Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання»

Виконав: студент 2-го курсу, групи 2КІ-24м,
спеціальності 123 – «Комп'ютерна інженерія»
_____Марценюк Д. В.

Керівник: к.т.н., доц. кафедри ОТ
_____Войцеховська О.В.
«12» 12 2025 р.

Опонент: к.т.н., доц. кафедри ПЗ
_____Рейда О.М.
«12» 12 2025 р.

Допущено до захисту
Завідувач кафедри
обчислювальної техніки
д.т.н., проф. Азаров О. Д.
«15» 12 2025 р.
_____

ВНТУ 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень магістр
Спеціальність – 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
проф., д.т.н. О. Д. Азаров
_____
(підпис)
«25» вересня 2025 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Марценюку Денису Вячеславовичу

(прізвище, ім'я, по батькові)

1 Тема роботи: «Метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання», керівник роботи Войцеховська Олена Валеріївна к.т.н. доц., затверджені наказом вищого навчального закладу від «24» вересня 2025 року № 313

2 Строк подання студентом роботи 4 грудня 2025 року.

3 Вихідні дані до роботи: максимальний час відновлення системи після аварії Recovery Time Objective – 10 хв, гібридна архітектура розгортання, сервіси хмарного провайдера AWS.

4 Зміст текстової частини: вступ, аналітичний огляд технологій аварійного відновлення інформаційної інфраструктури, розробка методу аварійного відновлення додатків у хмарному середовищі, практична реалізація методу аварійного відновлення, тестування методу аварійного відновлення, економічна частина, висновки, список використаних джерел, додатки.

5 Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема архітектури методу аварійного відновлення додатку, схема архітектури методу аварійного відновлення з використанням DMS; Блок-схема алгоритму роботи методу автоматичного аварійного відновлення додатку; Блок-

схема алгоритму роботи модифікації методу автоматичного аварій відновлення додатку з використанням «гарячої» бази даних.

6 Консультантів розділів роботи представлено в табл. 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1–4	Войцеховська О. В., к.т.н., доц. кафедри ОТ		
5	Адлер О. О., к.т.н, доц. кафедри ОТ		
Нормоконтроль	Швець С.І., асистент кафедри ОТ		

7 Дата видачі завдання 25.09 2025 року.

8 Календарний план наведено в табл. 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Підпис
1	Постановка мети та задач роботи	08.09.2025	
2	Аналітичний огляд технологій аварійного відновлення інформаційної інфраструктури	11.09.2025	
3	Розробка методу аварійного відновлення додатків у хмарному середовищі	01.10.2025	
4	Практична реалізація методу аварійного відновлення	21.10.2025	
5	Тестування методу аварійного відновлення	24.10.2025	
6	Підготовка економічної частини	03.11.2025	
7	Оформлення пояснювальної записки, графічного матеріалу та презентації	17.11.2025	

Студент _____ Денис МАРЦЕНЮК

Керівник роботи _____ Олена ВОЙЦЕХОВСЬКА

АНОТАЦІЯ

УДК 004.056

Марценюк Д. В. Метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання. Магістерська кваліфікаційна робота зі спеціальності 123 — комп'ютерна інженерія, освітня програма комп'ютерна інженерія. Вінниця: ВНТУ, 2025, 108с.

На укр.мові. Бібліогр.: 29 назв, рис. 25, табл. 9.

В магістерській кваліфікаційній роботі проаналізовано існуючі методи аварійного відновлення роботи додатків. Розглянуто хмарні й локальні ресурси для резервного копіювання важливої інформації.

Розроблено метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання на платформі Amazon Web Services, який відновлює критичні ресурси замість відновлення всієї системи, автоматизує процес аварійного відновлення без ручного втручання завдяки використанню сервісу AWS Lambda, дозволяє зменшити час простою та підвищити рівень доступності додатку.

Проведене тестування показало, що при застосуванні розробленого методу потрібно вдвічі менше часу для відновлення роботи додатку, ніж в існуючих методах.

Ключові слова: вебдодаток, резервна копія, відновлення, Disaster Recovery, AWS.

ANNOTATION

Martseniuk D. V. Method of Disaster Recovery for Applications in a Cloud Environment Using a Hybrid Deployment Architecture. Master's Thesis in specialty 123 — Computer Engineering, educational program "Computer Engineering". Vinnytsia: VNTU, 2025, 108 pages.

In Ukrainian. Bibliography: 29 titles, 25 picture, 9 tables.

The master's thesis analyzes existing methods of disaster recovery for application operation. Cloud and local resources for backing up critical information are examined.

A disaster recovery method for applications in a cloud environment using a hybrid deployment architecture on the Amazon Web Services platform was developed. The proposed method restores only critical resources instead of the entire system, automates the disaster recovery process without manual intervention using the AWS Lambda service, and helps reduce downtime and increase application availability.

Testing showed that applying the developed method requires half the time to restore application operation compared to existing methods.

Keywords: web application, backup, recovery, disaster recovery, AWS.

ЗМІСТ

ВСТУП

У сучасному світі спостерігається тенденція до зростання залежності організацій від цифрових технологій та безперервності бізнес-процесів. Більшість компаній використовують складні вебдодатки, корпоративні портали або інтегровані цифрові платформи, що забезпечують функції управління клієнтським досвідом, продажами, розрахунком аналітики та автоматизацією бізнес-процесів. Такі системи нерідко є критично важливою частиною бізнес-інфраструктури, а їх безперебійна робота безпосередньо впливає на репутацію, прибутковість і конкурентоспроможність підприємства. Таким чином, безперебійна робота вебдодатків є критично важливою умовою стабільного функціонування підприємства.

Однією із головних характеристик цифрових систем є їх доступність. Час, протягом якого додаток не працює (час простою), є вирішальним показником, особливо для онлайн-сервісів, де навіть короткий збій може призвести до фінансових втрат, порушення зобов’язань перед клієнтами та зниження репутації компанії.

Для України питання надійності IT-інфраструктури набуває ще більшої актуальності через регулярні планові та аварійні відключення електроенергії, які можуть бути спричинені пошкодженням енергетичної інфраструктури, дефіцитом генеруючих потужностей або військовими діями. За таких умов важливо впроваджувати механізми аварійного відновлення (Disaster Recovery), які дозволяють мінімізувати час простою та забезпечити безперервність роботи критично важливих систем.

Актуальність методу впровадження аварійного відновлення додатків у хмарному середовищі зумовлена необхідністю забезпечення безперервної роботи додатків та мінімізації простоїв у разі технічних збоїв, відключеннях електроенергії, а також підвищення рівня доступності інфраструктури в умовах

воєнних дій. Використання гібридної архітектури розгортання дозволяє ефективніше поєднувати локальні ресурси та хмарні сервіси, забезпечуючи вищу гнучкість і надійність. Такий підхід мінімізує ризики втрати даних і часу простою в разі аварійної ситуації, що є критичним для стабільної роботи бізнесу. Це підкреслює актуальність теми для розробки стратегічних та інноваційних рішень в управлінні сучасною IT-інфраструктурою.

Метою дослідження є підвищення стабільності та зменшення часу недоступності web-сервісів під час непередбачуваних ситуацій, шляхом використання гібридної архітектури розгортання додатків у хмарному середовищі Amazon Web Services.

Задачі дослідження:

— дослідити технології розгортання інформаційної інфраструктури для побудови надійної гібридної архітектури та визначити можливі об’єкти аварійного відновлення;

— проаналізувати існуючі хмарні сервіси для аварійного відновлення даних і додатків;

— розробити метод автоматичного аварійного відновлення додатків у хмарному середовищі;

— провести розгортання в хмарному середовищі інфраструктури розробленого методу аварійного відновлення;

— провести тестування запропонованого методу.

Об`єктом дослідження є процес аварійного відновлення роботи додатків.

Предметом дослідження є підходи до аварійного відновлення в хмарних середовищах, а також засоби налаштування та управління хмарними ресурсами.

Наукова новизна роботи полягає у розробці гнучкого автоматичного методу аварійного відновлення додатків, в якому, на відміну від існуючих, застосовано комбінований підхід до резервування та аварійного відновлення вебзастосунку у середовищі Amazon Web Services (AWS) з гібридною архітектурою розгортання, що

дасть можливість підвищити стабільність та зменшити час недоступності додатків.

Практична цінність полягає у розробці архітектури методу аварійного відновлення додатків, розробці інфраструктури для розгортання методу аварійного відновлення вебдодатків у хмарному середовищі Amazon Web Services та в інтеграції сервісів AWS для забезпечення безперервної роботи системи у випадку збою.

Апробація результатів наукової роботи: зроблено доповідь на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» [1].

Публікації:

1. Метод аварійного відновлення вебдодатків у хмарному середовищі AWS / Д. В. Марценюк, О. В. Войцеховський, О. В. Войцеховська // Матеріали міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» (Вінниця, 2026 р.) [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/viewFile/26501/21841.

За результатами роботи подана стаття до наукового журналу «ВІСНИК Вінницького політехнічного інституту»

1 АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ АВАРІЙНОГО ВІДНОВЛЕННЯ ІНФОРМАЦІЙНОЇ ІНФРАСТУКТУРИ

У сучасному світі інформаційні технології стали невід’ємною частиною нашого життя, проникаючи в усі сфери діяльності — від навчання та роботи до розваг і комунікації. Мати швидкий доступ до потрібного сайту чи додатку — це необхідність, яка забезпечує ефективність і зручність у повсякденних завданнях. Проте, на жаль, навіть найсучасніші системи не захищені від аварійних ситуацій.

До найчастіших причин збою відносять несправність серверного обладнання, відключення світла, вихід з ладу систем зберігання даних, вилучення техніки перевірчими органами (наприклад, для проведення експертизи), крадіжки та диверсії, видалення критично важливих для компанії даних помилково (викликається рядом факторів, включаючи людський), пожежа, потоп чи інше стихійне лихо. Компанії, які зберігають резервні копії даних на одному майданчику з робочою версією, також не захищені від подібних інцидентів, адже при їх настанні разом із майданчиком будуть втрачені і резервні копії. У свою чергу, ручне резервування не забезпечує високої швидкості відновлення IT-інфраструктури та може бути надто трудомістким для системного адміністратора і сильно залежить від людського чинника [2].

1.1 Інформаційна інфраструктура

Інформаційна інфраструктура — це сукупність апаратних, програмних, мережевих і організаційних компонентів, які забезпечують обробку, зберігання, передачу і доступ до інформації в межах певної системи, організації або навіть на глобальному рівні. Вона є базою для функціонування інформаційних систем і включає різні рівні технологій та ресурсів [3].

Інформаційна інфраструктура складається з апаратного забезпечення, що включає сервери, робочі станції, комп'ютери, пристрої зберігання даних (жорсткі диски, сховища NAS, хмарні сервери), а також мережеве обладнання

(маршрутизатори, комутатори, точки доступу). Також інформаційна інфраструктура містить програмне забезпечення, яке охоплює операційні системи, бази даних, додатки, системи управління даними та моніторингові програми. Наступним ключовим компонентом є мережі, які поділяються на локальні (LAN) та глобальні (WAN), включаючи Інтернет, а також засоби їхнього захисту (брандмауери, VPN). Четвертим важливим елементом є хмарні сервіси, які надають хмарні платформи для зберігання даних та обчислень (наприклад, AWS, Azure, Google Cloud) та інструменти для резервного копіювання і відновлення даних. Також, до інфраструктури відносяться організаційні ресурси, що складаються з людського капіталу (адміністратори, розробники, користувачі) та регламентів і політик безпеки, які забезпечують правильне використання інформаційних ресурсів. Функції інформаційної інфраструктури зводяться до підтримки інформаційних систем для забезпечення їх стабільної роботи, обміну даними через швидку і безпечну передачу інформації, а також масштабованості, що надає можливість додавати нові ресурси чи змінювати їх параметри відповідно до зростання потреб організації.

Інформаційна інфраструктура є фундаментом для побудови сучасних систем обробки даних і цифрових сервісів. Вона також відіграє ключову роль у стратегічному плануванні технологічного розвитку організацій [4].

1.2 Огляд технологій розгортання інформаційної інфраструктури

Технології розгортання інформаційної інфраструктури – це сукупність методів, інструментів та процесів, які використовуються для швидкого, ефективного та масштабованого створення та налаштування IT-середовищ. Ці технології дозволяють автоматизувати рутинні завдання, знизити ймовірність помилок, підвищити надійність та гнучкість IT-систем [5].

Існують різні підходи до розгортання, які обираються залежно від складності проєкту та його вимог. Найпростішим є ручне розгортання, яке підходить для невеликих проєктів з унікальними вимогами, де автоматизація не є пріоритетом, хоча

й несе найбільші ризики помилок. Для середніх проєктів використовується скриптоване розгортання, яке забезпечує певний рівень автоматизації та повторюваності, що є проміжним кроком до повної автоматизації. Найсучаснішим і найбільш ефективним є підхід Інфраструктура як код (Infrastructure as Code – IaC), що є пріоритетним для великих і складних проєктів, де критично важливі швидкість, масштабованість та надійність. IaC перетворює конфігурацію інфраструктури з ручної операції на версіонований, повторюваний код, як і при розробці програмного забезпечення. Серед популярних інструментів IaC можна виділити Terraform, який підтримує широкий спектр хмарних провайдерів і дозволяє керувати різними середовищами з єдиного коду; Ansible – інструмент для автоматизації конфігурації, що не вимагає встановлення агента на кожен сервер, спрощуючи його впровадження; Puppet та Chef, які використовують відповідно маніфести та інструкції для опису бажаного стану системи.

Сучасні хмарні технології тісно пов'язані з розгортанням і поділяються на три основні моделі обслуговування, які визначають ступінь відповідальності провайдера та користувача, до них відносяться: IaaS (Infrastructure as a Service) — надання обчислювальних ресурсів, таких як сервери, мережі та сховища, у вигляді служб, що дає користувачеві максимальний контроль над операційною системою та додатками; PaaS (Platform as a Service) — надання платформи для розробки та запуску додатків, знімаючи з розробника необхідність керувати базовою інфраструктурою; SaaS (Software as a Service) — надання готових програмних продуктів через мережу, де користувач просто отримує доступ до функціонала.

Використання цих гнучких моделей, особливо інтеграція IaC із хмарним середовищем, забезпечує необхідні можливості для модернізації та оптимізації бізнес-процесів. Хмарні технології значно спрощують процес розгортання інфраструктури, дозволяючи компаніям швидко створювати і масштабувати ресурси за потреби, що робить їхню роль у сучасному IT-менеджменті незамінною.

1.2.1 Метод розгортання On-premise

On-premise (локальний) – це метод розгортання програмного забезпечення або програм, які встановлюються та функціонують на власних серверах та інфраструктурі компанії-замовника. Дане рішення відноситься до програмної чи технологічної інфраструктури, яка встановлюється та обслуговується на власних серверах чи устаткуванні компанії. Локальні рішення вважаються безпечнішими, оскільки компанія має повний контроль над інфраструктурою, але вони також можуть бути більш дорогими та трудомісткими в управлінні порівняно з хмарними чи вебрішеннями [6].

