МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
_____ *Ірина КРАВЧУК*
(підпис) (ім'я, ПРІЗВИЩЕ)
« *10* » *06* 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Забезпечення безперервної роботи веб-додатків за допомогою хмарних технологій*

Група: *3-011* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти _____ *Дмитро СИВАЧОК*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Ірина ВДОВИЧЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки _____ *Оксана ОСАДЧА*
(підпис) (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітньо-професійний ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК*
(ім'я, ПРІЗВИЩЕ)
« *01* » *03* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

Дмитру СИВАЧКУ
(прізвище, ім'я, по батькові)

1.Тема роботи *Забезпечення безперервної роботи веб-додатків за допомогою хмарних технологій*

Керівник роботи *ВДОВИЧЕНКО Ірина Никіфорівна, ктн.,доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з 01.03.2025 по *10.06.2025*

3. Вихідні дані до роботи *1. Тип мережі – захищена; Сегмент бак - наземний 2. Тип випромінювання – радіовипромінювання; 3. Клас комп. мережі –*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1 Системний аналіз захищених від радіовипромінювання комп'ютерних мереж;

2 Оцінка базових топологій;

3 Розробка інструментарію дослідження;

4 Створення інтерфейсу користувача;

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітньо-професійний ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК*
(ім'я, ПРІЗВИЩЕ)
« *01* » *03* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

Дмитру СИВАЧКУ
(прізвище, ім'я, по батькові)

1.Тема роботи *Забезпечення безперервної роботи веб-додатків за допомогою хмарних технологій*

Керівник роботи *ВДОВИЧЕНКО Ірина Никіфорівна, ктн.,доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з 01.03.2025 по *10.06.2025*

3. Вихідні дані до роботи *1. Тип мережі – захищена; Сегмент бак - наземний 2. Тип випромінювання – радіовипромінювання; 3. Клас комп. мережі –*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1 Системний аналіз захищених від радіовипромінювання комп'ютерних мереж;

2 Оцінка базових топологій;

3 Розробка інструментарію дослідження;

4 Створення інтерфейсу користувача;

StrikePlagiarism.com

Дата звіту 6/4/2025

Дата завантаження ---

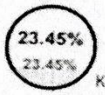
Звіт подібності

метадані

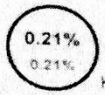
Назва організації
Ukrainian national aviation university
Заголовок
Сивачек_3-011_на проверку
Автор Науковий керівник / Експерт
СивачекКравчук І.
підрозділ
Криворізький Фаховий коледж

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	6323	48727
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість слів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		2
Інтервали		0
Мікропробіли		1
Білі знаки		0
Парафрази (SmartMarks)	a	196

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела з різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Забезпечення безперервної роботи веб-додатків за допомогою хмарних технологій ”: 60 с., 12 рис., 2 табл., 22 літературне джерело, 2 додатки.

SOC, МОНІТОРИНГ, РЕАГУВАННЯ, ІНЦИДЕНТ БЕЗПЕКИ, SIEM, SOAR, ХМАРНИЙ НАТИВНИЙ, КОНТЕЙНЕР, POD, ВУЗОЛ, КЛАСТЕР.

Кваліфікаційна робота присвячена актуальній проблемі оптимізації моніторингу безпеки в Cloud-Native середовищах за допомогою ефективного використання безкоштовних рішень з відкритим вихідним кодом. У сучасному IT-ландшафті, що характеризується стрімким розвитком хмарних технологій, мікросервісної архітектури та контейнеризації, традиційні методи моніторингу безпеки виявляються недостатніми, що створює "сліпі зони" та збільшує ризики кіберінцидентів. Ця проблема є особливо гострою для організацій з обмеженими бюджетами на кібербезпеку.

Метою роботи є дослідження та розробка підходів до оптимізації моніторингу безпеки в Cloud-Native середовищах шляхом ефективного використання безкоштовних рішень з відкритим вихідним кодом.

У роботі проведено аналіз сучасних Cloud-Native архітектур, їхніх переваг та специфічних загроз безпеці, що виникають у таких динамічних середовищах. Детально розглянуто концепції SOC (Security Operations Center), SIEM (Security Information and Event Management) та SOAR (Security Orchestration, Automation and Response), а також їхню роль у централізованому зборі, аналізі подій безпеки та автоматизації реагування. Висвітлено концепцію хмарно-орієнтованої безпеки "4C" (Code, Container, Cluster, Cloud) як основу для системного підходу до захисту.

Було здійснено вибір та аналіз безкоштовних та відкритих рішень для побудови підсистеми моніторингу подій безпеки. Серед них виділено інструменти, такі як ELK Stack (Elasticsearch, Logstash, Kibana) для SIEM функціоналу, TheHive та Cortex для SOAR, Grafana для візуалізації та Falco для моніторингу контейнерів на рівні ядра. Обґрунтовано їхній вибір на основі

відповідності вимогам хмарної безпеки, функціональності та можливостей інтеграції.

У практичній частині роботи створено віртуальне мережеве середовище, що імітує реальну Cloud-Native інфраструктуру (наприклад, з використанням GNS3 або віртуальних машин з Kubernetes/Docker). Проведено інтеграцію обраних безкоштовних інструментів SIEM та SOAR для централізованого збору та кореляції журналів подій. Змодельовано типові зловмисні дії та атаки, характерні для хмарних середовищ, і проведено тестування їх виявлення за допомогою налаштованих правил безпеки. Результати тестування підтвердили здатність розробленої підсистеми ефективно виявляти та обробляти інциденти безпеки, тим самим покращуючи стратегії реагування.

ЗМІСТ

ВСТУП

Сучасні веб-додатки мають складну багатокомпонентну архітектуру, де кожен компонент відповідає за певну функцію додатку, таку як: користувацький інтерфейс, база даних, частина бізнес-логіки тощо. Ці компоненти можуть розташовуватися на розподілених мережевих ресурсах та взаємодіяти один з одним через мережу. Такий підхід має багато переваг, таких як масштабованість, оптимальний розподіл навантаження, відмовостійкість, а також додавання нових функцій до додатку або зміна існуючих функцій.

Доступність застосунків – здатність безперервно виконувати очікувані функції суттєво впливає на взаємодію з користувачем. Методи покращення доступності застосунків включають реплікацію (резервування) компонентів, балансування навантаження обчислювальних ресурсів тощо. Для цього необхідно забезпечити надійність, стабільність та безпеку фізичної інфраструктури. Якщо ви хочете побудувати спеціалізовану апаратну платформу для розгортання застосунків, ви повинні інвестувати багато матеріальних та людських ресурсів. Побудова такої інфраструктури також

потребує часу, і її важко розширити, оскільки виділені ресурси потрібно пропорційно збільшувати.

Однак, існує ще одна альтернатива виділеній інфраструктурі – хмарні платформи. Хмарні постачальники зменшують клопоти з використанням фізичної інфраструктури та надають її у зручному для користувача вигляді. Найважливіші переваги хмари:

- Ви платите лише за використані ресурси (оплата за використанням).

- Швидке розгортання та можливість гнучкого та швидкого масштабування, збільшуючи або зменшуючи обсяг задіяних ресурсів залежно від фактичного навантаження.
- Хмарна платформа надає готові функціональні компоненти, які можна використовувати для створення додатків:

  DNS-сервіси, сховище даних, віртуальні машини тощо.
- Хмарні сервіси, що використовуються для розгортання додатків, мають дуже високий рівень надійності, якого самостійно важко досягти. Таким чином, розробники мають можливість абстрагуватися від питань підтримки власної інфраструктури та більше зосередитися на розробці функцій продукту.

Сьогодні утримувати власний центр обробки даних складніше та небезпечніше через руйнування інфраструктури та перебої з електропостачанням, спричинені повномасштабною війною. Тому перехід на хмарну платформу став важливішим. Наприклад, усі критично важливі національні реєстри були перенесені на хмарну платформу AWS після початку повномасштабного вторгнення.

Це дослідження має на меті оцінити можливості хмарних платформ та їх застосування у створенні новинних веб-сайтів. З огляду на внутрішні та міжнародні події, роль новинних ресурсів зростає. Ці ресурси містять велику кількість мультимедійного контенту та мають глобальну аудиторію, тому швидкий та надійний доступ до новин є критично важливим. Тому ресурси, що надають цей контент, повинні розташовуватися у фізичному місці ближче до користувачів.

Новинні ресурси характеризуються вибуховим (нерівномірним) зростанням активності користувачів. Це трапляється, наприклад, коли публікується «гаряча» новина. Щоб впоратися з таким навантаженням, потрібно вміти гнучко та автоматично керувати обсягом задіяних ресурсів. Такі вимоги до додатків ідеально узгоджуються з характеристиками хмарних платформ.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

У цьому розділі буде розглянуто та порівняно сервіси хмарних провайдерів для створення високодоступних мережевих додатків. Наразі лідерами на ринку хмарних технологій є Amazon Web Services (AWS), Google Cloud та Microsoft Azure. Популярні хмарні провайдери пропонують схожий спектр послуг. Тому потрібно розглядати лише одну платформу хмарних обчислень.

Для цієї роботи було обрано платформу AWS.

1.1 Обчислювальні послуги

AWS пропонує різноманітні варіанти обчислювальних послуг[1], які можна розділити на такі типи послуг:[3]

1. Інфраструктура як послуга (IaaS)
2. Контейнер як послуга (CaaS)
3. Функція як послуга (FaaS)

Інфраструктура як послуга

AWS дозволяє створювати, налаштовувати та повноцінно використовувати віртуальні машини під назвою Amazon Elastic Compute Cloud (EC2 [2]). Цей сервіс є типом сервісу IaaS. У цій моделі постачальник хмарних

послуг відповідає за роботу фізичної інфраструктури та гіпервізора віртуальної машини, тоді як користувач відповідає за керування всім іншим. Постачальник пропонує кілька способів оплати за окремі інсталяції EC2. Деякі з них:

- На вимогу – плата стягується погодинно або посекундно, залежно від типу віртуальної машини.
- Зарезервовано – Орендуйте EC2 на один або три роки. Заощадьте до 72% порівняно з орендою на вимогу.
- Спот – Використовуйте безкоштовні віртуальні машини в інфраструктурі AWS. Це може заощадити до 90% витрат, але коли попит EC2 зросте, екземпляр може бути зупинений.

Під час використання IaaS, відмовостійкість, яка забезпечує високу доступність, вимагає більше ресурсів для налаштування, ніж інші типи сервісів, і сильно залежить від дизайну застосунку. Платформа надає сервіси, що забезпечують високу доступність. За допомогою сервісу автоматичного масштабування можна налаштувати автоматичне масштабування сервісів. Цей сервіс швидко запускає та зупиняє екземпляри на основі поточного рівня навантаження та вибраних параметрів. У налаштуваннях сервісу можна вказати мінімальну кількість екземплярів EC2, які можуть працювати одночасно, тим самим підвищуючи доступність.

