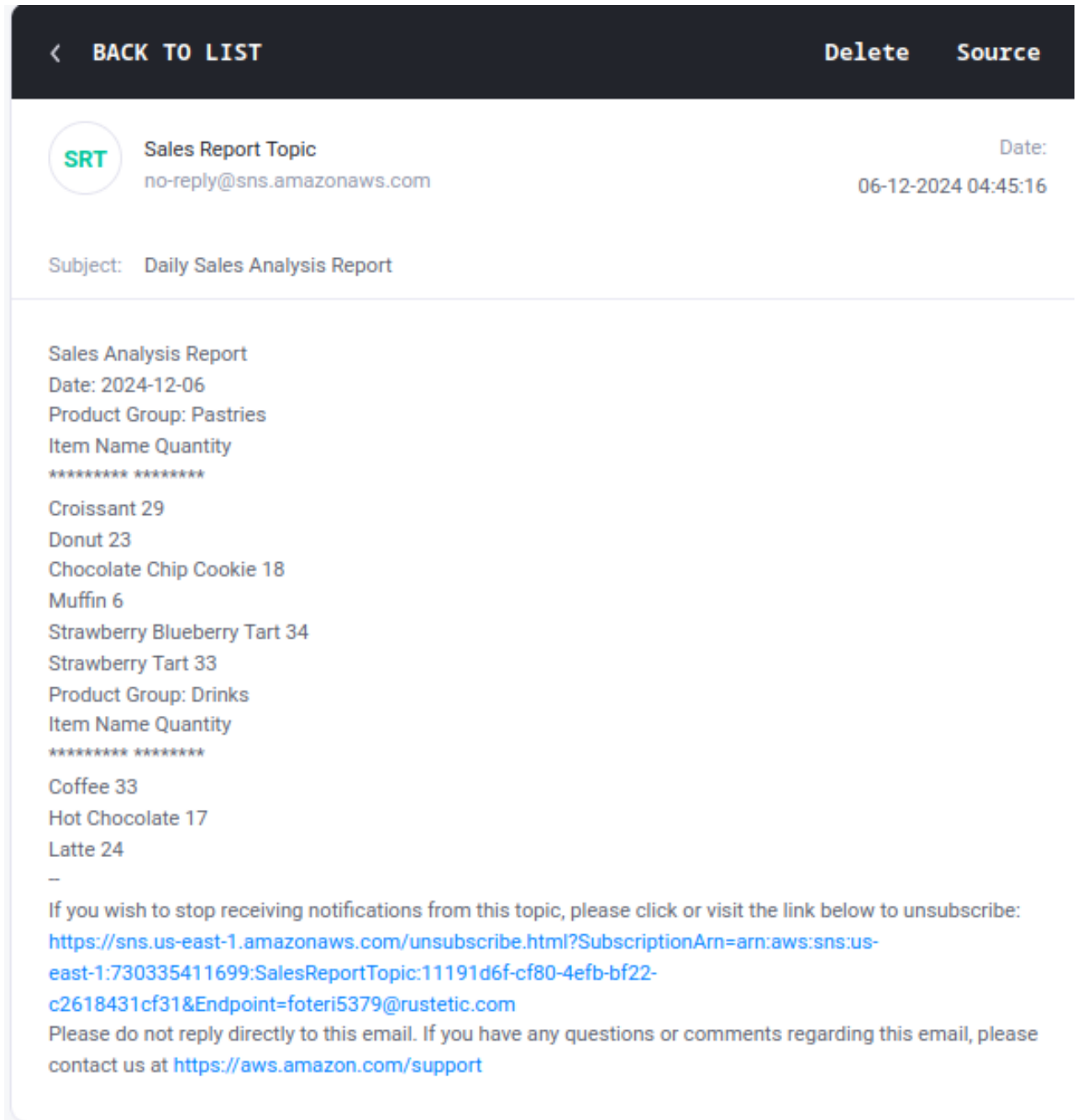


Рисунок 3.12 - Надісланий на пошту звіт

3.8 Щоденний звіт

Після підтвердження, що все працює як треба, було додано ще стон 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).

Add trigger

Trigger configuration Info

EventBridge (CloudWatch Events)
aws asynchronous schedule management-tools

Rule
Pick an existing rule, or create a new one.

Create a new rule
 Existing rules

Rule name
Enter a name to uniquely identify your rule.

salesAnalysisReportBridgeRule

Rule description
Provide an optional description for your rule.

sales Analysis Report Bridge Rule cron

Rule type
Trigger your target based on an event pattern, or based on an automated schedule.

Event pattern
 Schedule expression

Schedule expression
Self-trigger your target on an automated schedule using [Cron or rate expressions](#). Cron expressions are in UTC.

cron(0 15 ? * FRI *)
e.g. rate(1 day), cron(0 17 ? * MON-FRI *)

Lambda will add the necessary permissions for Amazon EventBridge (CloudWatch Events) to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

[Cancel](#) [Add](#)

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. [Электронный ресурс]. – Режим доступа: <https://aws.amazon.com/architecture/well-architected/>.
- [13] Amazon Web Services. Amazon S3 User Guide. [Электронный ресурс]. – Режим доступа: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>.
- [14] Amazon Web Services. Amazon SNS Developer Guide. [Электронный ресурс]. – Режим доступа: <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. [Электронный ресурс]. – Режим доступа:
<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.