Однією з головних переваг даного рішення є контроль ресурсів. Завдяки локальним рішенням компанії повністю контролюють свої конфіденційні дані та інфраструктуру та можуть налаштовувати систему відповідно до своїх потреб. Наприклад, корпоративною інформацією, що передається в сторонніх месенджерах (таких як Telegram і WhatsApp), можуть скористатися конкуренти або зловмисники, тоді як дані в корпоративному месенджері – eXpress (при моделі розгортання on-premise) зберігаються локально на серверах без ризику витоку і передачі третій стороні. Друга важлива перевага — це безпека. Компанії, яким особливо важлива інформаційна безпека, найчастіше вибирають локальні рішення, тому що це можливість впроваджувати власні протоколи безпеки та заходи моніторингу для захисту корпоративної інформації. Адміністратори компанії відповідають за налаштування файрволів та антивірусного програмного забезпечення, управління доступом користувачів та розмежування їхніх прав, захист від кібератак. Ці та інші дії забезпечують безпеку локальної інфраструктури та конфіденційних даних та дозволяють не залежати від сумлінності сторонніх постачальників. Третя перевага — вартість у довгостроковій перспективі. У довгостроковій перспективі локальні рішення можуть бути рентабельнішими, оскільки не вимагають постійної абонентської плати (яка може бути суттєвою залежно від масштабів бізнесу). Компаніям потрібно лише заздалегідь інвестувати в обладнання та програмне

забезпечення, і вони можуть продовжувати їх використовувати протягом багатьох років.

Незважаючи на переваги, локальний метод розгортання має суттєві недоліки. Перш за все, це високі початкові витрати, оскільки такі рішення вимагають значних інвестицій у власне обладнання та інфраструктуру. Крім того, компанія повинна мати у штаті кваліфікований персонал, який зможе керувати цією інфраструктурою. Це прямо пов'язано з іншою проблемою – витратами на обслуговування та оновлення, адже компанії самостійно несуть відповідальність за підтримку та оновлення своїх локальних рішень, що постійно вимагає наявності підготовлених IT-спеціалістів. Ще один недолік – це обмежена гнучкість. Локальні рішення, як правило, не можуть бути легко адаптовані до потреб бізнесу, що швидко змінюються. Їх важко масштабувати, і для розширення потужностей організаціям потрібно додатково інвестувати у нове обладнання та інфраструктуру. Якщо ж система є георозподіленою, у кожному регіоні потрібна окрема команда для підтримки дата-центрів та швидкого вирішення можливих проблем.

1.2.2 Хмарна технологія розгортання

Хмарне рішення є програмним забезпеченням або послугою, що надається через інтернет або мережу віддалених серверів, а не з локального обладнання. Ця модель є фундаментально іншою, оскільки дані та програми розміщуються на віддалених потужностях провайдера, а не на власному комп'ютері користувача чи локальних серверах компанії. Таке рішення дозволяє компаніям отримувати доступ і використовувати програмне забезпечення або сервіс з будь-якого місця, де є підключення до інтернету, оскільки дані та програми розміщуються на віддалених серверах, а не на власному комп'ютері користувача або локальних серверах. Хмарні рішення часто забезпечують такі переваги, як масштабованість, доступність та економічність, оскільки вони не вимагають підтримки апаратної або програмної інфраструктури. Прикладами є такі популярні хмарні рішення, як Trello, Figma,

«Бітрікс24» та ін [7].

Основні переваги хмарних рішень ґрунтуються на доступності, тобто забезпечення доступу до даних та програм з будь-якої точки світу за наявності підключення до Інтернету. Ця географічна незалежність забезпечує значну гнучкість для підприємств та їх співробітників, що є особливо важливим для тих, хто працює віддалено або має філії в різних країнах. Друга перевага – це низька вартість початкових вкладень. Хмарні рішення часто рентабельніші, ніж традиційні локальні рішення, оскільки провайдери пропонують моделі ціноутворення з абонентською платою, які дозволяють підприємствам платити лише за фактично використані ресурси. При цьому відпадає необхідність одразу вкладати великі кошти в купівлю дорогої IT-інфраструктури та обладнання. Наступна перевага – масштабованість. Постачальник хмарних послуг може миттєво збільшувати або зменшувати обсяг ресурсів, що надаються, залежно від поточних потреб клієнта, що робить цей варіант ідеальним для невеликих компаній, які з часом планують розширити свою діяльність, або для бізнесів із сезонними піками навантаження.

Проте, хмарні технології мають і низку недоліків. Найбільш критичним питанням залишається безпека. Зберігання конфіденційних даних на сторонньому сервері провайдера становить потенційну загрозу для корпоративної безпеки. Хакер, який отримав несанкціонований доступ, може вкрасти конфіденційну інформацію, таку як персональні дані, інтелектуальну власність та фінансові відомості. Наступний недолік – це високі витрати в довгостроковій перспективі. Хмарні рішення часто надаються за моделлю передплати, і постачальник послуг може в односторонньому порядку збільшити абонентську плату, що ускладнює довгострокове бюджетування. Крім того, можливі додаткові витрати, такі як плата за міграцію даних або інтеграцію. Ще одним ризиком є втрата даних. Дані, що зберігаються у хмарі, можуть бути втрачені через збій обладнання, програмні помилки або навіть стихійні лиха на стороні провайдера. Такі інциденти можуть призвести до безповоротної втрати критичних даних, якщо клієнт не створив власні незалежні механізми резервного

копіювання та відновлення. Нарешті, існує залежність від постачальника хмарних послуг. Компанії, які використовують хмарні рішення, повністю залежать від провайдера щодо обслуговування, оновлень та клієнтської підтримки. Якщо провайдер припиняє свою діяльність або не може надавати якісні послуги, це може призвести до значних і непередбачуваних збоїв у роботі компанії.

1.3 Аналіз аварій в інформаційних системах та стратегій аварійного відновлення

Аварія в інформаційній системі – це подія, що призводить до часткової або повної втрати працездатності апаратних, програмних чи мережевих компонентів, у результаті чого користувачі не можуть отримати доступ до сервісів або даних. Аварії можуть мати як локальний характер (відмова одного сервера), так і глобальний (повна зупинка дата-центру чи всієї інфраструктури) [8].

Причини аварій в інформаційній інфраструктурі є різноманітними і поділяються на кілька категорій. Основна – це апаратні збої, які включають вихід з ладу жорстких дисків, контролерів пам’яті, блоків живлення, а також відмову мережевого обладнання, наприклад маршрутизаторів, комутаторів чи балансувальників навантаження. Також значний ризик несуть програмні помилки, які виникають як збої операційної системи, некоректна робота драйверів, або ж помилки у прикладному програмному забезпеченні, витоки пам’яті, чи некоректне використання ресурсів. Поширена причина — людський фактор. Сюди відносяться помилкові дії адміністраторів, наприклад, випадкове видалення даних, неправильна конфігурація системи, або ж несвоєчасне встановлення виправлень та оновлень. Також, аварії можуть бути спричинені зовнішніми чинниками, такими як перебої електропостачання, пожежі, затоплення, стихійні лиха та кібератаки – DDoS, шкідливе програмне забезпечення та компрометація облікових записів.

Внаслідок цих аварій можуть виникати серйозні несправності, до яких належать тимчасова недоступність сервісів, коли користувачі не можуть працювати

із системою, втрата або пошкодження даних у разі відсутності актуальних резервних копій, значні фінансові збитки через простої бізнес-процесів, невиконані транзакції та штрафи. Крім того, компанії стикаються з репутаційними ризиками та зниженням довіри клієнтів і партнерів.

Ці наслідки можуть варіюватися за серйозністю, адже масштаб аварії безпосередньо впливає на її вплив і завдані збитки. Залежно від цього, аварії класифікують на:

— локальні аварії — вихід з ладу окремого сервера чи сервісу (наприклад, бази даних або вебсервера);

— аварії рівня дата-центру — відмова всієї інфраструктури в одній географічній локації;

— глобальні аварії — збої, що охоплюють кілька дата-центрів або цілу регіональну інфраструктуру [9].

Саме тому, що повністю уникнути аварій неможливо, сучасні організації розробляють і впроваджують стратегії аварійного відновлення Disaster Recovery (DR). Вони дозволяють мінімізувати втрати даних і час простою, забезпечуючи безперервність виконання бізнес-вимог.

Ключовим показником, на який спрямовані всі зусилля DR, є мінімізація часу недоступності. Період, коли система, сервер, сайт чи додаток не працює належним чином або недоступний називають Downtime. Саме для ефективного управління цим показником розробляють Disaster Recovery Plan (DRP).

DRP — це комплексна стратегія, яка визначає порядок дій, необхідних для відновлення важливих IT-систем і даних після виникнення надзвичайних ситуацій, що можуть призвести до часткової або повної зупинки бізнес-процесів. Основна мета DRP полягає в мінімізації наслідків аварій, зменшенні простоїв, втрат даних та забезпеченні безперервності роботи організації [10].

Розробка Disaster Recovery Plan починається з аналізу ризиків і оцінки впливу інцидентів на бізнес (Business Impact Analysis). На цьому етапі визначаються критичні

ресурси — сервери, бази даних, програми, сервіси, а також встановлюються ключові параметри відновлення.

Після аналізу ризиків формується стратегія відновлення, яка визначає, де і як саме будуть відновлюватися системи. Це може бути локальне резервне середовище, хмарна інфраструктура або гібридна модель, що поєднує переваги обох підходів. У хмарних рішеннях.

Не менш важливим компонентом DRP є створення процедур резервного копіювання (backup). Резервні копії мають зберігатися у кількох місцях, включно з віддаленим середовищем, і регулярно перевірятися на працездатність. У сучасних системах застосовуються різні типи резервування — повне, інкрементальне, диференціальне, а також безперервне резервне копіювання у реальному часі.

Важливою частиною плану є документація та інструкції для персоналу. DRP має містити детальний опис усіх етапів відновлення: від виявлення інциденту та оповіщення відповідальних осіб до запуску резервної інфраструктури, перевірки доступності сервісів і повернення до нормальної роботи. Для злагоджених дій під час кризи визначаються ролі та відповідальність членів команди реагування. У великих організаціях такі плани часто доповнюються сценаріями комунікації із зовнішніми партнерами, клієнтами та ЗМІ.

Окреме місце у DRP займає тестування. Без регулярних перевірок ефективність плану залишається лише теоретичною. Тестування може проводитися у вигляді моделювання аварійних ситуацій, перевірки часу відновлення систем, тестів на перемикання на резервні сервери. Після кожного тесту складається звіт із висновками та рекомендаціями щодо вдосконалення процесів.

1.4 Аналіз існуючих хмарних рішень для аварійних відновлень

Аналіз хмарних рішень для аварійного відновлення проведемо на базі пропозицій від провідних провайдерів: Amazon Web Services (AWS) та Microsoft Azure.

В термінології хмарного провайдера AWS зони доступності (Availability Zones або Multi-AZ) є незалежними фізичними дата-центрами в межах одного географічного регіону. Ця концепція є ключовою для забезпечення високої відмовостійкості та безперервності роботи сервісів. Кожна така зона функціонує як повністю ізольована інфраструктурна одиниця: вона має власну енергетичну та мережеву інфраструктуру. Важливо, що зони доступності розташовані на достатній географічній відстані одна від одної, що дозволяє уникнути ризиків, пов'язаних із фізичними катастрофами, які могли б одночасно вивести з ладу всі центри.

Так розгортаючи інформаційну інфраструктуру в різних Availability Zone у рамках одного регіону можна підвищити відмовостійкість в разі аварійного відключення, якщо одна зона вийде з ладу, інші продовжать працювати. Багато сервісів хмарного провайдера AWS підтримують розгортання в декількох регіонах і забезпечують синхронізацію даних між ними.

Для забезпечення автоматичного переключення трафіку в разі недоступності зони та рівномірного його розподілу потрібно використовувати механізми моніторингу або балансування трафіку, наприклад Elastic Load Balancer (ELB).

ELB — це сервіс від AWS, який автоматично розподіляє вхідний трафік додатків між кількома інстансами або контейнерами в одному чи кількох Availability Zones[11].

Використання Availability Zones спрощує процес аварійного відновлення, але він змушує завжди тримати в робочому стані декілька інформаційних систем, як наслідок це призводить до значних витрат на обслуговування системи.

Також можна використовувати Auto Scaling Groups, які автоматично додають або видаляють екземпляри залежно від навантаження та стану системи. У поєднанні з ELB така архітектура забезпечує гнучке масштабування та стабільну роботу додатку навіть під час пікових навантажень або часткових відмов.

На рисунку 1.1 зображена можлива архітектура з розгортанням в декількох Availability Zones.

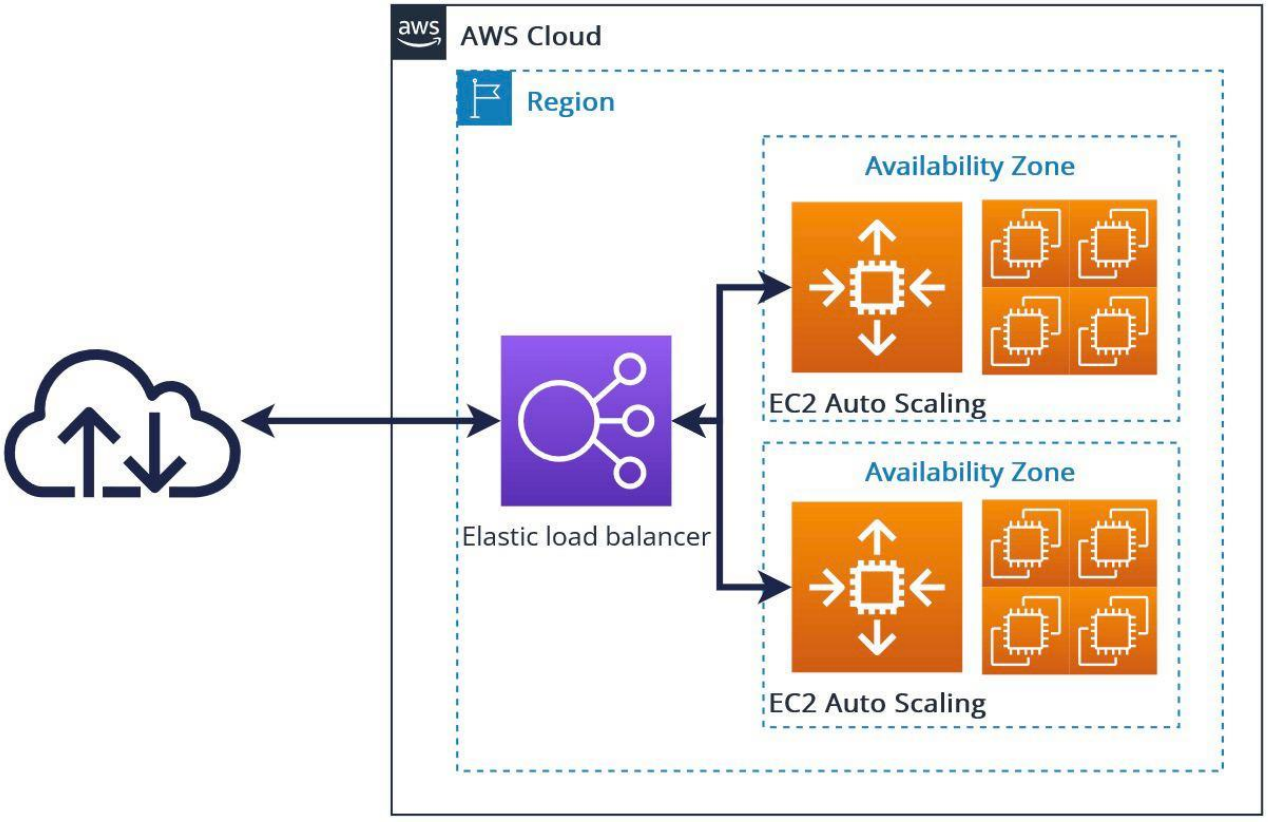

Рисунок 1.1 — Архітектура рішення AWS Availability Zones

Також існує сервіс еластичного відновлення після аварій, що має назву AWS Elastic Disaster Recovery (DRS), який характеризується тим, що є хмарним інструментом, розробленим для забезпечення відмовостійкості та швидкого відновлення після аварій інформаційних систем, незалежно від того, розгорнуті вони в хмарному середовищі чи на локальному сервері [12].