Може статися так, що на певний екземпляр потрапляє багато трафіку, що призводить до його перевантаження, тоді як інші екземпляри мають достатньо доступних ресурсів. Щоб уникнути цієї ситуації та рівномірно розподілити трафік між запущеними екземплярами EC2, можна скористатися сервісом Elastic Load Balancer.

Рекомендовано запускати екземпляри EC2 у різних зонах доступності. У межах географічного регіону інфраструктура постачальника складається з кількох ізольованих місць, які називаються зонами доступності. З різних

причин (наприклад, стихійних лих) зона доступності може бути недоступною. У цьому

випадку користувачі можуть використовувати її, якщо додаток розгорнуто в кількох зонах доступності.

Варто зазначити, що параметри масштабування EC2 можна налаштувати. Переваги EC2:

1. Можливість вибору типу віртуальної машини, який найкраще відповідає вашій програмі. Наприклад, D3 оптимізовано для програм, що потребують високої пропускної здатності диска.
2. Високорівневі елементи керування дозволяють повністю налаштувати середовище, в якому працює ваша програма.
3. Покращена безпека завдяки кращій ізоляції та можливості налаштування відповідно до власних вимог безпеки.

недолік:

1. Потрібно більше часу для налаштування та підтримки, ніж інші сервіси, що уповільнює розробку.
2. Для віртуалізації витрачаються надмірні обчислювальні ресурси. 3. Розгортання та оновлення програм стає складнішим

Іншим типом сервісу є контейнер як сервіс. Ці сервіси дозволяють запускати контейнери Docker, які містять код вашого застосунку та всі залежності, необхідні для його роботи. Це дозволяє запускати ваш застосунок у середовищі, яке підтримує контейнери, без необхідності налаштування середовища виконання.[4]

AWS надає різноманітні контейнерні сервіси[5]. Наприклад, з точки зору оркестрації контейнерів, існують Amazon Elastic Container Service (ECS) та Amazon Elastic Kubernetes Service (EKS). ECS більше інтегрований з хмарною платформою, тоді як EKS дозволяє користувачам використовувати популярну систему оркестрації контейнерів Kubernetes.

Ці рішення можуть використовувати EC2 та Fargate для запуску

контейнерів. Fargate — це безсерверне рішення. Це означає, що немає потреби

налаштовувати та конфігурувати сервери, такі як EC2, для запуску контейнерів. Це дозволяє користувачам зосередити більше ресурсів на розробці додатків. ECS та EKS надають чудові функції для забезпечення високої доступності програм. Вони також є високодоступними за своєю природою, наприклад, вони можуть автоматично відновлювати контейнери, які перестали працювати через проблеми. Крім того, контейнери легші за віртуальні машини, тому, якщо вам потрібно запустити контейнер, він запуститься швидше.

Подібно до EC2, ECS та EKS інтегровані з Elastic Load Balancer та дозволяють підвищити доступність програм, розгортаючи їх у різних зонах доступності. Однак, порівняно з EC2, ці сервіси надають більш гнучкі та зручні інструменти конфігурації розширення. Це не тільки покращує використання ресурсів, але й підвищує відмовостійкість. Крім того, під час використання цих сервісів можна легко оновлювати програми або їх компоненти, не зупиняючи їх роботу.

Якщо вам не потрібна гнучкість системи оркестрації, ви можете використовувати Elastic Beanstalk та App Runner. Ці сервіси запускають контейнери з меншим контролем та налаштуванням.

Переваги контейнерів:

1. Використовуйте обчислювальні ресурси ефективніше, ніж віртуальні машини.
2. Легко перенести в інші середовища виконання. Наприклад, з традиційних центрів обробки даних у хмару.
3. Швидке та гнучке розширення

недолік:

1. Менший контроль, ніж у EC2.
2. Через погану ізоляцію безпека нижча.

FaaS, або безсерверні сервіси, дуже зручні, оскільки користувачам не потрібно турбуватися про налаштування, запуск або масштабування серверів.

Постачальник хмарних послуг виконує всі ці завдання за вас. Для безсерверних обчислень AWS пропонує сервіс Lambda. [6]

За допомогою Lambda ви просто завантажуєте свій код, і сервіс створює функцію Lambda. Ви вибираєте обсяг пам'яті та ресурсів процесора для функції. Потім ви приєднуєте до функції події, які її запускають. Ця подія може бути згенерована сервісом AWS, таким як шлюз, база даних або користувач. Ви платите лише за функцію, яка виконується, а вартість залежить від вибраної вами ємності та кількості разів виконання функції.

Сервіс забезпечує високу доступність для лямбда-функцій без необхідності додаткового налаштування, такого як IaaS та CaaS. Наприклад, лямбда-функції за замовчуванням використовують різні зони доступності.

Цей постачальник автоматично масштабуватиме лямбда-функції, але майте на увазі, що існує обмеження на кількість функцій, які можна виконувати одночасно. Це обмеження залежить від регіону та становить приблизно 1000 функцій, але ви можете попросити постачальника збільшити це обмеження за потреби. Якщо з якоїсь причини функцію неможливо повністю виконати, постачальник автоматично спробує повторно виконати її двічі.

Якщо лямбда-функція рідко використовується протягом певного періоду часу, вона може перестати працювати [7]. Щоб уникнути цього, можна зарезервувати потужність лямбда-функції, але це призведе до додаткової плати.

Лямбда не підходить для тривалого виконання, оскільки максимальний час виконання становить 15 хвилин.

Переваги лямбда-технології:

1. Платіть лише за виконаний код, що вигідніше, ніж в інших сервісах. (Оплата за використання)
2. Автоматичне масштабування.
3. Немає потреби витрачати ресурси на налаштування інфраструктури недолік:

1. Якщо функція використовується нечасто, може виникнути затримка у її виконанні (холодний старт).

2. Кількість підтримуваних мов програмування обмежена.
3. Розмір коду функції та виділена обчислювальна потужність обмежені.

Різні типи послуг мають різні характеристики (Таблиця 1). Наприклад, ви можете вибрати послугу на основі ступеня контролю над інфраструктурою (Рисунок 1.1). Залежно від ситуації, це може бути перевагою або недоліком.

Таблиця 1.1- Порівняння IaaS, CaaS та FaaS

Особливості	Інфраструктура як послуга (EC2)	Хмарний сервер (ECS)	Функція як послуга (FaaS) (Lambda)
Рівень контролю	високий	Часткове	Низький, контролюється постачальником
Оплата	Залежить від типу EC2 та середовище виконання	Подібно до IaaS або з вибраних віртуальний процесор, пам'ять та час виконання	Залежить від часу виконання та кількості функцій
Середовище виконання	Залежить від обраної операційної системи	Залежить від контейнера	З урахуванням обмежень на надання послуг
Час виконання	Безлімітний	Безлімітний	Максимум 15 хвилин
час початку	Відносно довго	Відносно швидко	Швидкий, але можливий холодний запуск
Збільшити	Налаштовується	Налаштовується	Працює як сервіс без будь-якої конфігурації
Доступність програми	Залежить від дизайну програми	Залежить від дизайну програми	Залежить від постачальника

Очікуйте вбудовані функції спеціальних можливостей	99,95%	99,9%	99,99%

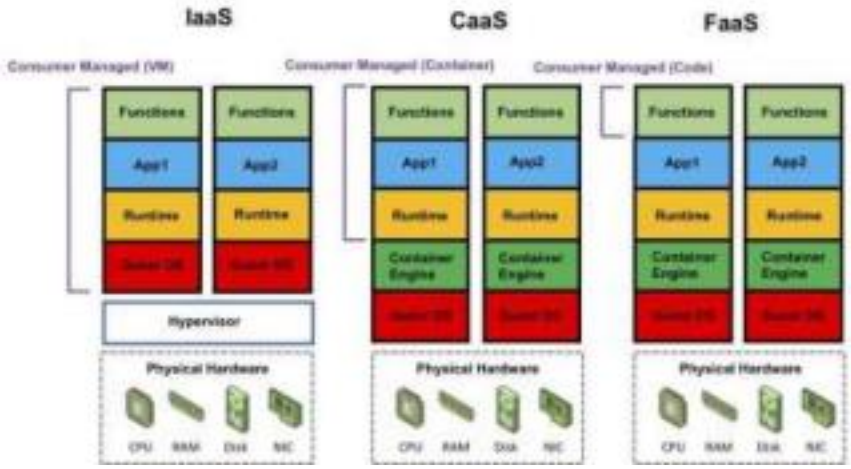


Рисунок 1.1 - Рівні контролю, що забезпечуються різними типами послуг[3]

Рішення, ймовірно, залежатиме від того, наскільки важливою є портативність застосунку та наскільки важливою є можливість запуску на інших платформах. У цьому випадку контейнери будуть найкращим варіантом, тоді як портування безсерверного застосунку буде складнішим. Наприклад, якщо навантаження невелике, і вашій програмі потрібно обробляти лише невелику кількість запитів, то вибір сервісу Lambda має сенс з точки зору бюджету. Однак, якщо уникнення холодного запуску є критично важливим, то це рішення може бути недоцільним.

Вибираючи сервіс, можна враховувати багато факторів. За необхідності та можливості можна використовувати різні технології для одного застосунку. Це дозволить використовувати сильні сторони сервісу та компенсувати його слабкі сторони. Наприклад, для постійних запитів можна використовувати віртуальні машини або контейнери; для періодичних функцій можна використовувати Lambda.

1.2 Візити на широкій географічній території

Для користувачів критично важливо мати високоякісний доступ до веб застосунків. Якщо користувачі розподілені по всьому світу, завдання забезпечення високоякісного доступу стає складнішим. Оскільки, якщо користувач знаходиться далеко від сервера, можуть виникати очевидні затримки під час використання застосунку. AWS, з його глобальною інфраструктурою та швидшою, ніж будь-коли раніше, швидкістю з'єднання, надає послуги, які можуть вирішити цю проблему.

Система доменних імен (DNS) зіставляє домен із певним IP-адреса сервера. AWS має високодоступну, масштабовану службу DNS під назвою Route 53 [9] (Рисунок 1.2).

Рисунок 1.2- Приклад захисту на маршрут 53

За допомогою цієї послуги ви можете реєструвати та керувати доменними іменами, а також переносити доменні імена від інших реєстраторів. Щоб покращити доступ до програм у різних регіонах, Route 53 дозволяє розподіляти трафік між ресурсами в різних регіонах. Водночас він також підтримує використання політик маршрутизації для розподілу трафіку на основі різних умов. Деякі політики маршрутизації наведено нижче: o Зважена маршрутизація – дозволяє розподіляти трафік між кількома ресурсами та налаштовувати обсяг трафіку, який отримуватиме кожен ресурс.