Цей сервіс працює за принципом безперервної реплікації. Він дозволяє створювати копії серверів (включно з екземплярами EC2, фізичними або віртуальними машинами) і постійно синхронізувати дані з копіями, які безпечно зберігаються в AWS. У разі аварії або катастрофи можливо швидко запускати ці копії на платформі AWS, тим самим відновлюючи повноцінну роботу систем без значних втрат часу або даних. Типове рішення з використанням архітектури Elastic Disaster Recovery зображено на рис. 1.2.

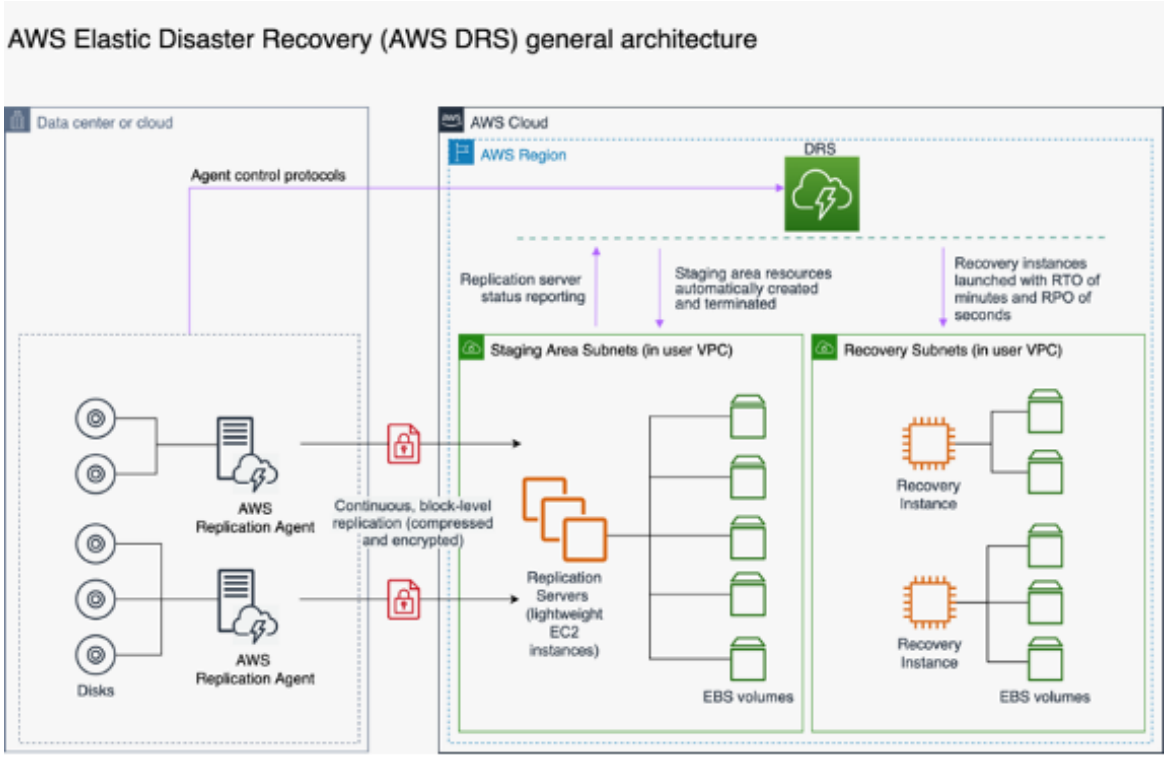

Рисунок 1.2 — Архітектура рішення Elastic Disaster Recovery

Варто зазначити, що аналогічне рішення пропонує і компанія Microsoft Azure, яке називається Azure Site Recovery (ASR). ASR є основним сервісом DRaaS від Azure і призначений для забезпечення безперервності бізнесу. На відміну від багатьох рішень, ASR працює як універсальний координатор, він дозволяє автоматизувати та оркеструвати реплікацію віртуальних машин та робочих навантажень. Його гнучкість дозволяє захищати різні середовища: як із локальних центрів обробки даних (ЦОД) у хмару Azure, так і між різними регіонами Azure. ASR забезпечує швидкий і надійний перехід у разі збою, а також підтримує створення комплексних планів відновлення, які дозволяють задати послідовність запуску сервісів для коректного відновлення багаторівневих додатків. Крім того, Azure Site Recovery тісно інтегрується з іншими сервісами екосистеми Microsoft, такими як Azure Monitor, Log Analytics та Azure Automation, що дозволяє централізовано керувати процесами резервування, моніторингу та відновлення. Типова архітектура використання сервісу ASR зображена на рис. 1.3.

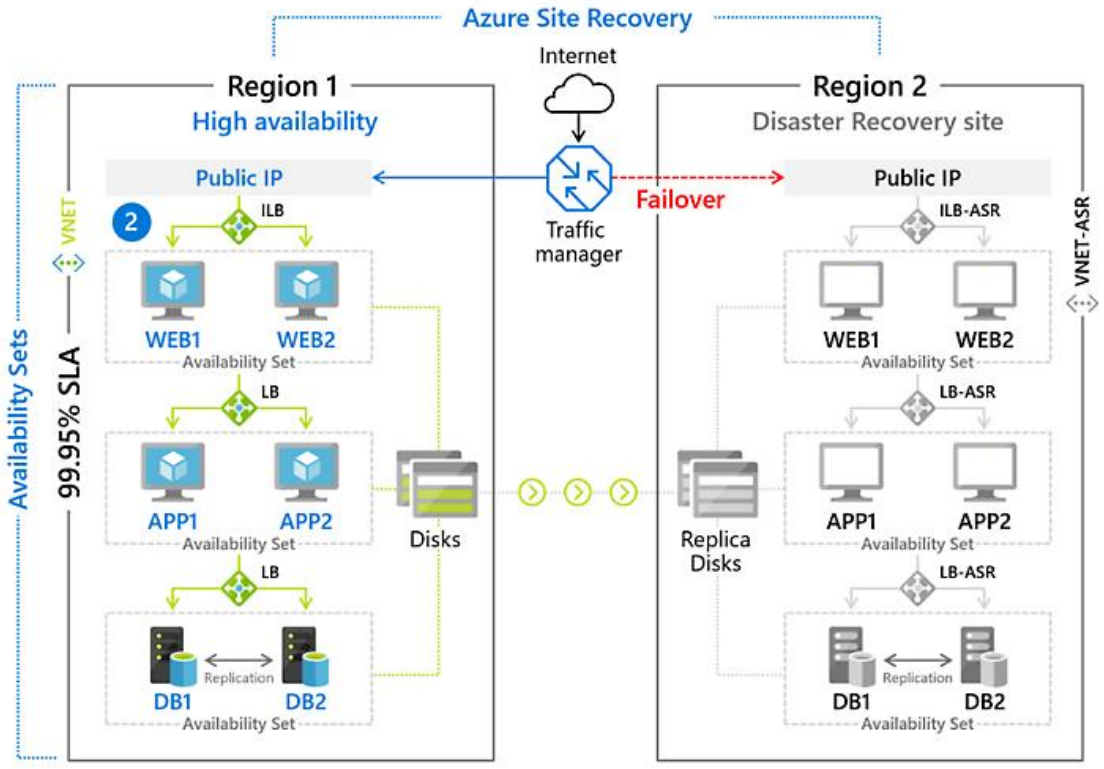

Рисунок 1.3 — Архітектура рішення ASR

Проте, варто враховувати, що DRS робить повну копію усієї серверної інфраструктури. З огляду на це, для оптимального використання даної технології рекомендується розгортати кожен додаток на окремому фізичному або віртуальному сервері. Ця технологія має свої недоліки, зокрема, вона навантажує мережу постійним копіюванням даних. Повне розгортання копії сервера може зменшувати гнучкість конфігурування гібридної інфраструктури, оскільки додає складності в управлінні та налаштуванні серверної частини. Це може призводити до збільшення витрат, а також потенційних проблем із розгортанням та доступом до ресурсів. Крім того, фізична копія сервера передбачає копіювання повного набору даних, що не завжди є необхідним при аварійному розгортанні, адже це може необґрунтовано збільшувати час підняття бази даних і спричиняти додаткові витрати на реплікацію та зберігання даних.

2 РОЗРОБКА МЕТОДУ АВАРІЙНОГО ВІДНОВЛЕННЯ ДОДАТКІВ У ХМАРНОМУ СЕРЕДОВИЩІ

2.1 Аналіз та обгрунтування вибору підходів для роботи з резервними базами даних при аварійному відновленні додатків

Основним завданням системи аварійного відновлення Disaster Recovery є забезпечення безперервності бізнес-процесів через мінімізацію часу простою та обсягу втрати даних у разі відмови основного середовища. Ефективність таких рішень оцінюється за двома ключовими метриками: Recovery Time Objective (RTO) та Recovery Point Objective (RPO).

RTO — це максимально допустимий час, протягом якого IT-система або сервіс може бути недоступним після збою. Іншими словами, це цільовий час відновлення, який визначає, як швидко бізнес повинен повернутися до нормальної роботи [13]. Низьке значення RTO (наприклад, лічені хвилини) вимагає значно більших інвестицій в інфраструктуру.

RPO (Recovery Point Objective) — це максимально допустимий обсяг втрати даних, виміряний у часі від моменту збою до останньої доступної точки відновлення. Це визначає, наскільки актуальними будуть дані після відновлення, тобто який обсяг роботи користувачів може бути втрачений [14]. Якщо RPO дорівнює нулю, це означає, що дані реплікуються практично миттєво.

Для досягнення мінімальних значень цих метрик розробляються різні стратегії, які, як правило, базуються на двох основних підходах до роботи з резервними базами даних: «гаряча» база даних та «холодна» база даних.

Метод «гарячої» бази даних (Hot Standby) є найбільш ефективною стратегією аварійного відновлення. Він базується на принципі безперервної синхронізації основної та резервної баз даних. Принцип роботи полягає в тому, що резервна база даних постійно підтримується в абсолютно актуальному стані, часто через технології реплікації (наприклад, асинхронна або синхронна реплікація) [15].

База даних перебуває в режимі постійного очікування, тобто є повністю функціональною і готовою до негайного переключення. У разі збою основної системи достатньо лише перенаправити трафік на резервну базу даних, оновивши DNS-записи або конфігурацію балансувальника. Завдяки цьому час переключення є мінімальним і зазвичай становить час T_{RTO} (базовий час відновлення), що робить метрику RTO надзвичайно низькою. Такий показник можливий через те, що резервна система вже працює і вимагає лише перенаправлення трафіку на рівні мережі, процес якого займає мінімальний час (T_{RPO}). При цьому обсяг втрати даних RPO майже нульовий або залежить виключно від затримки реплікації, що дозволяє фактично уникнути втрати даних.

Основними перевагами відновлення через «гарячу» базу даних є висока швидкість відновлення, мінімальні втрати даних та перехід є практично непомітним для кінцевих користувачів.

Однак, цей підхід має значний недолік: високу вартість, оскільки він вимагає підтримки двох повноцінних, повністю дубльованих екземплярів бази даних (БД), дублювання апаратних і програмних ресурсів, а також постійного моніторингу процесу реплікації.

На відміну від «гарячого» методу, підхід «холодної» бази даних (Cold Standby) є значно економічнішим, хоча при цьому передбачає більші показники RTO та RPO. Замість постійної реплікації, цей метод передбачає лише періодичний експорт критично важливих даних з основної БД та їх зберігання у доступному, але неактивному місці, наприклад, у хмарному сховищі S3 [16].

Для аварійної роботи системи достатньо лише відновлення обмеженого, критичного набору таблиць, таких як дані про користувачів, ролі, дисципліни, заняття та результати тестів. При цьому, аналітичні дані, журнали з історією змін або допоміжна інформація, які не є життєво необхідними для моментального запуску системи, не відновлюються одразу, що прискорює процес.

Переваги цього методу полягають у низькій вартості, оскільки він не потребує

постійно працюючої резервної інфраструктури; витрати виникають лише під час зберігання експортованих даних.

Однак, недоліки є наслідками економії, зокрема маємо значно вищий RTO (орієнтовно $3T_{RTO}$–$6T_{RTO}$ залежно від обсягу даних та швидкості запуску віртуальної машини), оскільки цей час визначається сукупністю неавтоматизованих кроків, таких як необхідність запуску віртуальних серверів, час на імпорт даних та початкове налаштування системи. RPO дорівнює інтервалу між експортами (орієнтовно 5 T_{RTO}), що означає можливу втрату даних, які були змінені в проміжку між останньою резервною копією та моментом збою.

Отже, можна зробити висновок, що вибір між «гарячим» і «холодним» підходом є класичним компромісом між вартістю та ефективністю. Організація може обрати більш економічний, але менш оперативний «холодний» варіант для некритичних систем, або інвестувати в швидший і надійніший «гарячий» метод для критичних бізнес-сервісів, де час простою може призвести до значних фінансових та репутаційних втрат.

2.2 Розробка математичної моделі методу аварійного відновлення додатків

Ефективність методів аварійного відновлення визначається двома ключовими показниками, які встановлюють бізнес-вимоги до швидкості та повноти відновлення: цільовий час відновлення (RTO) та цільова точка відновлення (RPO).

Максимально допустимий час , протягом якого система або додаток можуть бути недоступні після аварії RTO, визначає, скільки часу має пройти від моменту збою до повного відновлення працездатності сервісу [17]:

$$RTO = t_{restore} - t_{failure}, \tag{2.1}$$

де $t_{restore}$ — момент реєстрації відмови;

$t_{failure}$ – момент повного відновлення працездатності системи.

Якщо, наприклад, RTO дорівнює T_{RTO} одиницям часу, це означає, що після відмови інфраструктура повинна бути відновлена за цей час або швидше. Слід відмітити, що чим нижчий RTO, тим більш автоматизованим і дорогим буде рішення. Високі RTO (які можуть становити $10T_{RTO}$, або $100T_{RTO}$) допускають простіші й дешевші підходи (наприклад, відновлення з резервних копій вручну). Як правило, для критично важливих сервісів, таких як банківські системи, RTO може становити лише T_{RTO} одиниць часу, тоді як для внутрішньої аналітичної системи RTO може бути значно вищим.

Максимально допустима втрата даних, виміряна у часі RPO, показує, за який проміжок часу дані можуть бути втрачені без критичної шкоди для бізнесу:

$$RPO = t_{restore} - t_{last\_backup}\ , \tag{2.2}$$

де $t_{last\ backup}$ – момент останього бекапу.