- Маршрутизація на резервний ресурс – дозволяє визначити основний ресурс, на який буде спрямовано весь трафік, та інші ресурси, які переймуть трафік, якщо основний ресурс перестане функціонувати.
- Геолокаційна маршрутизація – дозволяє орієнтуватися на ресурси

трафіку на основі географічного регіону, де знаходиться користувач. ○ Маршрутизація за географічною близькістю – також дозволяє розподіляти трафік на основі регіонів користувачів та серверів. Однак ця стратегія дозволяє визначити розмір самих регіонів, що може знадобитися для більш рівномірного розподілу трафіку між серверами.

○ Маршрутизація на основі затримки – розподіляє трафік між ресурсами на основі найменшої затримки. Маршрут 53 вимірює, до якого ресурсу користувач має найменшу затримку доступу, і перенаправляє його трафік на цей ресурс. Якщо метою є мінімізація затримки між вашою програмою та вашими користувачами, це може досягти найкращого розподілу трафіку між ресурсами в різних регіонах.

Route 53 також дозволяє використовувати перевірки справності для перевірки доступності ресурсів (таких як веб-сервери) та використовувати політики маршрутизації для обходу недоступного ресурсу, якщо він недоступний. Ви можете налаштувати сповіщення, коли ресурс перестає працювати. [10] Доступні типи перевірок справності:

1. Відстежує стан кінцевих точок, визначених користувачем. 2. Перевірки справності, які відстежують стан інших перевірок справності. Це корисно, наприклад, коли потрібно контролювати, чи доступна мінімальна кількість справних ресурсів.

3. Перевірки справності, що відстежують тривоги від служби моніторингу AWS CloudWatch.

Завдяки цій функції моніторингу справності ресурсів ви можете покращити відмовостійкість усієї програми.

Вартість користування трасою 53 становить:

- Кількість керованих зон, що використовуються для зберігання правил маршрутизації домену
- DNS-запити
- Річна плата за домен.

Мережа доставки контенту

AWS надає мережу доставки контенту (CDN) через CloudFront. Цей сервіс допомагає швидко розповсюджувати контент (наприклад, зображення, файли html, js, css) користувачам. [11]

По-перше, розробникам потрібен сервер, який CloudFront використовуватиме для отримання контенту. Наприклад, це може бути сховище S3 або HTTP-сервер, що працює на EC2. Під час налаштування CloudFront потрібно вказати джерело контенту.

Сервіс використовує центри обробки даних, розташовані по всьому світу, які в інфраструктурі AWS називаються периферійними сайтами. Коли користувач робить запит до програми, підключеної до CloudFront, запит перенаправляється на найближчий до нього периферійний сайт.

Рисунок 1.3- Діаграма розташування країв [12]

Якщо потрібний користувачеві контент вже існує, він буде доставлений користувачеві негайно. Якщо контенту не існує, CloudFront зробить запит до вказаного сервера, що швидше, ніж надсилання запиту безпосередньо до сервера, оскільки інфраструктура AWS з'єднана між собою через дуже швидку приватну мережу. Потім CloudFront зберігає контент на сервері на периферійному сайті та доставляє його користувачеві. Наступного разу, коли надійде запит на цей контент, він буде негайно надісланий з периферійного сайту.

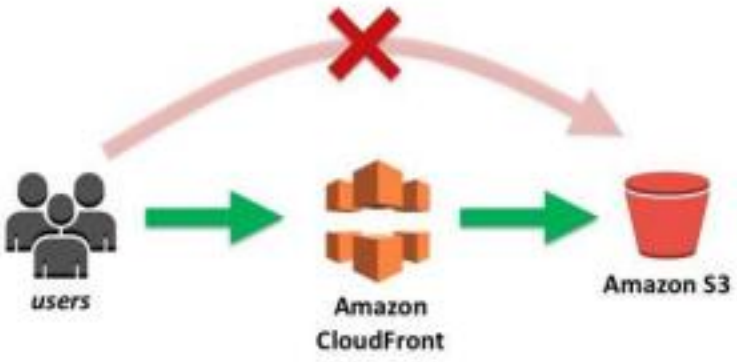


Рисунок 1.4 - Запит користувача до CloudFront

Завдяки цьому CloudFront може значно покращити доступ користувачів до даних програми. Сервіс має високу доступність, тому навіть якщо один граничний сайт вийде з ладу, інші сайти залишатимуться доступними. Кешування контенту дозволяє зменшити навантаження на сервер, і навіть якщо сервер недоступний, користувачі все ще можуть використовувати кешований контент. Це підвищує відмовостійкість самої програми.

Завдяки кешуванню та використанню периферійних сайтів, CloudFront може допомогти захиститися від DDoS-атак. За замовчуванням сервіс використовує AWS Shield Standard для захисту від DDoS-атак. За потреби ви можете підключитися до інших служб безпеки, що надаються AWS, для додаткового захисту. Наприклад, брандмауер веб-застосунків AWS (WAF) може

допомогти контролювати доступ до ваших застосунків та захистити від різних вразливостей. [13]

CloudFront також дозволяє використовувати Lambda@Edge. Це лямбда функції, які працюють на вашому edge-вебсайті. Їх можна використовувати для пришвидшення виконання запитів, реалізації додаткової безпеки, варіювання контенту залежно від місцезнаходження тощо.

CloudFront стягує плату лише за використані ресурси, що є моделлю оплати за використанням.

Глобальний прискорювач (також відомий як AWS Global Accelerator)

дозволяє пришвидшити з'єднання між програмами та користувачами за допомогою прискорювачів, що використовують глобальну мережеву інфраструктуру AWS. Він ідеально підходить для програм, які потребують з'єднань, відмінних від HTTP.[14]

Цей акселератор за замовчуванням надає 2 статичні адреси Anycast iPv4, які використовуються як точки підключення для додатків. Під час налаштування можна визначити кінцеві ресурси, на які буде перенаправлятися трафік після досягнення граничного сайту. Ці ресурси можуть бути розташовані в одному або кількох регіонах. Шляхи трафіку визначаються аналогічно CloudFront, де запити спочатку надсилаються до найближчого граничного сайту, а потім спрямовуються до кінцевих ресурсів через швидку мережу AWS.

Існує два типи акселераторів:

1. Стандартний прискорювач – спрямовує трафік до найкращої кінцевої точки

   Ресурси. Оптимальність визначається станом ресурсів, розташуванням користувачів та ваговими коефіцієнтами, призначеними ресурсам.

   Кінцевими ресурсами можуть бути:

   - EC2
   - Гнучка IP-адреса

   - Балансувальник мережевого навантаження (мережеве навантаження
     Балансувальник
   - Балансувальник навантаження застосунків

2. Налаштовувані прискорювачі – дозволяють направляти трафік від кількох користувачів до певного екземпляра EC2. Наприклад, ця функція може бути корисною, якщо у вас є ігровий додаток, якому потрібно підключати гравців до певного ігрового сервера. Для цього типу додатків для ресурсів кінцевих точок можна використовувати

лише підмережу Amazon Virtual Private Cloud (VPC), яка містить екземпляри EC2.

Сервіс використовує глобальну мережеву інфраструктуру постачальника для спрямування трафіку до запущених ресурсів, тим самим покращуючи доступність програм. Це досягається шляхом перевірки стану всіх зазначених ресурсів кінцевих точок.

Як і CloudFront, він за замовчуванням використовує AWS Shield для захисту від DDoS-атак і дозволяє додавати інші служби безпеки, такі як WAF.[15]

AWS Global Accelerator ідеально підходить для забезпечення високоякісного доступу у високоякісних географічних локаціях. Окрім зменшення затримки у швидкій мережі провайдера, цей сервіс допомагає оптимізувати розподіл ресурсів між регіонами. Ці переваги можуть значно пришвидшити запити додатків для користувачів у віддалених регіонах. AWS пропонує сторінку для тестування прискорення запитів до ресурсів в інших регіонах. Наприклад, на рисунку 1.5 файл розміром 100 КБ передається до регіону Північної Вірджинії (us-east-1) на 27% швидше, ніж за допомогою звичайного підключення до Інтернету.

Рисунок 1.5 - Глобальний тест швидкості акселератора

Вартість користування послугою залежить від кількості акселераторів (які мають фіксовану ціну) та обсягу трафіку, що проходить через них. Вибір між сервісами Route 53, CloudFront та Global Accelerator для забезпечення доступу в широкій географічній області залежить від програми та її функцій.

Кожен сервіс має свої переваги та недоліки (Таблиця 2). Route 53 дозволяє

керувати доменними іменами, вибирати необхідні політики маршрутизації та контролювати стан ресурсів. Це дозволяє автоматично перенаправляти трафік на найкращий кінцевий ресурс. Однак цей сервіс не дозволяє кешувати дані або використовувати глобальну мережеву інфраструктуру провайдера для покращення передачі даних між користувачами та програмами.

Таблиця 1.2-Порівняння Route 53, CloudFront та Global Accelerator

	Маршрут 53	CloudFront	Глобальний акселератор
Управління доменом	експонат	Загублений	Загублений
Маршрутизація трафіку до ресурсів у різних регіонах	Використання політик маршрутизації	Загублений	Використання акселератора
Кеш	Загублений	На межі	Загублений
Захист від DDoS-атак	Загублений	AWS Shield	AWS Shield

Прискорення передачі даних	Маршрутизація на основі затримки	Інфраструктура AWS та кешування	Інфраструктура AWS та оптимальна маршрутизація
Перевірити стан ресурсу	експонат	Загублений	експонат
Підтримка Lambda@Edge	Загублений	експонат	Загублений

CloudFront дозволяє тимчасово зберігати контент на периферії та використовувати переваги мережевої інфраструктури AWS, захисту від DDoS атак та інтеграції з іншими службами безпеки. Він також дозволяє використовувати Lambda@Edge для виконання коду безпосередньо на

периферії, що ще більше пришвидшує роботу ваших програм. Однак CloudFront не дозволяє керувати доменними іменами, маршрутизацією та балансуванням трафіку, як це робить Route 53, і не підходить для програм, які використовують протоколи, відмінні від HTTP.

Global Accelerator ідеально підходить для застосунків, що використовують протоколи, відмінні від HTTP. Він направляє трафік до ресурсів у різних регіонах та вибирає найкращий маршрут для трафіку, використовуючи інфраструктуру провайдера. Сервіс схожий на CloudFront, із захистом від DDoS

атак та інтеграцією з іншими службами безпеки. Однак він не кешує контент і не дозволяє виконання коду на периферійних сайтах. Він надає 2 статичні адреси, але не надає можливостей керування доменами, як Route.