Якщо, RPO дорівнює T_{RPO} одиницям часу, то у випадку аварії можна втратити дані, створені за останні T_{RPO} одиниць часу до збою. Чим менший RPO, тим частіше потрібно робити бекапи, або використовувати механізми безперервної реплікації. Це значно підвищує вартість інфраструктури. Навпаки, великий RPO (години чи дні) дозволяє дешевші рішення, але підвищує ризик втрати даних.

Для критично важливих систем, таких як платіжні системи, RPO має бути близьким до нуля (не можна втрачати транзакції). Для менш критичних або архівних систем RPO може бути значно вищим (наприклад, $24T_{RPO}$, що відповідає оновленню даних раз на добу).

Параметр RTO дає відповідь на питання, як швидко повинна відновитись система. В свою чергу параметр RPO показує скільки даних система може втратити внаслідок аварійного відновлення. Разом вони визначають рівень готовності системи до аварій і прямо впливають на вибір технологій (реплікація, бекапи, кластеризація, хмарні сервіси, тощо) та вартість рішення.

Загальний час відновлення системи (TRT) – це сума часу виявлення збою Failure Detection Time (FDT) та часу відновлення RTO, що визначає повний час, необхідний для відновлення після збою:

$$TRT = \quad FDT + RTO, \tag{2.3}$$

де FDT – час, який потрібен для виявлення збою.

FDT визначає інтервал між моментом фактичної відмови сервера та моментом, коли механізм контролю стану зафіксував, що сервіс став непрацездатним. Мінімізація FDT є важливою, оскільки вона безпосередньо прискорює запуск процесу відновлення [18]:

$$FDT = t_{detected} - t_{crash}\ , \tag{2.4}$$

де $t_{detected}$ – момент виявлення збою системою;

t_{crash} – момент фактичної відмови або виникнення збою в системі/сервері;

На основі отриманого значення TRT можна оцінити рівень доступності системи за моделлю Service Level Agreement (SLA).

Доступність SLA є одним із найважливіших показників якості роботи сервісів і безпосередньо визначає рівень надійності інфраструктури. Вона вимірюється у відсотках часу, протягом якого система залишається працездатною та доступною для користувачів. Наприклад, рівень доступності 99.9% (три дев’ятки) означає максимум близько 8,7 годин простою на рік, 99.99% (чотири дев’ятки) — близько 52 хвилин, а 99.999% (п’ять дев’яток) — лише близько 5 хвилин на рік.

Такі показники є стандартом для критично важливих сервісів, оскільки навіть короткочасний збій може призвести до значних фінансових втрат або зниження довіри користувачів:

$$SLA = 1 - \frac{TRT}{t_{total}}, \quad (2.5)$$

де t_{total} – загальний період спостереження (наприклад, доба, місяць, рік).

Для «холодного» методу аварійного відновлення, коли резервна інфраструктура створюється з нуля після збою, цільовий час відновлення $\mathrm{RTO}_{\mathrm{cold}}$ можна представити у вигляді суми окремих часових компонентів:

$$RTO_{cold} = t_{import} + t_{ASG\_start} + t_{DNS\_update}\ , \quad (2.6)$$

де t_{import} – час, необхідний для імпорту або розгортання даних із резервних копій;

t_{ASG_start} – час запуску груп серверів або ініціалізації обчислювальних ресурсів;

t_{DNS_update} – час оновлення записів DNS, необхідний для маршрутизації трафіку на нове середовище.

Таким чином, $\mathrm{RTO}_{\mathrm{cold}}$ показує сумарний час, необхідний для повного відновлення працездатності системи в умовах відсутності попередньо розгорнутої інфраструктури.

Відповідно, для цього ж сценарію цільова точка відновлення RPO визначається інтервалом між останнім створеним резервним копіюванням та моментом збою:

$$PRO_{cold} \leq T_{backup\_interval}\ , \quad (2.7)$$

де $T_{backup\_interval}$ – інтервал між створенням резервних копій.

Це означає, що максимальні потенційні втрати даних у «холодному» методі не перевищують часу між двома послідовними бекапами.

У модифікації методу аварійного відновлення з використанням «гарячої» бази даних резервна інфраструктура функціонує постійно, синхронізуючись у реальному часі з основною системою. У цьому випадку час відновлення RTO_{hot}, визначається

лише мінімальними затримками, пов’язаними з перемиканням трафіку та активацією резервних компонентів:

$$RTO_{hot} = t_{ASC\_start} + t_{DNS\_update}\,. \quad (2.8)$$

Оскільки резервне середовище вже готове до роботи, цільовий час відновлення у «гарячому» методі практично наближається до нуля:

$$RTO_{hot} \approx 0. \quad (2.9)$$

Таким чином, «гарячий» метод забезпечує максимально швидке відновлення сервісів після збою, але потребує значних витрат на підтримку постійно активної інфраструктури.

Для зменшення часу Recovery Time Objective потрібно забезпечити мінімальний час відновлення бази даних. Цього можна досягнути шляхом тримання «гарячої» бази з усіма актуальними даними, але цей метод потребує значних ресурсів на обслуговування двох екземплярів бази даних. Інший варіант зменшення часу відновлення це відновлення часткового набору даних. Для аварійної роботи системи не потрібна наявність історичних даних, аналітики і іншого не критичного фунціоналу. Для цього методу використаємо часткове відновлення даних з файлів експорту свіжих даних критичних таблиць бази даних.

Також для зменшення показника RTO потрібно використовувати автоматичні системи перевірки працездатності сервера, які зможуть в автоматичному режимі перевіряти стан серверу і генерувати відповідно події в разі аварії на сервісі які запустять процес аварійного перенесення інфраструктури.

2.3 Проєктування архітектури автоматичного відновлення додатків

Для реалізації методу аварійного відновлення використовується інтегрований

підхід на базі сервісів Amazon Web Services, який забезпечує автоматичне виявлення відмов та запуск процесу відновлення без втручання користувача.

Архітектура складається з чотирьох компонентів [19].

Компонент Amazon Route 53 здійснює моніторинг доступності локального сервера за допомогою health checks. У випадку, коли сервер не відповідає, стан змінюється зі Healthy на UnHealthy.

Компонент Amazon CloudWatch збирає та аналізує метрики доступності, отримані від Route 53. CloudWatch слугує системою оповіщення. При зміні стану (з Healthy на UnHealthy) він генерує подію, яка сигналізує про необхідність активації алгоритму відновлення.

Компонент Amazon SNS виконує роль сервісу доставки повідомлень (message broker), що забезпечує асинхронну та гарантовану доставку інформації про інцидент до всіх підписників.

Компонент AWS Lambda є обчислювальним ядром системи, вона виконує алгоритм відновлення. Функція Lambda містить логіку, яка адаптується під обрану стратегію відновлення. В базованому оптимальному алгоритм створює новий екземпляр бази даних, відновлює критичні таблиці з CSV у S3, після чого запускає Auto Scaling Group для підняття EC2-екземплярів. Також можна модифікувати Lambda-функцію для адаптації методу до «гарячого» сценарію та одразу активовувати Auto Scaling Group, оскільки база даних буде перебувати у синхронізованому стані.

На першому етапі Route 53 виконує моніторинг доступності сервісу за допомогою механізму health checks. Система здійснює регулярні DNS-запити до локального сервера. У випадку, коли сервер не відповідає, стан ресурсу автоматично змінюється зі Healthy на Unhealthy. Цей стан відображає недоступність основного середовища. Далі у процес вбудовується Amazon Cloud Watch, який збирає та аналізує метрики доступності. CloudWatch має можливість налаштовувати правила реагування на зміну стану моніторингу. При виявленні збою він генерує подію, що сигналізує про

аварію.

Згенерована подія передається у Amazon Simple Notification Service (SNS). Даний сервіс використовується як брокер повідомлень, що забезпечує надійну доставку інформації про інцидент до всіх зареєстрованих підписників. У нашому випадку підписником виступає функція AWS Lambda, яка виконує віддалену обробку подій.

На заключному етапі функція AWS Lambda ініціює алгоритм аварійного відновлення. Вона виконує декілька ключових дій, спрямованих на відновлення працездатності системи. Спочатку Lambda створює новий екземпляр бази даних, використовуючи останню доступну резервну копію, збережену у сховищі Amazon S3. Після цього вона ініціалізує групу автоматичного масштабування EC2 Auto Scaling Group, яка розгортає необхідну кількість віртуальних серверів для запуску основного застосунку. Завершальними кроками є додаткові налаштування, які можуть включати оновлення записів DNS, перевірку працездатності відновленої системи або запуск тестових запитів.

Таким чином, у випадку збою локального сервера, система автоматично перемикає навантаження на резервну інфраструктуру в AWS. Це забезпечує мінімальний час простою сервісу та дозволяє досягати мінімальних показників RTO та RPO. Схема архітектури представлена на рисунку В.1.

Оскільки вимоги до аварійного відновлення можуть бути різними і змінюватись з часом, розроблений метод повинен бути гнучким для зменшення показників RTO.

Для забезпечення зменшення RTO розробимо варіант модифікації методу з використання сервісу AWS Database Migration Service. На відміну від розробленого методу, який передбачає відновлення з CSV-бекапів, цей підхід дозволяє тримати резервну базу даних у постійному робочому стані та синхронізувати з нею критичні таблиці у режимі реального часу або з мінімальною затримкою.

Основна ідея полягає у створенні додаткового екземпляра бази даних, який функціонує у хмарному середовищі AWS. Data Migration Service забезпечує міграцію

та реплікацію даних з основної бази на резервну, виконуючи як початкове повне завантаження таблиць (Full Load), так і подальший виявлення та застосування змін (Change Data Capture, CDC). Такий підхід дозволяє у разі аварії миттєво переключитися на резервну базу без необхідності відновлення даних з файлів. Ця оновлена архітектура, що мінімізує час відновлення, представлена на рисунку Г.1.

Використання AWS DMS для реплікації даних передбачає виконання кількох ключових етапів. Процес починається зі створення Replication Instance, який є обчислювальним ресурсом, що відповідає за сам процес копіювання даних. Далі необхідно налаштувати Source Endpoint, що вказує на основну базу даних, та Target Endpoint, який позначає резервну базу даних у хмарному середовищі (наприклад, RDS або Aurora). Після визначення точок входу та виходу даних, виконується визначення Table Mapping. Цей етап дозволяє налаштувати реплікацію лише критичних таблиць, необхідних для забезпечення мінімальної працездатності системи, ігноруючи другорядні дані. В кінці обирається режим роботи завдання: Full Load або Full Load + CDC для забезпечення безперервної актуалізації даних.

Застосування AWS DMS надає кілька ключових переваг для забезпечення високої доступності. Він забезпечує значно менший показник RPO, оскільки дані реплікуються практично безперервно у режимі Change Data Capture. Також, цей сервіс усуває потребу у ручному експорті та імпорті таблиць, автоматизуючи процес синхронізації. Використання DMS дозволяє знизити час повного відновлення системи, оскільки резервна база даних уже знаходиться у робочому та актуальному стані, готова до негайного переключення [20].

Разом з тим, даний метод має і певні обмеження: він потребує постійного функціонування додаткової бази даних, що збільшує вартість рішень, а також вимагає більш ретельного налаштування та контролю за процесом реплікації.

Таким чином, використання AWS DMS можна розглядати як більш продуктивний варіант аварійного відновлення у порівнянні з періодичним експортом у S3, що дозволяє досягати майже нульового значення RPO але при значно вищих

витратах на інфраструктуру.

Отже, розроблений метод аварійного відновлення в AWS є досить гнучким і легко модифікується під різні бізнес вимоги. Вибір варіанту модифікації залежить від критичності даних та бюджетних обмежень. Якщо пріоритетом є мінімізація втрати даних (низький RPO) і час простою (низький RTO), то використання AWS DMS є кращим, хоч і дорожчим, рішенням. Натомість, якщо економічна ефективність важливіша, а незначні втрати даних та час на відновлення є прийнятними, то підхід з відновленням з S3 є цілком достатнім.

2.4 Розробка алгоритму роботи методу автоматичного аварійного відновлення додатків

Для розробки стратегії аварійного відновлення, важливим є не лише опис архітектурних компонентів, але й деталізація алгоритму її функціонування. Алгоритм функціонування сервісів Amazon Web Services для забезпечення безперервної роботи системи у випадку збою представлено на рисунку Д.1.

Опишемо наведений алгоритм роботи Amazon Web Services для забезпечення безперервної роботи системи у випадку збою.

На першому етапі система моніторингу Route 53 Health Check постійно перевіряє стан основного сервера або застосунку. У разі виявлення недоступності ресурсу (наприклад, відсутності відповіді на HTTP-запит або перевищення часу очікування) сервіс фіксує відмову. Це є початком процесу аварійного відновлення.

Після фіксації збою система Amazon CloudWatch створює подію (сигнал), що містить інформацію про тип інциденту, час ідентифікації відмови та відповідний ресурс. Ця подія передається до Amazon SNS (Simple Notification Service) для подальшої обробки.

Наступний кроком сервіс SNS активує спеціально налаштовану AWS Lambda-функцію, яка є центральним елементом механізму автоматичного відновлення.

Далі створюється новий екземпляр бази даних у сервісі Amazon RDS. Цей

процес включає підготовку середовища, налаштування параметрів доступу, безпеки та виділення необхідних ресурсів для відновлення даних.

Імпорт CSV-резервних копій у новий екземпляр бази даних відбувається на кроці 5. Після створення резервної бази даних система автоматично завантажує останні резервні копії із сховища (наприклад, Amazon S3) і виконує їх імпорт у новий екземпляр бази даних. Це забезпечує відновлення актуальних даних, доступних на момент останнього резервного копіювання, тим самим визначаючи значення параметра RPO.

На кроці 6 (Запуск аварійного додатку) система ініціює Auto Scaling Group (ASG), яка автоматично створює і запускає екземпляри додатку в аварійному середовищі. Це дозволяє швидко розгорнути робочу копію сервісу без ручного втручання адміністратора.

Після запуску резервного середовища на кроці 7 «Оновлення DNS-записів» Lambda ініціює оновлення Amazon Route 53 DNS-записів для маршрутизації користувацьких запитів на нову інфраструктуру.

На фінальному етапі користувачі отримують доступ до аварійного середовища, яке забезпечує безперервність бізнес-процесів. Подальша робота може тривати до моменту відновлення основної інфраструктури, після чого відбувається зворотне перемикання.

При модифікації методу з використанням «гарячої» бази даних після запуску Lamba функції відбувається перехід відразу до пункту 6, пропускаючи етапи розгортання нової бази даних. Оновлений алгоритм модифікації методу представлено на рисунку Е.1.