Ви можете використовувати ці сервіси разом, щоб вони доповнювали один одного. Наприклад, ви можете використовувати Route 53 та CloudFront. Route 53 можна використовувати для керування доменом та оптимізованої маршрутизації, а CloudFront — для прискорення трафіку та кешування.

РОЗДІЛ 2
РОЗРОБКАСТРУКТУРИ ВЛАСНОГО РІШЕННЯ

У цьому розділі обговорюється, як розробити архітектуру новинного веб сайту, який можна розгорнути на AWS. Основою цієї архітектури є обчислювальна платформа та сервіси, що використовуються для забезпечення міжрегіонального доступу, проаналізовані в попередньому розділі.

2.1 Вибір обчислювальних платформ та послуг для забезпечення

міжрегіонального доступу

Щоб вибрати послугу, потрібно визначити основні критерії, яким має відповідати обране рішення.

Критерії вибору:

1. Характерною рисою новинних веб-сайтів є те, що після публікації новин, якими цікавиться велика кількість користувачів, трафік швидко та значно зростає. Тому для комфорту користувачів веб-сайт повинен легко справлятися з величезними навантаженнями та швидким зростанням.

2. Затримка веб-сайту для користувачів має бути мінімальною. 3. Він легкодоступний для новинних веб-сайтів з глобальною аудиторією.

4. Контент новинного сайту здебільшого статичний і добре кешується.

5. Оптимальне використання коштів на інфраструктуру та часу експертів для її налаштування та підтримки.

Route 53 та CloudFront ідеально підходять для ситуацій, коли користувачі географічно розподілені. Route 53 дозволяє керувати доменними іменами та використовувати політики маршрутизації для спрямування трафіку до найкращих обчислювальних кінцевих точок, зменшуючи затримку. CloudFront дозволяє кешувати контент (в першу чергу новини), що значно зменшує навантаження на ваші обчислювальні ресурси та пришвидшує доставку користувачам. Global Accelerator не підходить для новинного сайту, оскільки він не має можливостей кешування.

Вибираючи обчислювальну платформу, найкраще скористатися перевагами різноманітних сервісів.

Платформи без обмежень часу виконання мають свої переваги. Вони можуть обробляти велику кількість запитів користувачів без необхідності створювати нові екземпляри для кожного запиту. Це дозволяє їм відповідати на запити користувачів без затримок, спричинених «холодними стартами», як у Lambda.

Якщо кількість запитів велика і навантаження постійне, ці платформи

дешевші за Lambda. Якщо навантаження різко зростає, ці платформи можуть швидко масштабуватися. Тому вони ідеально підходять для обробки великих обсягів завдань, які виконуються дуже швидко.

Наприклад, для користувача новинного веб-сайту дуже важливо, щоб контент завантажувався швидко. Під час використання Lambda це завдання може мати затримки через холодний старт. Для цього типу завдань добре підходить платформа EC2 для новинного веб-сайту. Ви також можете використовувати інші платформи, такі як контейнерні платформи, але це може бути дорожче.

Лямбда-функції не підходять для цього завдання через холодні запуски та цінові фактори. Щоб уникнути холодних запусків, ви можете підтримувати функціональність лямбда-функції. Крім того, ви можете "розігрівати" функцію, періодично викликаючи її, щоб контейнер функцій не завершував роботу, тим

самим зменшуючи витрати. Ви також можете використовувати Lambda@Edge для пришвидшення. Але всі ці опції значно збільшать вартість платформи. Однак, платформа є гарним рішенням для завдань, які виконуються періодично та протягом короткого періоду часу, і для яких вимога "холодного запуску" не є високою. Наприклад, на новинному веб-сайті публікація нового контенту є таким завданням. Використання Lambda для виконання таких завдань може зменшити навантаження на платформу, яка працює безперервно. Оскільки функції Lambda викликаються рідше, вони також є найбільш економічно ефективними.

2.2 Архітектура застосунку

Маршрут 53 спрямовує запити користувачів до CloudFront. Ці два сервіси є глобальними, тому вони розміщені в різних регіонах. Усі інші компоненти розташовані в одному регіоні. Якщо потрібний користувачеві контент існує в кеші CloudFront, CloudFront надсилає відповідь назад користувачеві. В іншому випадку CloudFront спрямовує запит до відповідного компонента.

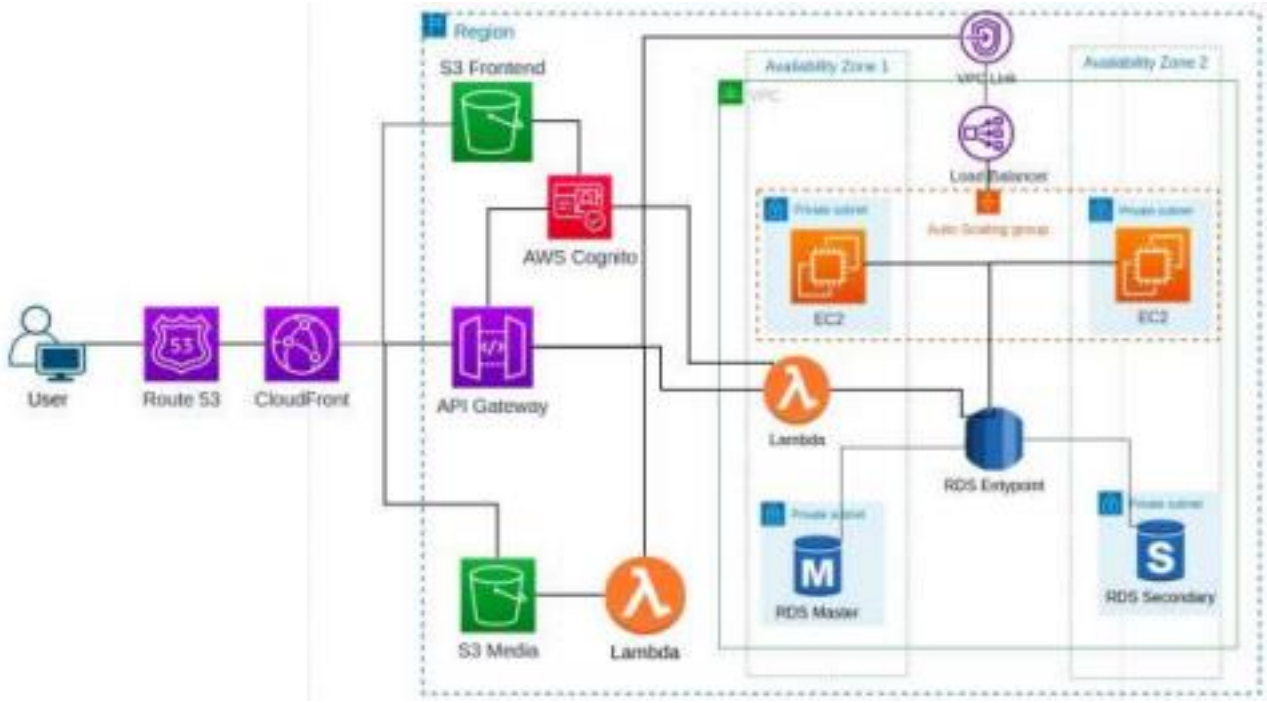


Рисунок 2.1- Архітектура застосунку

CloudFront кешує дані з таких компонентів:

1. Клієнтський рівень, розміщений на S3
2. API Gateway, який є точкою входу на серверний рівень
3. S3-сховище для медіафайлів.

Amazon Simple Storage Service (S3) дозволяє розміщувати статичні веб сайти, використовуючи файли на стороні клієнта (*.html, *.js, *.css) [16]. S3 зберігає дані щонайменше в трьох зонах доступності в межах одного регіону. Це дозволяє досягти доступності сервісу 99,99%. Клієнтські компоненти надсилають запити до Cognito та сервера через API-шлюз під час взаємодії з користувачами.

Для виконання таких операцій, як реєстрація, вхід, підтвердження електронної пошти тощо, клієнт надсилає запити до Cognito. Цей сервіс виконує авторизацію, автентифікацію та керування користувачами[17]. Завдяки йому ви можете використовувати надійні та безпечні рішення, що надаються постачальниками хмарних послуг, замість того, щоб писати власні рішення. Сервіс також забезпечує високу доступність, використовуючи кілька зон доступності в регіоні. Після створення нового користувача Cognito викличе

функцію Lambda, щоб додати дані користувача до бази даних.

Інші запити, здійснені через Route 53 та CloudFront, надсилаються безпосередньо клієнтом (або користувачем) до API Gateway. Сервіс дозволяє створювати та керувати бекенд-API для вашої програми [18]. Він перенаправляє запит до відповідного обчислювального сервісу. Крім того, якщо запит вимагає автентифікації доступу користувача, API Gateway використовує Cognito для виконання цієї дії. CloudFront дозволяє керувати кешуванням, тому дані, які рідко або ніколи не змінюються, будуть кешовані.

Послуга має високу доступність, а доступність встановлюється постачальником на таких рівнях

99,99%.

Бізнес-логіка виконується на платформах EC2 та Lambda. Екземпляри EC2 мають бути розміщені у VPC (віртуальній приватній хмарі), яка є логічно ізольованою віртуальною мережею. Для EC2 потрібно вибрати підмережу, яка являє собою групу IP-адрес у VPC. Кожна підмережа розташована в певній зоні доступності.

Екземпляри EC2 розміщені у двох приватних підмережах, що належать до різних зон доступності. Це критично важливо для покращення доступності системних компонентів, оскільки хмарні провайдери передають доступність користувачам, а не запускають автоматично екземпляри EC2 у різних зонах доступності. Для подальшого покращення відмовостійкості можна збільшити кількість використовуваних зон доступності.

Для досягнення автоматичного масштабування у відповідь на значне зростання навантаження на віртуальні машини ми використовуємо групи автомасштабування. Для рівномірного розподілу навантаження на різні екземпляри EC2 ми використовуємо балансувальник навантаження. Цей балансувальник використовує внутрішнє рішення та обмежує доступ до Інтернету для підвищення безпеки. Балансувальник підключено до каналу VPC, щоб шлюз API міг перенаправляти запити до екземплярів EC2. Постачальник забезпечує 99,99% доступність цих сервісів. EC2 взаємодіє з базою даних, яка

також розташована у VPC.

Рівень даних використовує службу баз даних RDS. Додаток використовує PostgreSQL. Для покращення відмовостійкості рівня даних прийнято розгортання з кількома зонами доступності. Два екземпляри бази даних розташовані в різних зонах доступності та автоматично синхронізуються. Вторинний екземпляр реплікує стан основного екземпляра, а основний екземпляр виконує операції запису. Якщо основний екземпляр недоступний, вторинний екземпляр візьме на себе його управління. Архітектура бази даних наведена в Додатку А.

Лямбда-функції використовуються для виконання повторюваних завдань. Лямбда-функції, які взаємодіють з базами даних для доступу до них, також виконуються у VPC. Інші функції використовуються для взаємодії зі сховищем S3 для зберігання медіафайлів. Як згадувалося в попередньому розділі, висока доступність лямбда-функцій забезпечується постачальником, і вони не потребують виконання додаткових операцій для цього, як це робить EC2.

Використання сервісів відмовостійкості AWS та додаткових удосконалень відмовостійкості віртуальних машин робить програми, що використовують вищезгадану архітектуру, високодоступними та відмовостійкими. Досягти такої надійності за допомогою самої програми складно, особливо враховуючи використання глобальних сервісів, таких як Route 53 та

CloudFront використовує інфраструктуру AWS, розташовану по всьому світу. Ця програма дозволяє неавторизованим користувачам:

- Перегляд новин, відсортованих за датою публікації.
- Переглядайте новини за категоріями.
- Переглянути всі категорії новин
- Переглянути коментарі до новин, залишені авторизованими користувачами.

Неавторизовані користувачі можуть зареєструватися. Для реєстрації користувачам необхідно надати таку інформацію: електронну пошту, ім'я користувача та пароль. Для підтвердження реєстрації користувачам необхідно

ввести код, надісланий Cognito (код буде надіслано користувачеві електронною поштою).

Функція реєстрації користувача:

- Змініть зображення профілю за замовчуванням.
- Прокоментуйте новину.
- Видалити свій коментар

Окрім звичайних користувачів, є також редактори новин, які можуть: ● Новини випуску

- Редагувати новини
- Видалити новини
- Додати нову категорію
- Редагувати категорію

РОЗДІЛ 3
РОЗРОБКА КОМПОНЕНТІВ СЕРВЕРА ЗАСТОСУНКІВ

3.1 Шлюз API та авторизатор

Сервіс API Gateway дозволяє створювати HTTP API, REST API та Websocket API. REST API є гнучкішими та багатшими на функції, ніж HTTP API, але вони також трохи дорожчі. Ми створимо REST API для новинного веб-сайту.

Таблиця 3.1 - Опис API

метод	Ресурси	Параметри	описувати	платформа

отримати	/стаття	Вебсайт: page – номер сторінки page_size – розмір сторінки category – категорія asc_order – сортувати за датою	Повертає список новин на основі параметрів	EC2
отримати	/стаття/{ідентифікатор}	PATH:id Новини	Назад до новин	EC2
Поштова служба	/стаття	Основна частина: Стаття у форматі JSON Назва: Редактор Теги	Новини випуску	Лямбда
покласти	/стаття/{ідентифікатор}	Основна частина: Стаття у форматі JSON PATH:id Новини Назва: Редактор Теги	Змініть новини	Лямбда

видалити	/стаття/{ідентифікатор}	PATH:id Новини	Видалити новини	Лямбда
отримати	/Категорія		Назад до списку категорій новин	EC2
Поштова служба	/Категорія	Тіло: Клас у форматі JSON Назва: Редактор Теги	Додати нову категорію	Лямбда
покласти	/Категорія	Тіло: Клас у форматі JSON Назва: Редактор Теги	Редагувати категорію	Лямбда

видалити	/категорія/{ідентифікатор} }	Шлях: Ідентифікатор категорії	Видалити новини	Лямбда
отримати	/article/{id}/comments	Шлях: URL адреса ідентифікатора новин: page – номер сторінки page_size – розмір сторінки asc_order – сортувати за датою	Повертає список коментарів до новини на основі параметрів	EC2
Поштова служба	/article/{id}/comments	PATH:id Новини Текст: Коментарі у форматі JSON Заголовок: Маркер користувача	Додати коментар до новини	EC2
видалити	/коментарі/{ідентифікатор}	Шлях: Ідентифікатор коментаря Заголовок: Користувач або редактор Токен	Видалити коментар	Лямбда
Поштова служба	/Зображення	Вміст: Зображення Назва: Редактор Теги	Додавання зображень до сховища S3	Лямбда
покласти	/Користувач/Зображення	Вміст: Зображення Заголовок: Ідентифікатор користувача	Змінити зображення профілю користувача	Лямбда

отримати	/користувач/{ідентифікатор}	Шлях: Ідентифікатор користувача Заголовок: Маркер користувача	Дозволити користувачам переглядати їхній профіль	EC2

Під час створення REST API, окрім створення нового API, ви можете

копіювати з існуючого API, імпортувати з Swagger або Open API 3, або використовувати зразок AWS. Ви також можете вибрати тип кінцевої точки: регіональна, приватна, доступ лише в межах VPC, оптимізована для периферійних розташування та підключена до CloudFront.

Рисунок 3.1- Створення REST API

У цьому випадку ми вирішуємо створити новий API під назвою news-api. Після цього API може підключитися до дистрибутива CloudFront. REST API надає можливість створювати авторизатори, які використовуються для перевірки того, чи має користувач дозвіл на здійснення запиту перед його надсиланням на сервер. Авторизатори можна розділити на два типи:

1.Лямбда

2.Когніція

Авторизатор Cognito перевіряє лише дійсність токена доступу та не може підтвердити, що користувач належить до потрібної групи в пулі користувачів Cognito.

У нашому випадку редактори новин належать до групи адміністраторів. Отже, щоб створити авторизатора для редакторів, ми використаємо функцію Lambda, яка спочатку надсилає запит до Cognito для перевірки токена доступу, потім аналізує токен і перевіряє, чи є він членом групи адміністраторів.

Для написання функції Lambda ми використовуємо мову програмування Go та AWS SKD [22]. Оскільки компілятор AWS Code не підтримує Go, нам потрібно його завантажити. Це можна зробити безпосередньо, завантаживши скомпільований код у zip-архіві. Оскільки код функції Lambda (див. Додаток В) використовує змінну середовища REGION, нам потрібно додати її до

середовища виконання функції, що можна зробити в меню налаштувань. У цьому випадку значення цієї змінної – eu-west-1.

Рисунок 3.2 -Нові змінні середовища

Після створення відповідної лямбда-функції буде створено авторизатор, який її використовує. Під час створення авторизатора потрібно вказати ім'я AdminAuth, тип лямбда, регіон виконання eu-west-1 (Ірландія) та корисне навантаження події лямбда - Token. Джерело токена вказує на заголовок, де розташована функція, а його значення - Authorization. Крім того, функція кешування ввімкнена, а кеш дійсний протягом 300 секунд. Перевірка токена дозволяє перевірити токен за допомогою регулярного виразу, але він не використовується. Роль виклику лямбда - це необов'язковий параметр, який дозволяє вказати роль з відповідними дозволами для виклику функції. Якщо не вказано, для створення авторизатора потрібно надати дозволи на доступ до API для виклику лямбда.

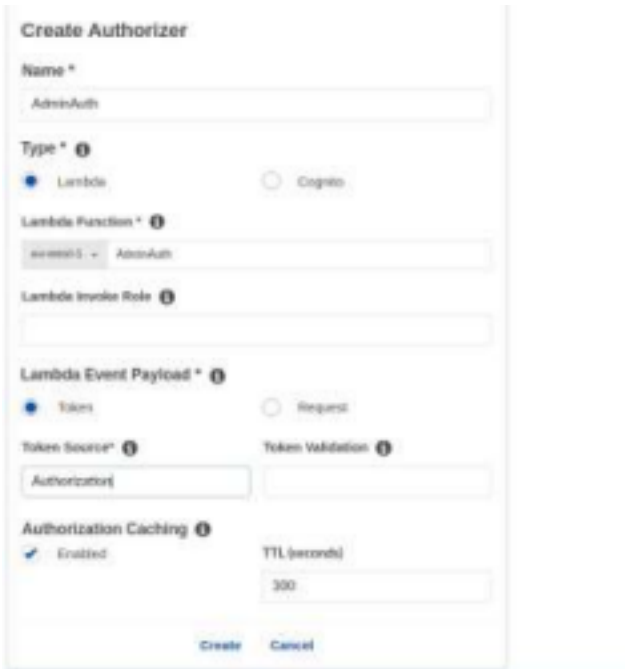


Рисунок 3.3 - Створення авторизатора

Рисунок 3.4 - Надання API Gateway дозволу на виклик функції

Після створення авторизатора ви можете його протестувати:

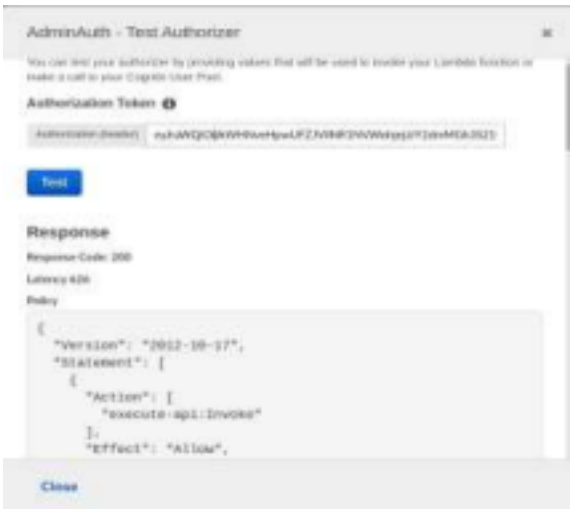


Рисунок 3.5 - Тестування створеного авторизатора

Для авторизації звичайних користувачів, які не є редакторами, не потрібно використовувати авторизатор Lambda. Потрібно лише створити авторизатор Cognito.

3.2 Структура серверних компонентів

Бекенди додатків поділяються на три рівні відповідно до їх функціонального використання:

1. Обробник – шар, що відповідає за обробку HTTP-запитів, які можуть бути надіслані до функцій EC2 або Lambda. Цей шар аналізує дані запиту та перевіряє їх правильність. Потім він передає дані запиту компоненту сервісного рівня, який виконує запит та повертає HTTP-відповідь користувачеві.

2. Сервіс – шар, який виконує бізнес-логіку. Він викликається компонентом Handler цього шару, обробляє дані, перевіряє їх та викликає компоненти Repository за потреби. Після виконання бізнес-логіки він передає відповідь назад компоненту Handler.
3. Репозиторій – шар, який взаємодіє з базою даних. Він викликається сервісним компонентом і повертає відповідь сервісному компоненту після обробки бази даних.

Такий підхід спрощує підтримку, модифікацію та розширення кодової бази. Кожен рівень програми має п'ять компонентів, по одному для кожної сутності: Статті, Категорії, Користувачі, Коментарі та Медіаконтент.