Запропонований метод аварійного відновлення має низку переваг порівняно з існуючими готовими механізмами AWS, такими як розгортання в різних Availability Zones та використання сервісу Elastic Disaster Recovery. По-перше, він орієнтований на вибіркове резервування критичних даних, що дозволяє зменшити обсяг копій і скоротити час відновлення, тоді як стандартні рішення забезпечують повну

синхронізацію всієї бази. По-друге, підхід реалізує автоматизоване розгортання через описані скрипти, що дає можливість гнучко налаштовувати алгоритм відновлення залежно від конкретних вимог до RTO та RPO, тоді як готові сервіси є більш жорстко фіксованими. По-третє, реалізація не потребує постійного утримання інфраструктури у режимі високої готовності на всіх рівнях, що дозволяє знизити вартість експлуатації у порівнянні з Multi-AZ або Multi-Region, де резервні ресурси функціонують безперервно. Таким чином, метод поєднує прийнятний рівень надійності з гнучкістю конфігурації та економічною доцільністю, що робить його практичним для систем з обмеженими ресурсами або для освітніх і дослідницьких проєктів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДУ АВАРІЙНОГО ВІДНОВЛЕННЯ

Для відтворення робочого середовище, яке буде імітувати локальний сервер було застосовано розгортання в окреме середовище AWS. Такий підхід дозволяє змоделювати реальні умови експлуатації інформаційної системи та відпрацювати сценарії відмови й подальшого відновлення. Також це рішення обране через його гнучкість, високу доступність сервісів, широкий спектр інструментів для автоматизації та можливість інтеграції з системами моніторингу й аварійного відновлення [21].

Склад елементів локального сервера, що підлягає емулюванню, включає два ключові компоненти: прикладний сервіс та система управління базами даних.

Прикладний сервіс — це вебзастосунок, розроблений на платформі .NET Core MVC, який виконує роль бізнес-логіки та забезпечує інтерфейс користувача.

Система управління базами даних (СУБД) — PostgreSQL, яка зберігає основні дані та забезпечує транзакційну цілісність. Обидва ці компоненти розгорнуті у хмарному середовищі AWS, що дозволяє сформувати архітектуру, максимально наближену до розгортання на локальному сервері. Крім того, хмарне середовище надає можливість гнучко налаштовувати параметри інфраструктури (зокрема, обчислювальні ресурси, мережеві конфігурації, політики доступу) та забезпечує можливість швидкого масштабування для моделювання різних сценаріїв навантаження та відмов. Використання PostgreSQL у хмарній інфраструктурі дає змогу реалізувати гнучкі механізми резервного копіювання та відновлення даних, а також налаштувати реплікацію між інстансами для підвищення продуктивності під час високих навантажень завдяки балансування навантаження між вузлами,

Такий підхід також спрощує проведення технічного обслуговування — оновлення або тестування можна виконувати на копіях бази без зупинки основного середовища.

Таким чином, розгортання .NET Core MVC сервісу та PostgreSQL бази даних у AWS виконує роль застосунку, який розгорнуто на «локальному сервері», і дозволяє у подальшому дослідити механізми аварійного відновлення. Це рішення забезпечить контрольоване середовище для експериментів, де можна відтворювати як часткові, так і повні збої з подальшим аналізом показників RTO та RPO.

На рисунку 3.1 зображено схему розгортання вебдодатку в середовищі AWS Elastic Beanstalk.

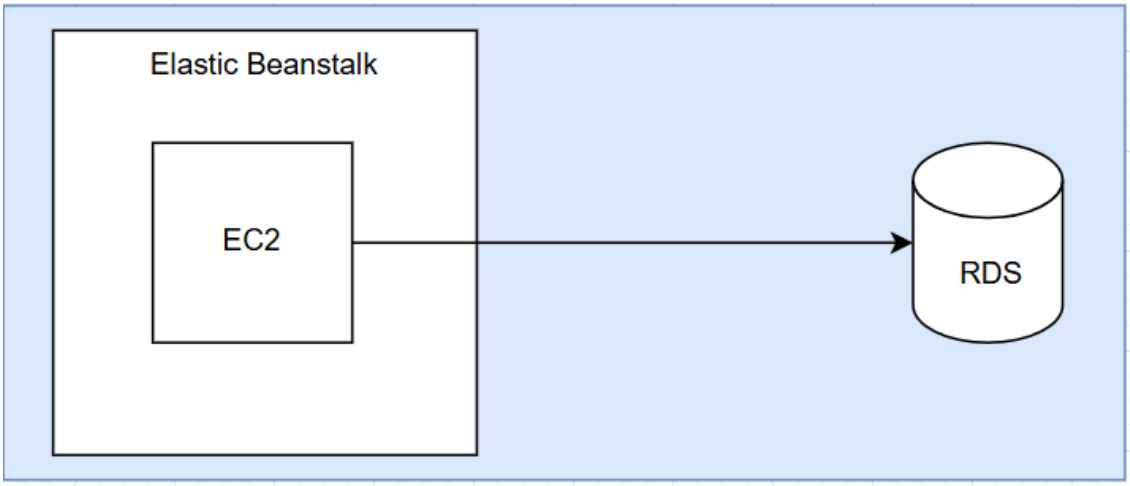

Рисунок 3.1 — Схема розгортання додатку

Усередині середовища працює екземпляр EC2, на якому розміщено додаток. Додаток підключається до керованої бази даних RDS для збереження та обробки даних. Elastic Beanstalk забезпечує автоматичне керування інфраструктурою, масштабування та моніторинг середовища

3.1 Розгортання бази даних в середовищі AWS

Для імітації роботи локального сервера бази даних було використано сервіс Amazon Relational Database Service (RDS). RDS — це керований сервіс реляційних баз даних від AWS, який значно спрощує розгортання, адміністрування та масштабування СУБД у хмарі [22].

Його вибір обумовлений кількома перевагами: RDS бере на себе рутинні

завдання, такі як оновлення операційної системи та оновлення версій бази даних, звільняючи користувача від необхідності управління інфраструктурою. Також, він забезпечує автоматичне резервне копіювання, створюючи щоденні знімки даних та записів транзакцій, що критично важливо для забезпечення відновлення даних Point-In-Time Recovery та аварійного відновлення [23].

RDS дозволяє легко масштабувати обчислювальні ресурси та дисковий простір, імітуючи гнучкість, необхідну для моделювання різних навантажень. Завдяки функції Multi-AZ (Multi-Availability Zone), RDS може автоматично створити синхронну репліку в іншій фізично ізольованій зоні доступності, забезпечуючи високу відмовостійкість. Крім того, сервіс має вбудований моніторинг продуктивності через інтеграцію з Amazon CloudWatch та простоту інтеграції з іншими ключовими сервісами AWS, що використовуються в архітектурі аварійного відновлення. Таким чином, RDS є надійною основою для імітації СУБД PostgreSQL у хмарі.

Розгортання бази даних виконувалося за допомогою інструменту Terraform, що дозволяє описувати інфраструктуру як код та забезпечує відтворюваність середовища.

Terraform — це провідний платформонезалежний інструмент з відкритим вихідним кодом, розроблений компанією HashiCorp, який повністю реалізує концепцію інфраструктура як код [24].

Далі необхідно створити групу безпеки (лістинг 3.1) з ідентифікатором rds_pg_public яка буде використовуватись для контролю мережевого доступу до бази даних PostgreSQL, що розміщена в службі Amazon RDS. Параметр name задає назву групи безпеки, яка спрощує її ідентифікацію серед інших ресурсів.

Поле description" містить опис, що пояснює призначення ресурсу — дозвіл на публічний доступ до сервера PostgreSQL. Параметр vpc_id вказує, у якому саме віртуальному приватному кластері (VPC — Virtual Private Cloud) створюється ця група безпеки. Це забезпечує логічну ізоляцію мережевих ресурсів у межах AWS.

Така конфігурація є основою для подальшого визначення правил Ingress і

Egress, які регулюють, з яких джерел можна підключатися до бази даних і куди дозволено надсилати трафік.

Лістинг 3.1 — Створення віртуального екрану

```
resource "aws_security_group" "rds_pg_public"
{
        name        = "rds-pg-public"
        description = "Allow Postgres public access"
        vpc_id      = "vpc-0def8356f6632c9ad" # default VPC
}
```

Опишемо Ingress-правило в межах Security Group, яке визначає дозволений вхідний трафік до сервера бази даних PostgreSQL. Опис правила продемонстровано в лістингу 3.2

Лістинг 3.2 — Створення Ingress-правила в межах Security Group

```
ingress
{
        description = "Postgres"
        from_port   = 5432
        to_port     = 5432
        protocol    = "tcp"
        cidr_blocks = [var.allowed_cidr]
}
```

Далі створюємо правило Egress (лістинг 3.3) яке визначає політику вихідного трафіку в межах групи безпек. Воно дозволяє надсилати дані в будь-якому напрямку без обмежень. Параметри from_port = 0 і to_port = 0 означають, що правило застосовується до всіх портів, а protocol = "-1" вказує, що дозволено використання всіх мережевих протоколів (TCP, UDP, ICMP тощо). Значення cidr_blocks = ["0.0.0.0/0"] відкриває можливість надсилати трафік до будь-якої IP-адреси. Таке налаштування зазвичай використовується для забезпечення повної відкритості вихідного трафіку, при цьому безпека залишається на належному рівні завдяки суворим обмеженням на вхідний трафік (Ingress).

Лістинг 3.3 — Створення Egress правила

```
egress {
from_port   = 0
to_port     = 0
protocol    = "-1"
cidr_blocks = ["0.0.0.0/0"]}}
```

Приклад конфігурування керованого екземпляра бази даних PostgreSQL в сервісі AWS RDS. Приклад такої конфігруації подано в лістингу 3.4

Лістинг 3.4 — Створення екземпляра бази даних PostgreSQL в AWS RDS

```
resource "aws_db_instance" "pg_public" {
identifier            = "vntu-local-db"
engine                = "postgres"
instance_class        = "db.t4g.micro"    # Free Tier
allocated_storage     = 20
storage_type          = "gp3"
db_name               = "vntu_local_db"
username              = "masteruser"
password              = "admin123!"
db_subnet_group_name   = "default-vpc-0def8356f6632c9ad"
vpc_security_group_ids = [aws_security_group.rds_pg_public.id]
multi_az              = false
deletion_protection    = false
skip_final_snapshot    = true
backup_retention_period = 7
auto_minor_version_upgrade   = true
performance_insights_enabled = true
tags = { Name = "vntu-local-db" }}
```

У даній конфігурації передбачено:

— створення групи безпеки (Security Group), яка дозволяє підключення до порту 5432 лише з визначеного діапазону IP-адрес;

— розгортання екземпляру PostgreSQL класу db.t4g.micro (рівень Free Tier), з виділенням 20 GB дискового простору;

— увімкнення механізмів автоматичного бекапу та Performance Insights для моніторингу продуктивності;

— забезпечення доступності бази ззовні для подальшого використання її як «локального» сервера у алгоритмі аварійного відновлення.

Після виконання даного скрипта у Terraform у вебконсолі AWS можна перевірити створений ресурс (рис. 3.2), переконатися в його працездатності та протестувати підключення до бази даних за допомогою клієнтів PostgreSQL.

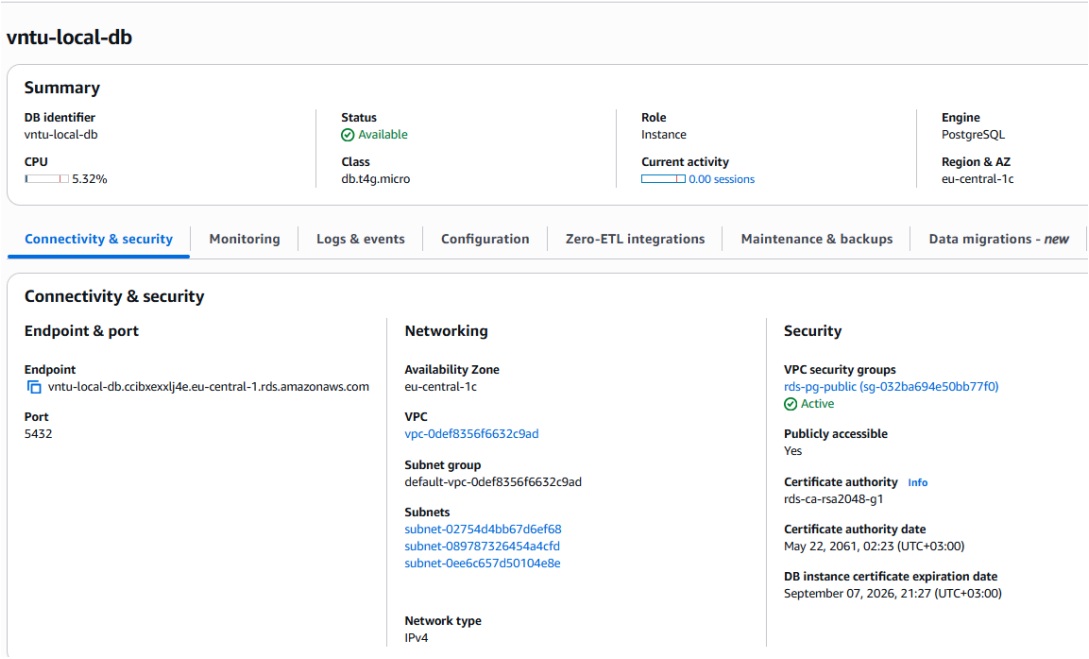

Рисунок 3.2 — Результат розгортання екземпляру PostgreSQL у вебконсолі AWS

3.2 Розгортання вебдодатку .NET Core MVC в середовищі AWS

Для імітації роботи прикладного сервісу на локальному сервері було обрано сервіс AWS Elastic Beanstalk, який дозволяє швидко розгортати та керувати вебдодатками без необхідності глибокого адміністрування інфраструктури. Elastic Beanstalk автоматично забезпечує створення необхідних ресурсів (EC2-екземплярів, балансувальників навантаження, груп безпеки, моніторинг CloudWatch), що значно спрощує процес моделювання «локального» середовища у хмарі [25].

Крім того, сервіс підтримує автоматичне масштабування залежно від навантаження та дозволяє швидко відновлювати середовище у разі збоїв, що є важливим аспектом при дослідженні Recovery Time Objective.

Приклад публікації додатку за допомогою AWS Elastic Beanstalk продемонстровано на рис. 3.3.

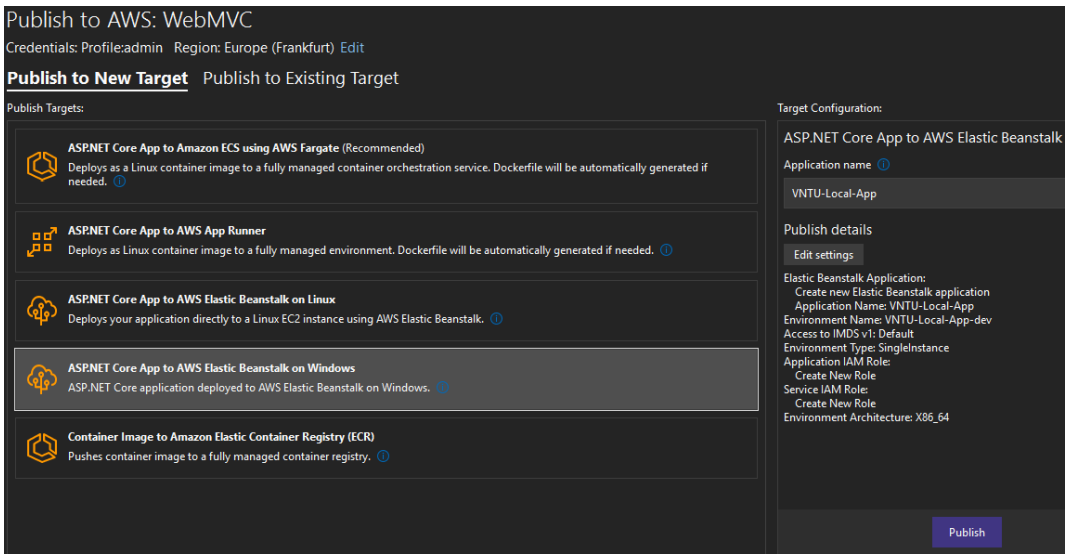

Рисунок 3.3 — Публікація додатку за допомогою AWS Elastic Beanstalk

Прикладний сервіс реалізований у вигляді .NET Core MVC вебдодатку. Розгортання виконувалося за допомогою інтеграції середовища розробки Microsoft Visual Studio з інструментарієм Amazon SDK for .NET, який містить плагіни для публікації застосунків у хмару.

Послідовність розгортання прикладного сервісу в хмарі починається з підготовки застосунку у Visual Studio, що охоплює конфігурування середовища запуску та створення профілю публікації. Після цього здійснюється вибір цільової платформи — Elastic Beanstalk, який є повністю керованим сервісом AWS із нативною підтримкою .NET Core.

Elastic Beanstalk автоматично бере на себе створення необхідного хмарного середовища: він розгортає один або кілька екземплярів EC2 для виконання вебдодатку, налаштовує балансувальник навантаження (за потреби), формує групи безпеки для регулювання вхідного/вихідного трафіку та забезпечує інтеграцію з CloudWatch для збору метрик продуктивності та записів. Далі відбувається завантаження пакета застосунку у середовище AWS, як правило, у формі ZIP-

артефакту, що містить скомпільований .NET Core MVC проєкт. На завершальному етапі, Elastic Beanstalk виконує автоматичне розгортання та запуск.

На рис. 3.4 наведено результат успішної публікації вебзастосунку у середовищі AWS Elastic Beanstalk. Система автоматично створила необхідні ресурси та надала посилання для доступу до застосунку.

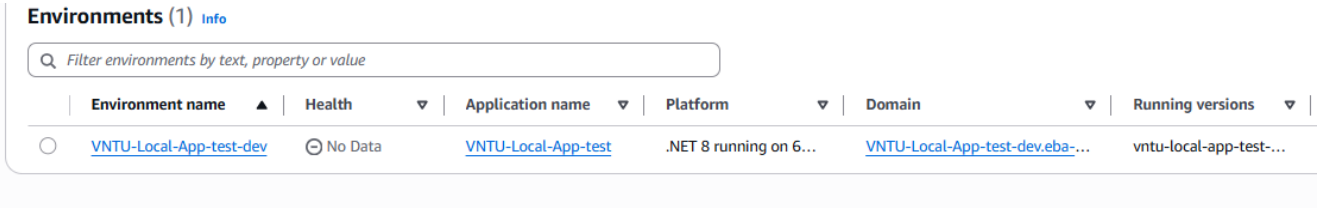

Рисунок 3.4 — Результат публікації вебзастосунку у середовищі AWS Elastic Beanstalk

На рис. 3.5 показано деталі публікації, де відображено інформацію про середовище Elastic Beanstalk, параметри розгортання та стан виконання додатку.

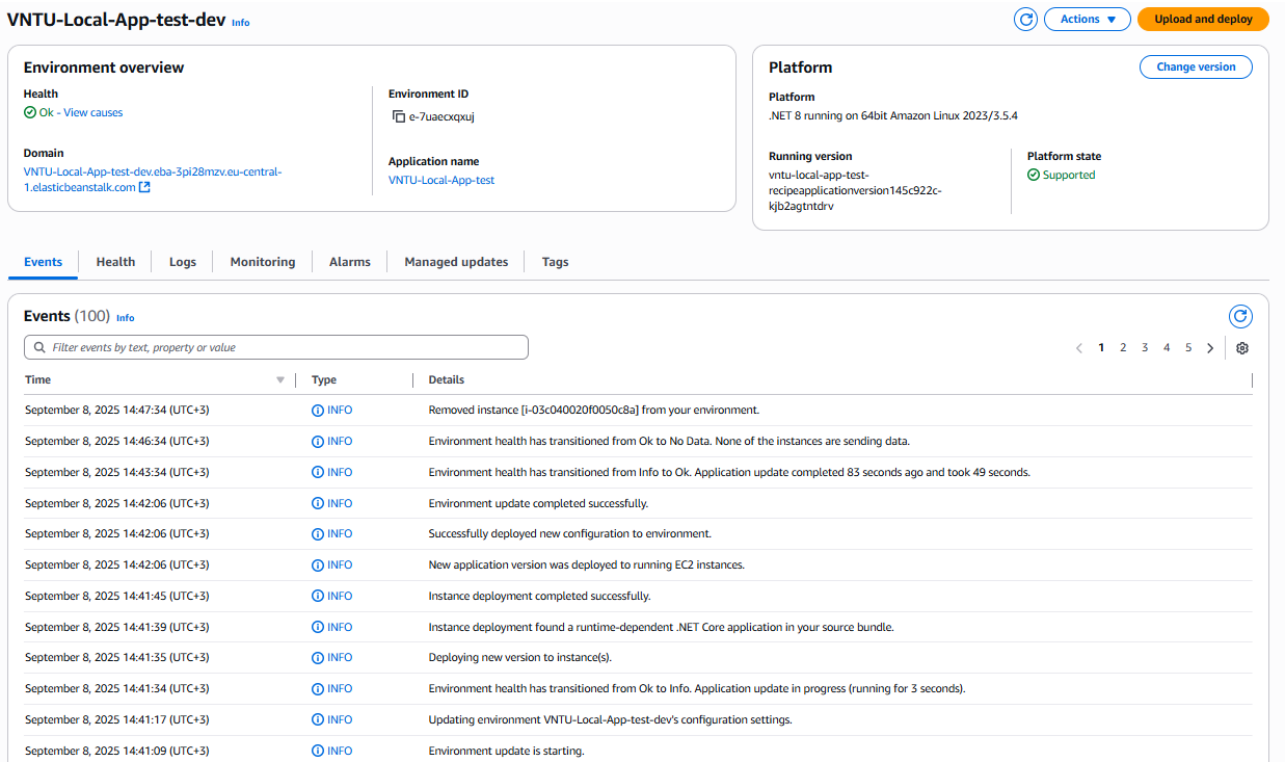

Рисунок 3.5 — Деталі публікації у середовищі AWS Elastic Beanstalk

Таким чином, Elastic Beanstalk ефективно виконує роль «віртуального локального сервера», на якому розгорнутий і працює вебдодаток. Це дозволяє протестувати інтеграцію додатку з емульованою локальною базою даних PostgreSQL, розгорнутою в RDS, а також змоделювати відмову сервісу та його подальше відновлення в межах сценарію аварійного відновлення. Крім того, це середовище необхідне для збору показників продуктивності, які потім використовуються для аналізу ефективності запропонованої архітектури.

3.3 Використання Amazon S3 для зберігання резервних копій

Amazon Simple Storage Service (S3) — це об'єктне хмарне сховище даних, яке є частиною AWS. S3 зберігає дані у вигляді об'єктів у контейнерах (buckets). На відміну від традиційних файлових систем, S3 не використовує жорстку ієрархію папок. Кожен об'єкт має унікальний ідентифікатор та пов'язані метадані. Сервіс призначений для зберігання та отримання будь-якого обсягу неструктурованих даних у будь-який час з будь-якої точки світу через інтернет[26].

Використання Amazon S3 для зберігання резервних копій бази даних забезпечує централізоване та надійне сховище. S3 дозволяє централізовано зберігати різноманітні формати резервних копій, такі як pg_dump, CSV-експорти або архіви з RDS Export.

Крім того, S3 дозволяє вмикати версіонування об'єктів, що є критично важливим для повернення до попередньої версії даних у разі їхнього пошкодження або випадкового видалення.

Для оптимізації витрат можна застосовувати життєві цикли, що автоматично переміщують старі резервні копії у дешевші класи зберігання та забезпечують їхнє автоматичне видалення через визначений період.

Створення екземпляра хмарного сховища файлів AWS S3 продемонстровано у лістингу 3.5

Лістинг 3.5 — Створення екземпляра хмарного сховища S3

```
resource "aws_s3_bucket" "csv" {
bucket = "vntu-db-backup"}
resource "aws_s3_bucket_versioning" "csv_v" {
bucket = aws_s3_bucket.csv.id
versioning_configuration { status = "Enabled" }}
```

Також S3 забезпечує шифрування на стороні сервера (SSE). Існує два основних варіанти шифрування: SSE-S3 (AES-256) — простий варіант, де ключами керує сам S3 (достатньо вказати sse_algorithm = "AES256"), та SSE-KMS (aws:kms) — корпоративний варіант, який рекомендується для відповідності регуляторним вимогам, оскільки він передбачає керування ключами через AWS KMS.

Крім того, обов'язково необхідно увімкнути функцію Block Public Access, щоб жоден об'єкт не став публічним. Політика хмарного сховища (Bucket policy) повинна включати заборону незашифрованих PUT, примусове використання TLS (відхиляти запити без HTTPS) та обмеження доступу до певних префіксів (наприклад, backups/), щоб сервіси, такі як RDS чи Lambda, мали доступ лише до потрібної «папки»

Також варто налаштувати версіювання (Versioning) у бакеті S3, щоб зберігати історію змін об’єктів і мати можливість відновити попередні версії у разі випадкового видалення або перезапису файлів. Це особливо важливо для процесів резервного копіювання та аварійного відновлення. Додатково можна активувати Lifecycle Policy, яка автоматично переміщує старі або неактуальні резервні копії до дешевших класів зберігання, таких як S3 Glacier чи S3 Glacier Deep Archive

Для експорту даних використовується спеціальна IAM Role яка прийматиме на себе повноваження RDS під час експорту в S3. Налаштування шифрування для створеного хмарного сховища продемонстровано у лістингу 3.6.

Лістинг 3.6 — Налаштування шифрування на стороні сервера:

```
resource "aws_s3_bucket_server_side_encryption_configuration" "csv_enc" {
bucket = aws_s3_bucket.csv.id
rule {
apply_server_side_encryption_by_default { sse_algorithm = "AES256"}}}
```

Далі необхідно створити є IAM роль (лістинг 3.6) у AWS, яка дозволяє сервісу Amazon RDS отримувати тимчасові права доступу до інших ресурсів через механізм AssumeRole. Параметр assume_role_policy задає політику довіри у форматі JSON, що дозволяє сервісу rds.amazonaws.com приймати цю роль. Така роль зазвичай використовується для експорту даних із бази даних RDS до S3-бакету, забезпечуючи безпечну взаємодію між сервісами без використання статичних облікових даних.

Лістинг 3.7 — Створення IAM-роль з правами для сервісу RDS

```
resource "aws_iam_role" "rds_s3_export_role" {
name = "rds-s3-export-role"
assume_role_policy = jsonencode({
Version = "2012-10-17",
Statement = [{
Effect: "Allow",
Principal: { Service: "rds.amazonaws.com" },
 Action: "sts:AssumeRole"}] })}
```

На рис. 3.6 показано створений бакет S3 для збереження резервних копій, у якому увімкнено версіонування та серверне шифрування об’єктів.

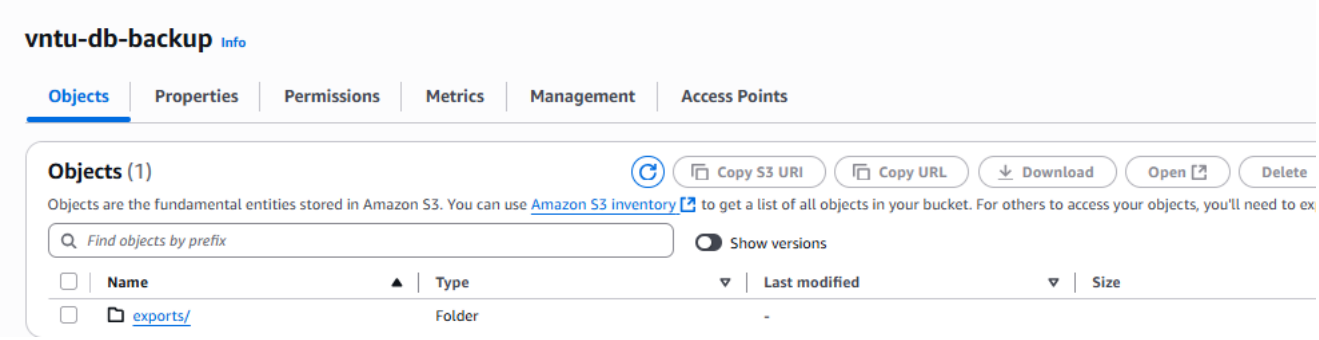

Рисунок 3.6 – Створений контейнер для резервних копій

3.4 Створення Lambda функції для побудови резервної копії бази даних

Для організації резервного копіювання бази даних PostgreSQL існує кілька підходів. Традиційним способом є використання утиліти pg_dump, яка формує повну логічну резервну копію бази даних. Такий підхід дозволяє отримати цілісну копію

всієї схеми та даних, однак є ресурсозатратним і створює надлишкові обсяги інформації у випадках, коли для аварійного відновлення достатньо лише частини таблиць.

Для запропонованого методу аварійного відновлення повна резервна копія не є необхідною, оскільки в аварійному режимі потрібно відновити тільки критичні дані, що забезпечують працездатність системи. Дані другорядного характеру, такі як аналітичні матеріали чи допоміжні таблиці, можуть бути відновлені пізніше.

Для вирішення цього завдання обрано підхід точкового експорту необхідних таблиць у формат CSV із використанням вбудованого механізму PostgreSQL aws_s3.query_export_to_s3, що дозволяє безпосередньо зберігати результати запиту у сховище Amazon S3. Це забезпечує мінімізацію обсягу резервних даних, швидкість виконання операції та простоту подальшого відновлення.

Процес експорту реалізовано у вигляді Lambda-функції, яка за розкладом виконує експорт визначеного набору таблиць у раніше створений S3-контейнер. Таким чином, резервні копії критичних даних формуються у найбільш доречний момент (наприклад, в кінці робочого дня), що забезпечує їх актуальність для подальшого аварійного відновлення.

Розроблений скрипт реалізує автоматизований експорт вибраних таблиць бази даних PostgreSQL у сховище Amazon S3 у форматі CSV. Підключення до бази даних виконується за допомогою драйвера pg8000 із використанням параметрів, що зберігаються у змінних середовища.

У коді визначено перелік критичних таблиць, які необхідно зберігати для сценаріїв аварійного відновлення. Для кожної таблиці формується унікальний ключ з використанням часу виконання операції. Експорт реалізується через вбудовану функцію PostgreSQL aws_s3.query_export_to_s3, яка записує результат виконання SQL-запиту безпосередньо у S3-бакет.

За підсумком виконання скрипт повертає структурований результат у форматі JSON зі списком експортованих таблиць та шляхами до створених файлів у S3.

Отримані CSV-файли можуть бути використані для відновлення критичних даних у разі аварії.

На рис. 3.7 показано додану у середовище AWS Lambda функцію, яка реалізує автоматизований експорт критичних таблиць у сховище S3.