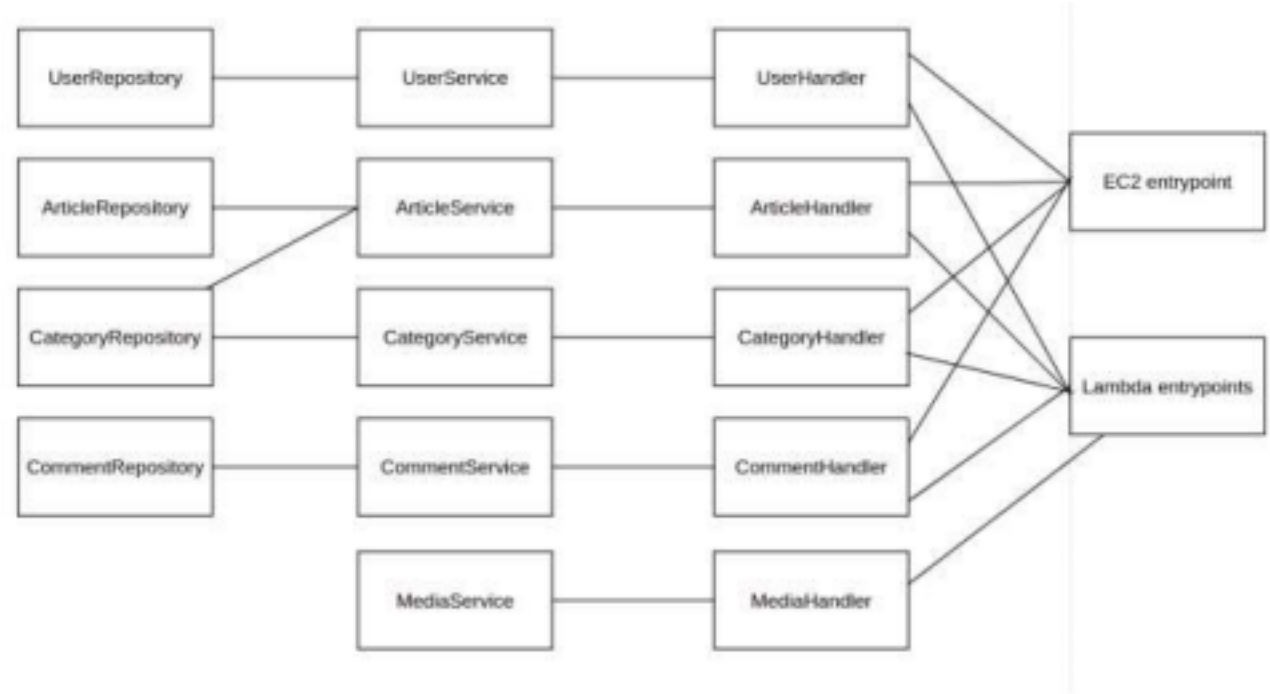


Рисунок 3.6 - Діаграма взаємодії компонентів бекенду

Компоненти сутностей переважно взаємодіють з компонентами тієї ж сутності. Винятком є ArticleService, який взаємодіє з CategoryRepository. Також Repository не використовується для медіаконтенту, оскільки AWS SDK використовується для взаємодії з S3.

Кожен компонент має свої власні методи. Наприклад, інтерфейс компонента сутності Article написаний на GO з використанням фреймворку Gin:

Репозиторій статей:

```
type ArticleRepository interface {
    InsertArticle(article model.Article, categoryID int) (*model.Article, error)
    UpdateArticle(article model.Article, categoryID int) (*model.Article, error)
    DeleteArticle(articleID int64) error
    GetArticle(articleID int64) (*model.Article, error)
    GetArticles(categoryID int, param model.GetArticlesParams) ([]model.Article, error)
}
```

Послуги зі статей:

```
type ArticleService interface {
    CreateArticle(requestParam model.CreateArticleRequest, editorID string) (*model.Article, error)
    UpdateArticle(article model.Article) (*model.Article, error)
    DeleteArticle(articleID int64) error
    GetArticle(articleID int64) (*model.Article, error)
    GetArticles(params model.GetArticlesParams) ([]model.Article, error)
}
```


Обробник статті:

```
type ArticleHandler interface {
    CreateArticle(c *gin.Context)
    UpdateArticle(c *gin.Context)
    DeleteArticle(c *gin.Context)
    GetArticle(c *gin.Context)
    GetArticles(c *gin.Context)
}
```

Повний список інтерфейсів компонентів наведено у Додатку В.

3.3 Детальні приклади розробки

Візьмемо для прикладу додавання новин і розглянемо серверну частину детальніше. Для функції додавання новин використовується обчислювальна платформа Lambda. Lambda була обрана тому, що час виконання цих функцій короткий, затримки, пов'язані з холодними стартами, прийнятні, а частота додавання новин відносно низька.

Метод CreateArticle компонента ArticleHandler для додавання нових статей приймає об'єкт JSON з наступної функції:

{
"Назва":"Назва",
Категорія:Категорія,
"контент":"контент"
}

Щоб додати новину, усі поля мають бути непорожніми. Після перевірки полів у JSON-об'єкті буде повернуто об'єкт такого типу:

Потім він передається методу CreateArticle компонента ArticleService разом з ідентифікатором редактора, отриманим з токена в заголовку авторизації. Тут це дата публікації та об'єкт типу:

Він також перевіряє наявність отриманої категорії в таблиці категорій та отримує ідентифікатор категорії за допомогою методу GetArticleIDByName компонента ArticleRepository. Після цього компонент ArticleRepository додає статтю до бази даних та повертає об'єкт типу Article, який містить згенерований ідентифікатор. Цей об'єкт походить від ArticleService та передається до ArticleHandler, який повертає відповідь користувачеві. Якщо на будь-якому кроці виникає помилка, користувачеві також буде повернуто відповідь.

Рисунок 3.7 - Діаграма успішної обробки запиту

Щоб лямбда-функція мала доступ до бази даних, вона має бути виконана в VPC. Щоб лямбда-функція була приєднана до VPC, вона повинна мати дозволи на виконання в VPC. Для цього AWS надала готову політику доступу

AWSLambdaVPCAccessExecutionRole, яку можна використовувати для лямбда функцій.

Рисунок 3.8 - Політика доступу awslambdavpcaccessexecutionrole

Потім у REST API створюється метод POST для ресурсу Articles за допомогою функції Lambda. Конфігурація використовує інтеграцію Lambda Proxy, щоб забезпечити передачу даних запиту до функції Lambda у форматі, отриманому шлюзом API. Також встановлено спеціальне значення тайм-ауту на 5 секунд замість стандартних 29 секунд. Після створення авторизатор, створений раніше для редактора, встановлюється в запиті методу.

Рисунок 3.9- Додавання лямбда-функції до REST API

Після цього ви можете розгорнути API та зробити запити до сервера:

Рисунок 3.10 - Приклад успішного запиту на створення новинного елемента

Якщо додавання нових новин виконується через функцію Lambda, то запит на отримання списку новин виконується платформою EC2. Такий підхід обрано тому, що користувачі часто перевіряють список новин, і також важливо мінімізувати затримку у поверненні результатів. Крім того, така ситуація трапляється часто, і якщо використовується Lambda, її вартість може бути набагато дорожчою, ніж у EC2.

Щоб дозволити користувачам гнучко переглядати списки новин, запит може приймати 4 параметри:

- сторінка – номер сторінки новини для перегляду, починаючи з 0
- page_size – кількість новин на сторінці
- Категорія – категорія новин, яка буде повернута
- asc_order – Сортує новини за датою публікації. Якщо asc_odred=false, новини повертаються в порядку від найновіших до найстаріших.

Коли ArticleHandler обробляє запит за допомогою методу GetArticles, ці параметри розбиваються на об'єкти таких типів:

У цьому випадку, якщо page_size перевищує 100, користувач отримає помилку. Розмір сторінки обмежений, щоб уникнути перевантаження сервера великими сторінками. Після цього ці параметри передаються до ArticleService. Якщо користувач вказує категорію в параметрах, вона перевіряється на наявність, як і під час додавання нової статті. Крім того, якщо користувач не вказує розмір сторінки, встановлюється 20 елементів. Після цього викликається метод GetArticles компонента ArticleRepository, який запитує базу даних на основі параметрів, переданих користувачем, і повертає список статей.

Щоб інтегрувати екземпляри EC2, розташовані в приватній мережі, трафік яких розподіляється внутрішнім балансувальником навантаження, потрібно створити VPC-з’єднання з API Gateway. Потім, під час додавання методу до API Gateway, виберіть тип інтеграції «VPC-з’єднання».

Рисунок 3.11 - Встановлене VPC-з'єднання

Потім ви можете подати запит на отримання списку новин:

Рисунок 3.12 - Приклад успішного запиту на отримання списку новин

ВИСНОВОК

В результаті виконання цієї роботи ми переглянули можливості хмарної платформи AWS щодо забезпечення високої доступності програм. Зокрема, ми розглянули обчислювальну платформу та платформні рішення, що використовуються для забезпечення міжрегіонального доступу.

Серед обчислювальних платформ ми розглянули та порівняли віртуальні машини EC2, контейнерні платформи та безсерверні лямбда-функції. Серед рішень для міжрегіонального доступу, що надаються AWS, ми розглянули та порівняли Route 53, CloudFront та Global Accelerator. Було проаналізовано та доведено можливість повноцінного впровадження застосунку на хмарній платформі та виконання всіх вимог щодо відмовостійкості та функціональності.

Для новинного веб-сайту було побудовано високодоступну багатокомпонентну архітектуру з використанням таких сервісів: EC2, Lambda, Route 53, CloudFront, Cognito, API Gateway, RDS та S3.

Відповідно до функцій новинного веб-сайту було розроблено базу даних

PostgreSQL.

Детально описує API серверного рівня програми та обговорює процес створення REST API за допомогою служби API Gateway та налаштування для неї авторизації адміністратора.

Досліджено структуру програмного забезпечення серверного рівня новинного веб-сайту, а також детально описано розробку деяких елементів серверного рівня, підключення REST API та демонстрацію роботи.

У ході проведеного дослідження було здійснено комплексний аналіз можливостей хмарної платформи Amazon Web Services (AWS) щодо забезпечення високої доступності, масштабованості та відмовостійкості програмних систем, зокрема в контексті розгортання новинного веб-сайту з багатокомпонентною архітектурою. Основна увага приділялася практичному вивченню та порівнянню

обчислювальних і мережевих сервісів, що використовуються для побудови сучасних хмарних рішень.