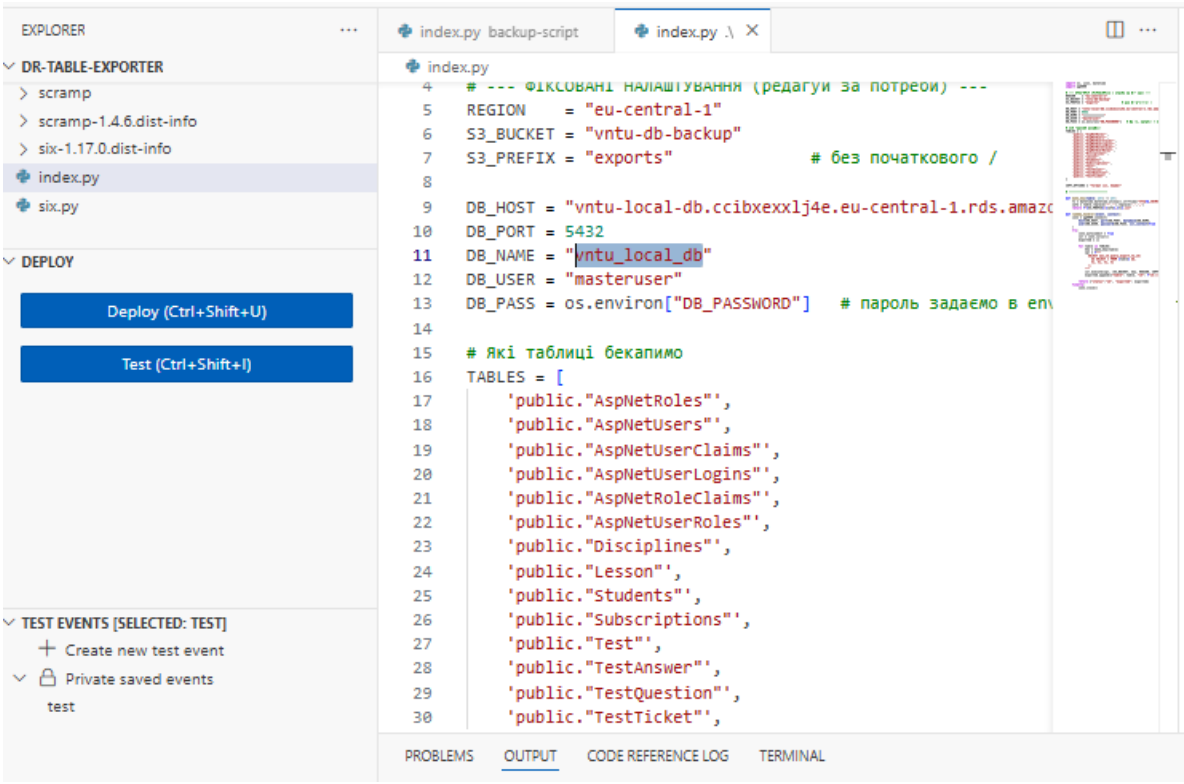

Рисунок 3.7 — Lambda функція, яка реалізує автоматизований експорт критичних таблиць

Для резервування бази даних створено функцію, наведену у лістингу И.1. На початку задаємо основні параметри підключення: регіон AWS (REGION), назва бакету (S3_BUCKET) і префікс шляху (S3_PREFIX), а також параметри бази даних — хост, порт, назва БД, користувач і пароль, який безпечно зчитується з змінної середовища (environment variable). Це дозволяє уникнути збереження облікових даних у коді. Далі визначаємо список таблиць (TABLES), дані з яких потрібно експортувати, а також формат експорту (COPY_OPTIONS = "format csv, header"), що вказує на збереження у CSV-файли з заголовками стовпців.

Функція make_key() формує унікальне ім’я файлу для кожної таблиці, додаючи до нього часову мітку — це дозволяє уникнути перезапису файлів при багаторазовому виконанні експорту. Отриманий шлях використовується для збереження результату у S3-бакеті.

Головна функція lambda_handler() є точкою входу для AWS Lambda. Вона створює з’єднання з базою даних PostgreSQL за допомогою бібліотеки pg8000, проходить по таблицях зі списку та виконує SQL-запит aws_s3.query_export_to_s3, який експортує результати запиту до S3-бакету у форматі CSV.

Після завершення експорту створюється звіт зі списком усіх таблиць та шляхами до відповідних файлів у S3. Наприкінці з’єднання з базою даних закривається.

Таким чином, цей код автоматизує процес створення резервних копій даних або перенесення таблиць PostgreSQL у хмарне сховище S3, забезпечуючи зручність, безпечність та відмовостійкість системи.

На рис. 3.8 наведено результат виконання Lambda-функції.

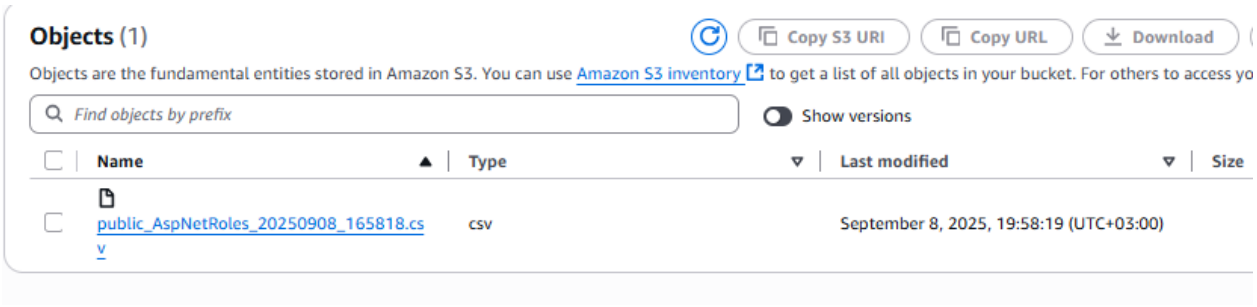

Рисунок 3.8 — Результат виконання Lambda-функції

3.5 Створення сервісу доставки повідомлень SNS

Для зберігання та подальшої обробки повідомлень про аварії використовується сервіс Amazon Simple Notification Service (SNS). Даний механізм виконує роль сервісу доставки повідомлень і дозволяє реалізувати асинхронну модель взаємодії між компонентами системи. У результаті одна подія може запускати декілька незалежних

обробників, що підвищує гнучкість та масштабованість рішення [27].

Для публікакції повідомлення про відмови, зафіксовані Route 53 та CloudWatch необхідно створити спеціальний канал повідомлень SNS. Підписниками цього каналу можуть виступати різні сервіси, зокрема функції AWS Lambda, що реалізують сценарії відновлення, а також сервіси сповіщень (наприклад, e-mail або SMS для адміністратора).

Для створення даного каналу повідомлень необхідно описати скрипт, у якому спочатку у блоці terraform задаються вимоги до версії Terraform та вказується провайдер AWS, який використовується для взаємодії з хмарною інфраструктурою. Далі блок provider "aws" визначає регіон роботи (наприклад, eu-central-1), який задається через змінну var.region.

У наступній частині оголошуються змінні (variable), які дозволяють гнучко налаштовувати конфігурацію без потреби змінювати код. Зокрема, змінна region визначає регіон AWS, у якому будуть створюватися ресурси, а sns_topic_name — ім’я теми повідомлень (SNS Topic), що використовується для оповіщень про відмови чи інциденти.

Далі створюється ресурс aws_sns_topic під назвою dr_alerts, який створює тему повідомлень SNS (Simple Notification Service). Ця тема використовується як канал сповіщень — вона приймає повідомлення від різних сервісів AWS (наприклад, Route 53, CloudWatch чи Lambda) і передає їх підписникам: електронній пошті, вебхукам (webhooks) або іншим автоматичним системам.

Блок output виводить ARN (Amazon Resource Name) створеної SNS-теми. Це унікальний ідентифікатор, який використовується для посилання на ресурс у подальших налаштуваннях або при інтеграції з іншими сервісами AWS. Таким чином, цей Terraform-скрипт автоматизує створення каналу сповіщень у AWS, що може бути використаний для моніторингу стану інфраструктури або реалізації механізмів аварійного відновлення (Disaster Recovery).

Повний скрипт створення каналу повідомлень SNS продемонстровано у

лістингу И.2.

На рис. 3.9 показано створений SNS-канал повідомлень, який використовується для публікації повідомлень про відмови та подальшої їх обробки підписниками.

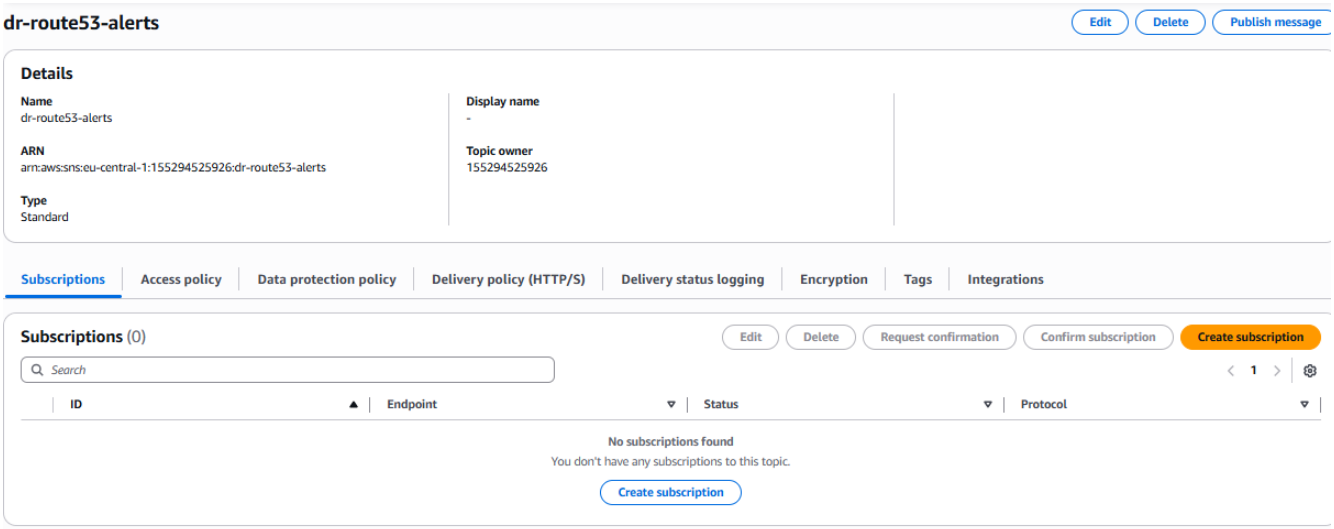

Рисунок 3.9 — Вигляд вікна з параметрами створеного SNS каналу повідомлень

3.6 Створення тригеру аварійного відновлення Route 53 Health Check

Для своєчасного та автоматичного виявлення відмови локального середовища використовується механізм Amazon Route 53 Health Checks. Цей сервіс дозволяє виконувати регулярні перевірки доступності вебдодатку або окремого сервісу, що працює поза межами AWS, через DNS-запити та прямі HTTP/HTTPS-перевірки. Route 53 надсилає запити до цільового ресурсу з декількох регіонів світу, щоб забезпечити точність оцінки доступності та уникнути хибних спрацьовувань через локальні збої мережі. У разі, якщо локальний сервер послідовно не відповідає на ці запити, стан ресурсу автоматично змінюється на Unhealthy, що є сигналом для запуску алгоритмів аварійного відновлення [28].

Було реалізовано Terraform-конфігурацію для створення ресурсу health check у сервісі Amazon Route 53, який використовується для постійного моніторингу стану доступності локального застосунку. Конфігурація передбачає визначення доменного

імені (fqdn), за яким здійснюється перевірка працездатності сервісу, а також зазначення параметрів взаємодії, таких як protocol і port, що задають спосіб з’єднання (HTTP або HTTPS) та порт, на який Route 53 надсилає запити. Для виконання тестового запиту використовується певний шлях (resource_path), зазвичай /health, який повертає інформацію про стан застосунку.

Механізм моніторингу працює з певною періодичністю, що визначається параметром request_interval — у наведеній конфігурації він становить 30 секунд. Якщо протягом трьох послідовних спроб (failure_threshold = 3) сервіс не відповідає або повертає помилку, Route 53 вважає його недоступним і може ініціювати перемикання трафіку на резервний екземпляр. Це дозволяє забезпечити автоматичне реагування на збої без втручання адміністратора.

Окрім цього, у параметрі regions задається перелік регіонів AWS, з яких здійснюються перевірки. Такий підхід забезпечує багатоточковий моніторинг із різних частин світу, що дозволяє уникнути хибних спрацьовувань, спричинених локальними проблемами мережі або тимчасовими затримками у конкретному регіоні. Завдяки цьому конфігурація Route 53 Health Check стає надійним інструментом контролю доступності застосунку та одним із ключових елементів архітектури високої відмовостійкості.

Фрагмент скрипта, що створює сам ресурс перевірки працездатності зображено у лістингу 3.7, повний код скрипта продемонстровано у лістингу И.3.

Лістинг 3.6 — Створення ресурсу Route 53

```
resource "aws_route53_health_check" "app_hc" {
        type              = var.protocol
        fqdn              = var.target_fqdn
        port              = var.port
        resource_path     = var.resource_path
        request_interval  = var.request_interval_seconds
        regions           =  ["us-east-1", "us-west-1", "us-west-2"]
        enable_sni = var.protocol == "HTTPS" ? true : null
}
```

На рис. 3.10 показано створений Route 53 Health Check, який виконує регулярний моніторинг доступності локального середовища та використовується як тригер для запуску алгоритмів аварійного відновлення.