Серед обчислювальних рішень особливу увагу було приділено трьом основним моделям обробки даних в AWS — віртуальним машинам Amazon EC2, контейнерним платформам (наприклад, AWS ECS/EKS) та безсерверним обчисленням (AWS Lambda). Проведене порівняння дозволило оцінити сильні сторони кожного підходу з погляду продуктивності, гнучкості масштабування, вартості та простоти інтеграції у великомасштабні системи.

У сфері мережевих рішень було досліджено сервіси Route 53, CloudFront та AWS Global Accelerator як ключові інструменти для забезпечення географічно розподіленого доступу, низьких затримок і підвищеної стійкості до збоїв. Встановлено, що комбінація цих сервісів дозволяє досягти високого рівня SLA (Service Level Agreement) у міжрегіональному масштабі.

На основі отриманих результатів було доведено можливість повноцінного впровадження веб-застосунку на платформі AWS з урахуванням вимог до безперервної доступності, надійності та функціонального різноманіття. Запропонована архітектура використовує взаємодію між такими сервісами, як EC2, Lambda, Route 53, CloudFront, Cognito, API Gateway, Amazon RDS

(PostgreSQL) та S3, що забезпечує логічно розділену, масштабовану і безпечну інфраструктуру.

Особливу увагу було приділено побудові серверної частини застосунку, зокрема — проектуванню REST API за допомогою API Gateway, реалізації механізмів авторизації з використанням AWS Cognito, а також опису процесу підключення фронтенду до серверної логіки. Це дозволило не лише досягти високої продуктивності та безпеки, а й продемонструвати приклад реального використання хмарних технологій у сучасному веб-розробленні.

У підсумку, робота підтверджує ефективність використання AWS як надійної платформи для створення розподілених веб-систем з підвищеною

доступністю та відмовостійкістю, особливо актуальних для інформаційних ресурсів типу новинних порталів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Overview of Amazon Web Services: AWS Whitepaper [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/whitepapers/latest/aws overview/awsoverview.pdf
2. Amazon Elastic Compute Cloud: User Guide for Linux Instances [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/AWSEC2/latest/UserGuide/ec2-ug.pdf 3. Cloud Computing Service Models – IaaS, PaaS, SaaS [Електронний ресурс] – Режим доступу до ресурсу: https://digitalcloud.training/cloud computingservice-models-iaas-paas-saas/
4. ECS vs EC2 vs Lambda [Електронний ресурс] – Режим доступу до ресурсу: https://digitalcloud.training/ecs-vs-ec2-vs-lambda/
5. What's The Difference Between Containers And Virtual Machines? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/compare/the-difference-between-containers-andvirtual machines/

6. Docker on AWS: AWS Whitepaper [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/whitepapers/latest/dockeronaws/docker-on-aws.pdf

7. AWS Lambda: Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/lambda/latest/dg/lambda-dg.pdf 8. What is AWS Lambda? [Електронний ресурс] – Режим доступу до ресурсу: https://www.serverless.com/aws-lambda/

9. Reliability Pillar: AWS Well-Architected Framework [Електронний ресурс] – Режим доступу до ресурсу:

https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliabilitypillar/wellarchitected-reliability-pillar.pdf

10. What is Route 53 and What are its functionalities? [Електронний ресурс] – Режим доступу до ресурсу: https://www.novelvista.com/blogs/cloud-and-aws/what-is-route-53-and-whatare its-functionalities

11. Amazon Route 53: Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/Route53/latest/DeveloperGuide/route53dg.pdf

12. What is Amazon CloudFront and How Does It Work? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/mindful-engineering/today-we-will-learn-about cloudfront690bf3a8819a

13. Amazon CloudFront Key Features [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/cloudfront/features/?nc=sn&loc=2&whatsnewcloudfront.sort-by=item.additionalFields.postDateTime&whatsnewcloudfront.sort-order=desc

14. Amazon CloudFront: Developer Guide [Електронний ресурс] – Режим

доступу до ресурсу:
https://docs.aws.amazon.com/pdfs/AmazonCloudFront/latest/DeveloperGuide/AmazonCloudFront_DevGuide.pdf
15. AWS Global Accelerator: Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/pdfs/globalaccelerator/latest/dg/globalaccelerator-guide.pdf
16. Amazon Web Services – Global Accelerator [Електронний ресурс] – Режим доступу до ресурсу:

https://www.geeksforgeeks.org/amazon-web-services-global-accelerator/ 17. AWS CloudFront vs Global Accelerator [Електронний ресурс] – Режим доступу до ресурсу: https://jayendrapatil.com/aws-cloudfront-vs-global accelerator/
18. Amazon Simple Storage Service User Guide [Електронний ресурс] – Режим доступу до ресурсу:
https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
19. Amazon Cognito Developer Guid [Електронний ресурс] – Режим доступу до ресурсу:
https://docs.aws.amazon.com/pdfs/cognito/latest/developerguide/cognito-dg.pdf
20. Amazon API Gateway Developer Guide [Електронний ресурс] – Режим доступу до ресурсу:
https://docs.aws.amazon.com/pdfs/apigateway/latest/developerguide/apigateway-dg.pdf
21. Amazon Relational Database Service User Guide [Електронний ресурс] – Режим доступу до ресурсу:
https://docs.aws.amazon.com/pdfs/AmazonRDS/latest/UserGuide/rds-ug.pdf
22. AWS SDK for Go v2 [Електронний ресурс] – Режим доступу до ресурсу:
https://github.com/aws/aws-sdk-go-v2

ДОДАТКИ

Додаток А

Схема бази даних застосунку

Додаток Б

Код Lamba функції для авторизатора редакторів:

```
package main
 import
 (
       "context"
       "fmt"
       "os"
       "strings"


       "github.com/aws/aws-lambda-go/events"
       "github.com/aws/aws-lambda-go/lambda"
       "github.com/aws/aws-sdk-go/aws"
       "github.com/aws/aws-sdk-go/aws/session"
       "github.com/aws/aws-sdk-go/service/cognitoidentityprovider"
```

```
	"github.com/dgrijalva/jwt-go"
)
const
(
 allow = "Allow" deny =
"Deny"
) func main() {
lambda.Start(authorizerHandler)
} func authorizerHandler(ctx context.Context,
req events.APIGatewayCustomAuthorizerRequest)
(events.APIGatewayCustomAuthorizerResponse, error)
{ token := req.AuthorizationToken //
Створення сесії AWS
	sess := session.Must(session.NewSession(&aws.Config{
		Region: aws.String(os.Getenv("REGION")),
	}))
	// Створення клієнта для взаємодії з Cognito
cognitoClient := cognitoidentityprovider.New(sess) // Параметри для
запиту до Cognito, який приймає AccessToken getUserInput  :=
&cognitoidentityprovider.GetUserInput{
		AccessToken: aws.String(token),
	}
	// Валідація токену за допомогою GetUser, оскільки метод який це робить
напряму недоступний в SDK
	_, err := cognitoClient.GetUser(getUserInput)
if err != nil {
		// Відмова в доступі, якщо токен невалідний
return generatePolicy(deny, req.MethodArn), nil
	}
```



```
	// Перевірка приналежності до групи Administrator в токені
isAdmin := isAdmin(token)
	// Надання чи відмова в доступі в залежності від значення isAdmin
return generatePolicy(getEffect(isAdmin), req.MethodArn), nil  }
// Перевірка того, чи входить користувач в групу адміністраторів за допомогою
токену func isAdmin(tokenString string) bool { token, _ :=
jwt.Parse(tokenString, func(token *jwt.Token) (interface{}, error) {  return
[]byte(""), nil
	})
	claims, _ := token.Claims.(jwt.MapClaims) return
strings.Contains(fmt.Sprintf("%v", claims["cognito:groups"]),
"Administrator")
} func getEffect(isAdmin bool)
string {
	if isAdmin {
return allow
	}
	return deny
}
// Функція генерації відповіді для авторизатора func
generatePolicy(effect, resource string)
events.APIGatewayCustomAuthorizerResponse { return
events.APIGatewayCustomAuthorizerResponse{
		PolicyDocument: events.APIGatewayCustomAuthorizerPolicy{
			Version: "2012-10-17",
```

```
                Statement: []events.IAMPolicyStatement{
                        {
                                Action: []string{"execute-api:Invoke"},
                              Effect: effect,
                                 Resource: []string{resource},
                        },
                },
        },
    }
}
```


Додаток В

Повний список інтерфейсів застосунку: Repository:

Service:

Handler:

Додаток Г

Код для додавання новин:

repository/article.go

```
import (
"database/sql"
	"fmt"
	"news/model"
)

// Структурра, що реалізує інтерфейс ArticleRepository
type PSQLArticleRepository struct {
	db *sql.DB
}
```

```
// Конструктор
func NewPSQLArticleRepository(db *sql.DB) *PSQLArticleRepository {
	return &PSQLArticleRepository{
		db: db,
	}
}

// Метод для додавання статті в SQL базу даних
func (r *PSQLArticleRepository) InsertArticle(a model.Article, categoryID int)
(*model.Article, error) {
	// Текс SQL запиту, який додає статтю та повертає її id
insertArticleQuery := `INSERT INTO articles (
title, category, content, editor_id, publication_date
) VALUES ($1, $2, $3, $4, $5)  RETURNING id` // Додавання статті та
встановлення ArticleID err := r.db.QueryRow(insertArticleQuery,  a.Title,
categoryID, a.Content, a.EditorID, a.PublicationDate).  Scan(&a.ArticleID)
	return &a, err
}

...
```

service/article.go

```
import (
	"log"
	"news/model"
	"news/repository"
	"time"
)

// Структура, яка імплементує інтерфейс ArticleService
// Складається з агрегованих структур, які імплементують інтерфейси //
ArticleRepository та CategoryRepository type
ArticleServiceStruct struct { articleRepo
```



```
repository.ArticleRepository categoryRepo
repository.CategoryRepository }

// Конструктор
func NewArticleServiceStruct(articleRepo repository.ArticleRepository,
categoryRepo repository.CategoryRepository) *ArticleServiceStruct { return
&ArticleServiceStruct{
		articleRepo: articleRepo,
		categoryRepo: categoryRepo,
	}
}

// Метод для створення новин
func (s *ArticleServiceStruct) CreateArticle(
requestParam model.CreateArticleRequest,
editorID string) (*model.Article, error) {
	// Створення структури Article
	article := articleRequestToArticle(requestParam, editorID)

	// Перевірка того, чи є отримана категорія статті в базі даних //
	і повернення ID категорії
	categoryID, err := s.categoryRepo.GetCategoryIDByName(article.Category)  if
err != nil { // Повернення помилки, якщо вона виникла
```

```
            return nil, err
      }
      // Додавання статті в БД і повернення результату виконання
InsertArticle return s.articleRepo.InsertArticle(article, categoryID) }  //
Допоміжна функція, яка створює структуру Article з CreateArticleRequest та
editorID
func articleRequestToArticle(request model.CreateArticleRequest,
editorID string) model.Article { return model.Article{
            Title: request.Title,
            Category: request.Category,
            Content: request.Content,
            EditorID: editorID,
            PublicationDate: time.Now(),
      }
}
```

handler/article.go

```
// Структура, яка імплементує інтерфейс ArticleHandler
// Складається структури, яка імплементує інтерфейс
// ArticleService
type ArticleHandlerStruct struct {
articleServ service.ArticleService
}


// Конструктор для створення ArticleHandlerStruct
func NewArticleHandlerStruct(articleServ service.ArticleService)
*ArticleHandlerStruct {
      return &ArticleHandlerStruct{articleServ: articleServ}
}
```



```
// Метод для створення статті з отриманого запиту func (h
*ArticleHandlerStruct) CreateArticle(c *gin.Context) {
      var articleRequest model.CreateArticleRequest
      // Парсинг тіла з JSON в структуру CreateArticleRequest та перевірка //
      на те, чи заповнені поля err
:= c.BindJSON(&articleRequest)
      // Надсилання StatusBadRequest та помилки при невалідності тіла
запиту if err != nil { sendError(c, http.StatusBadRequest,  err.Error())
return
      }
      // Отримання ID редактора з токену
authHeader := c.GetHeader("Authorization")
editorID := getEditorID(authHeader)
      // Виклик ArticleService для створення статті
      article, err := h.articleServ.CreateArticle(articleRequest, editorID)  //
      Надсилання помилки у разі її виникнення
if err != nil {
            if errors.Is(err, sql.ErrNoRows) {
                  // Якщо помилка пов'язана з тим, що категорії не існує,
відправляється StatusBadRequest
 categoryError := fmt.Sprintf("category %s doesn`t exist",
articleRequest.Category)
                  sendError(c, http.StatusBadRequest, categoryError)
            } else {
                  // В іншому випадку StatusInternalServerError
                   sendError(c, http.StatusInternalServerError, err.Error())
            }
```

```
            return
      }
      // Надсилання StatusOK відповіді та доданої статті у JSON форматі
c.JSON(http.StatusOK, article)
}


// Допоміжна функція для отримання ID редактора з токену
func getEditorID(auth string) string {
      token, _ := jwt.Parse(auth, func(token *jwt.Token) (interface{}, error) {
      return []byte(""), nil
      })
      claims, _ := token.Claims.(jwt.MapClaims)
return fmt.Sprintf("%s", claims["sub"])
}
```

cmd/create-article/main.go

```
var ginLambda *ginadapter.GinLambda


func init() {
      // Створення gin роутеру
r := gin.Default() //
Підключення до бази даних
      db, err := sql.Open("postgres", os.Getenv("DB_CONNECTION_STRING"))  if
err != nil {
            panic(err)
```



```
      }
      // Ініціалізація ArticleHandlerStruct
articleHandler := app.InitArticleHandler(db)
      // Додавання CreateArticle до роутеру
      r.POST("/articles", articleHandler.CreateArticle)
// Створення ginLambda функції з роутеру
ginLambda = ginadapter.New(r)
}
func Handler(ctx context.Context, req events.APIGatewayProxyRequest)
 (events.APIGatewayProxyResponse, error) {
      // Використання ginLambda проксі для взаємодії роутера gin з
APIGatewayProxyRequest return
ginLambda.ProxyWithContext(ctx, req)
}
func main() {
      // Старт lambda функції
      lambda.Start(Handler)
}
```


Додаток Д

Код для отримання списку новин: repository/article.go

```
// Метод для отриммання списку новин з бази даних за параметрами
func (r *PSQLArticleRepository) GetArticles(categoryID int,
param model.GetArticlesParams) ([]model.Article, error) { //
створення запиту та параметрів для нього
      getArticlesQuery, queryParams := buildQueryAndParams(categoryID, param)  //
      виконання запиту
      rows, err := r.db.Query(getArticlesQuery, queryParams...)
if err != nil {
```

```
            return nil, err
      }
      // оголошення змінної для зберігання списку
новин articles := []model.Article{} for
rows.Next() { var a model.Article
            // заповнення новини даними з бази даних
            if err := rows.Scan(&a.ArticleID, &a.Title, &a.Category, &a.Content,
&a.EditorID, &a.PublicationDate); err != nil {
                  return nil, err
            }
            // додавання новини в список
            articles = append(articles, a)
      }
      // закриття читання новин if err
:= rows.Close(); err != nil {
            return nil, err
      }
      // перевірка чи виникла помилка під час ітерації по рядкам
if err := rows.Err(); err != nil {
            return nil, err
      }
      // повернення результату
      return articles, nil
}


// Метод побудови запиту для БД та створення списку параметрів для запиту func
buildQueryAndParams(categoryID int, param model.GetArticlesParams) (string,
[]interface{}) {
      // Основні елементи для запиту queryParams :=
 []interface{}{ param.PageSize, param.PageSize
* param.Page,
      }
      // Основа запиту getArticlesQuery := `SELECT A.id, A.title,  C.name,
A.content, A.editor_id, a.publication_date
                                          FROM articles A INNER JOIN categories C ON
A.category = C.id`
      sortOrder := ""
      if param.AscendingOrder {
sortOrder = "ASC"
      } else {
            sortOrder = "DESC"
```



```
      }
      // Якщо користувач вказав котегорію
      if categoryID != -1 {
              // вставка категорії першим елементом в список параметрів
    queryParams = append([]interface{}{categoryID}, queryParams...) //
  формування кінцевого запиту разом з пошуком по категорії та сортуванням
           getArticlesQuery = getArticlesQuery + fmt.Sprintf(`
                                             WHERE A.category = $1
                                              ORDER BY A.publication_date %s
                                            LIMIT $2
                                             OFFSET $3`, sortOrder)
      } else {
            // формування кінцевого запиту з сортуванням
            getArticlesQuery = getArticlesQuery + fmt.Sprintf(`
                                              ORDER BY A.publication_date %s
                                            LIMIT $1
```

```
                                                OFFSET $2`, sortOrder)
      }
      return getArticlesQuery, queryParams
}
```

service/article.g o

```
// Отримання списку новин за параметрами
func (s *ArticleServiceStruct) GetArticles(params model.GetArticlesParams)
([]model.Article, error) {
categoryID := -1
      // Якщо користувач не вказав розмір сторінки, встановлюється значення 20
if params.PageSize == 0 {
            params.PageSize = 20
      }
      log.Printf("params: %v\n", params)
if params.Category != "" {
            // Перевірка того, чи є отримана категорія новини в базі даних  //
            і повернення ID категорії
            databaseID, err :=
s.categoryRepo.GetCategoryIDByName(params.Category) if err  != nil {
// Повернення помилки, якщо вона виникла
                  return nil, err
            }
            categoryID = databaseID
      }
      // повернення результату виконання методу компонента ArticleRepository
return s.articleRepo.GetArticles(categoryID, params)
}
```

handler/article.g o

```
// Отримання списку новин
func (h *ArticleHandlerStruct) GetArticles(c *gin.Context) {
var getArticlesParams model.GetArticlesParams
      // Парсинг тіла параметрів в структуру GetArticlesParams та їх
перевірка err := c.BindQuery(&getArticlesParams)
```



```
      // Надсилання StatusBadRequest та помилки при невалідності параметрів
запиту
      if err != nil {
            sendError(c, http.StatusBadRequest, err.Error())
return
      }
      //// Виклик ArticleService для отримання новин
      articles, err := h.articleServ.GetArticles(getArticlesParams)  //
      Надсилання помилки у разі її виникнення
if err != nil {
            if errors.Is(err, sql.ErrNoRows) {
                  // Якщо помилка пов'язана з тим, що категорії не існує,
 відправляється StatusBadRequest categoryError :=  fmt.Sprintf("category
            %s doesn`t exist", getArticlesParams.Category)  sendError(c,
                              http.StatusBadRequest, categoryError)
            } else {
                  // В іншому випадку StatusInternalServerError
sendError(c, http.StatusInternalServerError, err.Error())
            }
            return
      }
      // Надсилання новин користувачу в JSON форматі
```

c.JSON(http.StatusOK, articles)

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
Державного некомерційного підприємства
«Державний університет «Київський авіаційний інститут»

ВІДГУК
керівника кваліфікаційної роботи

випускника спеціальності 123 Комп'ютерна інженерія
факультет/відділення «Комп'ютерної і програмна інженерія»
Дмитра СИВАЧКА
(Ім'я, ПРІЗВИЩЕ)

Актуальність теми: тема «Забезпечення безперервної роботи веб-додатків за допомогою хмарних технологій» є надзвичайно актуальною та важливою в контексті сучасного розвитку інформаційних технологій. Залежність бізнесу та користувачів від безперебійної роботи веб-додатків зростає експоненційно. Будь-які простої або зниження доступності можуть призвести до значних фінансових втрат та репутаційних ризиків. Дослідження та практична реалізація методів забезпечення високої доступності та відмовостійкості за допомогою хмарних технологій є ключовим для побудови надійних та масштабованих систем. Автор у вступі вдало обґрунтовує важливість теми, вказуючи на багатокомпонентну архітектуру сучасних додатків та переваги розподілених мережевих ресурсів.

Змістовність та структура роботи: Робота має логічну та послідовну структуру, що відповідає вимогам до кваліфікаційних робіт. Вона складається із вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків.

Наукова новизна та практична цінність: Робота несе елементи новизни в частині системного підходу до інтеграції різноманітних хмарних сервісів AWS для побудови єдиного, високодоступного веб-додатку. Практична цінність полягає у розробці конкретної архітектури та її реалізації, що може слугувати як методичний посібник або основа для розробки подібних веб-додатків із забезпеченням високої доступності в хмарному середовищі AWS.

Загальні висновки: Кваліфікаційна робота Дмитра Сивачка є завершеним науковим дослідженням, що демонструє глибокі теоретичні знання автора в області хмарних технологій, архітектури веб-додатків та забезпечення їх безперервної роботи. Практична частина роботи підтверджує вміння автора застосовувати ці знання на практиці для вирішення конкретних інженерних задач. Робота написана грамотно, матеріал викладено логічно та послідовно.

Кваліфікаційна робота повністю відповідає вимогам, що висуваються до робіт освітньо-професійного ступеня фаховий молодший бакалавр за спеціальністю 123 «Комп'ютерна інженерія», і заслуговує на оцінку «добре».

Керівник кваліфікаційної роботи викладач, ктн, доцент
(науковий ступінь, посада)
«___» _____ 20__ р. (підпис) *Ірина ВДОВИЧЕНКО*
(Ім'я, ПРІЗВИЩЕ)