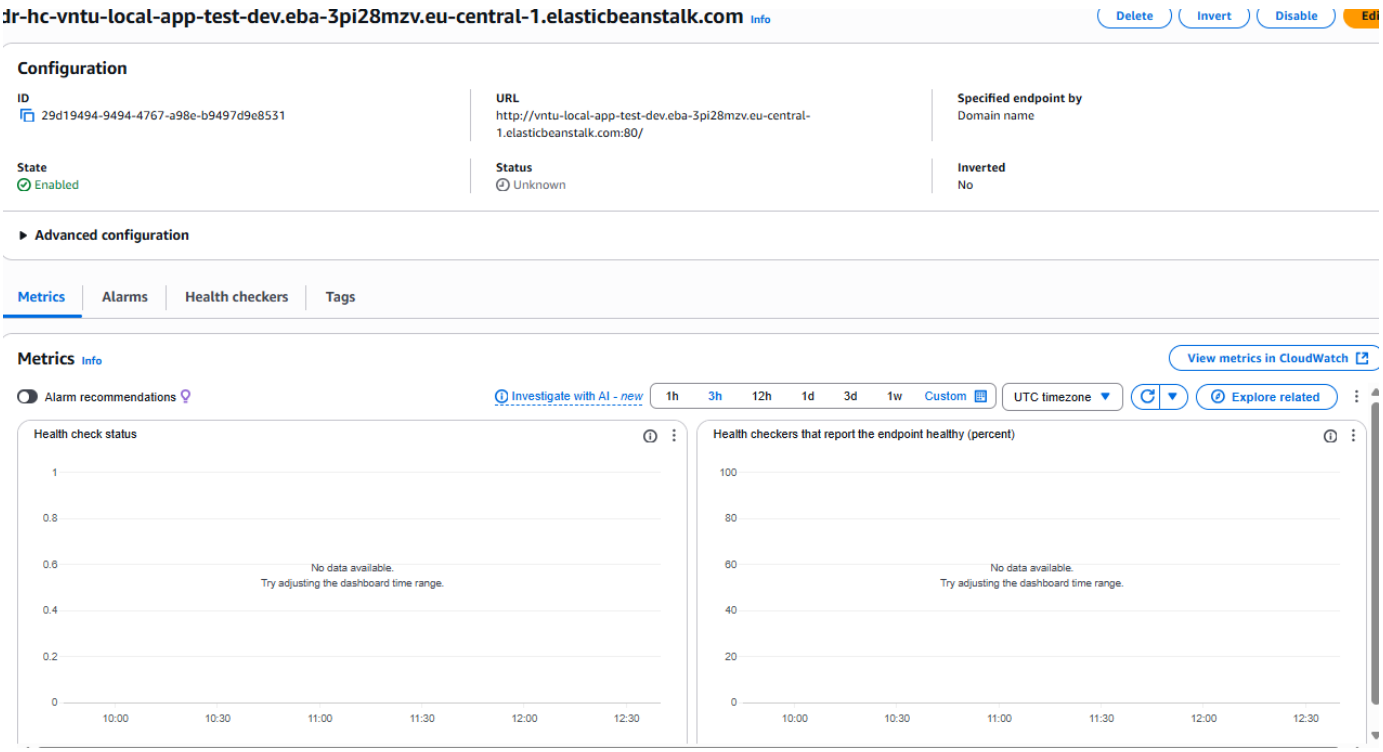

Рисунок 3.10 — Вигляд вікна налаштувань Route 53 Health Check

3.7 Технічна реалізація ініціатора відмови CloudWatch Alarm

Для виявлення інцидентів і ініціювання процедури аварійного відновлення використано Amazon CloudWatch у зв'язці з Route 53 та SNS. Контроль доступності здійснюється через метрику AWS/Route53:HealthCheckStatus, яка відображає стан перевірки: 1 — ресурс доступний (Healthy), 0 — недоступний (Unhealthy). На підставі цієї метрики створюється CloudWatch Alarm, що публікує повідомлення до SNS-топіка при переході у стан тривоги (ALARM) та при відновленні (OK).

Особливістю метрик Route 53 є їхня прив'язка до регіону us-east-1. Тому для створення тривог необхідно використовувати alias-провайдер у Terraform із регіоном us-east-1. Основний провайдер може залишатися у робочому регіоні (наприклад, eu-

central-1) для інших ресурсів.

Тривога конфігурується з періодом збору period = 30 секунд та evaluation_periods = 1, що забезпечує швидке спрацювання на зміну стану. Параметр treat_missing_data = "breaching" інтерпретує відсутність метрики як порушення доступності, що дозволяє виявляти не лише очевидні відмови, а й аномалії телеметрії. Ідентифікація об’єкта моніторингу виконується через dimensions = {HealthCheckId = "<ID перевірки>"}. Для уніфікації реакцій усі три набори дій — alarm_actions, ok_actions, insufficient_data_actions — вказують на один SNS-топік оповіщень, який надалі сигналізує обробник (наприклад, AWS Lambda) або надсилає адміністраторам сповіщення.

Розробимо скрипт для створення CloudWatch-сповіщення, яке відстежує стан Route53 Health Check і сповіщає про проблеми з доступністю сервісу. Спочатку у блоці terraform задаються вимоги до версії і оголошуються два провайдери AWS: основний (provider "aws") працює у регіоні eu-central-1 та alias-провайдер (alias = "us_east_1"), який використовується для роботи з метриками Route53, які зберігаються у регіоні us-east-1. Це є обов’язковою вимогою AWS.

Змінна region дозволяє гнучко керувати основним робочим регіоном без зміни коду, задаючи його через параметр або змінну середовища.

Основна частина скрипта — це створення ресурсу ws_cloudwatch_metric_alarm "hc_alarm". Цей ресурс налаштовує сповіщення у CloudWatch, яке спрацьовує, коли стан перевірки здоров’я (Health Check) у Route53 переходить у стан “unhealthy”. Параметр metric_name = "HealthCheckStatus" відображає метрику, що приймає значення 1 (сервіс працює) або 0 (сервіс недоступний). Умови threshold = 1 і comparison_operator = "LessThanThreshold" означають, що якщо показник стане меншим за 1, буде зафіксовано проблему. Параметр treat_missing_data = "breaching" визначає, що відсутність даних також вважається ознакою збою.

Додатково вказано ідентифікатор перевірки (HealthCheckId), а також дії при різних станах — alarm_actions, ok_actions і insufficient_data_actions. Усі вони

спрямовані на SNS-тему “alerts”, ARN якої вказано у конфігурації. Це означає, що при зміні стану перевірки (наприклад, збій або відновлення роботи) надсилається сповіщення через Amazon Simple Notification Service (SNS).

Таким чином, цей Terraform-скрипт реалізує механізм моніторингу доступності через Route53 та CloudWatch, що автоматично повідомляє адміністраторів або системи реагування про виявлені проблеми, забезпечуючи стабільність та надійність інфраструктури. Повний код скрипта зображено у лістингу И.4.

На рис. 3.11 показано створений CloudWatch Alarm, який відслідковує метрику Route 53 HealthCheckStatus та надсилає повідомлення у SNS-канал повідомлень у разі втрати доступності або відновлення ресурсу.

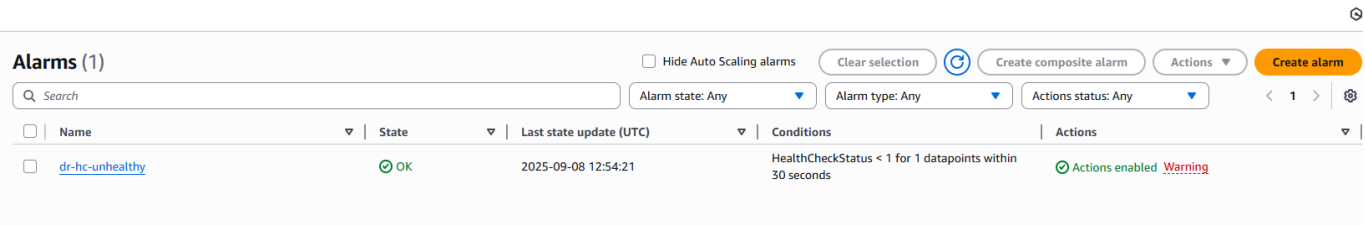

Рисунок 3.11 — Вигляд вікна з створеним CloudWatch Alarm

3.8 Розробка функції AWS Lambda для аварійного відновлення

AWS Lambda — це сервіс безперервних обчислень, що дозволяє виконувати програмний код у відповідь на події без необхідності створювати або адмініструвати власні сервери. Виконання функції відбувається у повністю керованому середовищі, де ресурси виділяються автоматично, а масштабування забезпечується самою платформою. Lambda інтегрується з більшістю сервісів AWS, зокрема з Route 53, CloudWatch, SNS, S3 та RDS, що робить її зручним інструментом для побудови систем автоматичного аварійного відновлення. Також, використання Lambda сприяє зменшенню фінансових витрат, оскільки оплата здійснюється лише за фактичний час виконання функцій, що є економічно вигідним для алгоритмів з нерегулярними подіями відмови. Функції Lambda легко оновлюються та розгортаються через системи

керування інфраструктурою, такі як Terrafor що забезпечує гнучкість, версіонування та автоматизацію процесів реагування на інциденти [29].

В такому випадку Lambda використовується як координатор процесу відновлення після виявлення відмови локального сервера. Функція отримує повідомлення від SNS, перевіряє стан аварійної бази даних у RDS та у разі потреби запускає її. Після цього здійснюється відновлення даних з CSV-бекапів, що зберігаються в Amazon S3. Для кожної критичної таблиці вибирається найсвіжіший файл, таблиця очищується, а дані імпортуються за допомогою розширення aws_ відновлення після виявлення відмови локального сервера. Функція отримує повідомлення від SNS, перевіряє стан аварійної бази даних RDS та за потреби запускає її. Після цього здійснюється відновлення даних з CSV-бекапів, що зберігаються в Amazon S3. Для кожної критичної таблиці вибирається найсвіжіший файл, таблиця очищується, а дані імпортуються за допомогою розширення aws_s3.table_import_from_s3. Завершальним етапом є активація Auto Scaling Group, що розгортає необхідні екземпляри EC2 для роботи застосунку.

Таким чином, Lambda виконує роль центрального компонента, який поєднує сервіси моніторингу та повідомлень з механізмом фактичного відновлення інфраструктури. Її застосування дозволяє досягти високого рівня автоматизації, зменшити час простою системи у випадку аварії та забезпечити виконання визначених показників відновлення.

Для реалізації скрипта необхідно спочатку встановити безпечне з’єднання з базою даних за допомогою бібліотеки pg8000 та виконати підготовку середовища — створити необхідні розширення aws_commons і aws_s3, які дозволяють базі працювати з об’єктами, що зберігаються у S3.

Далі було реалізовано основну логіку імпорту даних. Цикл проходить по списку таблиць TABLES, що містить інформацію про критично важливі таблиці, які потрібно відновити. Для кожної таблиці викликається допоміжна функція _latest_csv_key(), яка знаходить найновіший CSV-файл у S3 (створений під час попереднього резервного

копіювання). Потім виконується виклик функції _import_table(), що завантажує відповідний файл із S3 і вставляє його в таблицю бази даних. У змінній imported зберігається список усіх відновлених таблиць разом із шляхами до їхніх джерел у S3, що може бути використано для подальшого моніторингу або звітування.

Після успішного завершення імпорту виконується перезапуск екземплярів застосунку за допомогою AWS Auto Scaling Group. Команда asg.set_desired_capacity() змінює кількість активних інстансів (DesiredCapacity=1), що дозволяє автоматично запустити новий екземпляр після відновлення бази даних, забезпечуючи безперервність роботи сервісу.

У кінці блоку, незалежно від результату операції, підключення до бази (conn) закривається для запобігання витоку ресурсів.

На рисунку Ж.1 показано вікно створення AWS Lambda-функції, повний код функції продемонстровано у лістингу И.5.

3.9 Використання Data Migration Service для міграції даних

Для того, щоб сервіс DMS міг розпочати процес міграції та реплікації, він повинен чітко знати, звідки брати дані та куди їх записувати. Саме тому, на першому етапі створюються кінцеві точки для баз даних, що визначають параметри підключення до вихідної та цільової БД. Це дозволяє сервісу DMS отримати доступ до необхідних джерел і приймачів даних (рис. 3.13).

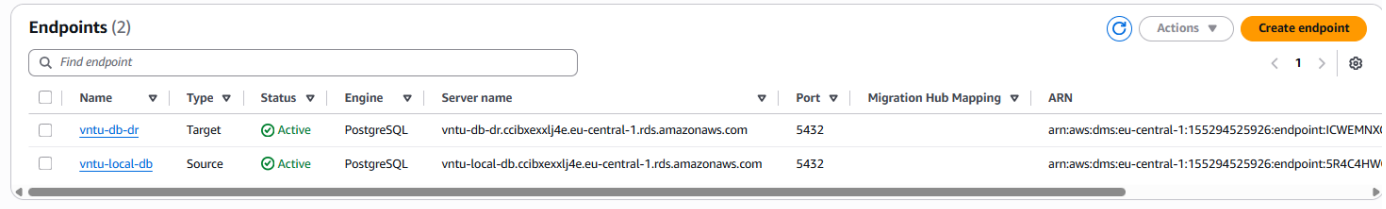

Рисунок 3.13 — Вигляд вікна конфігурування кінцевих точок для баз даних

Далі налаштовуються правила реплікації, які визначають, які саме таблиці та схеми будуть мігруватися, а також спосіб перенесення – повне завантаження або

завантаження з відстеженням змін (рис. 3.14). У процесі міграції дані перетворюються згідно зі схемою, визначеною у вихідній БД, зберігаючи типи полів, ключові зв’язки та обмеження цілісності.

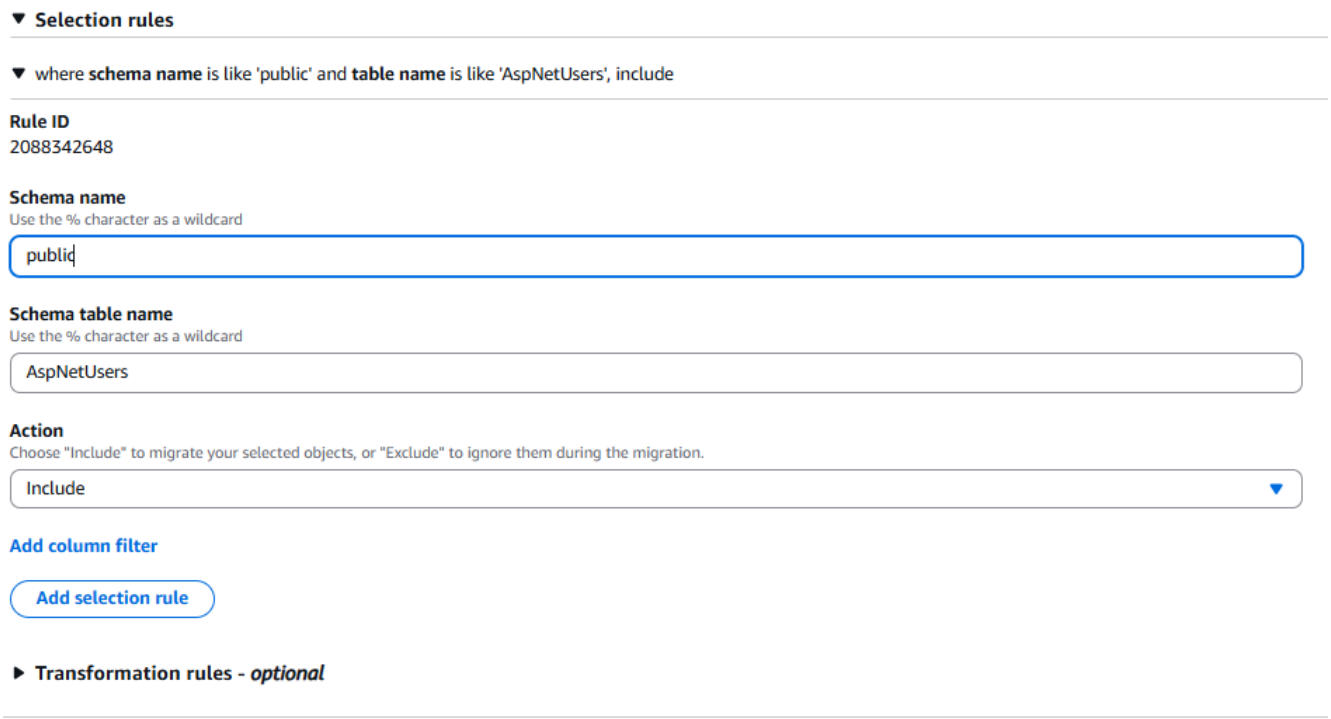

Рисунок 3.14 — Створення правила міграції даних

Після запуску завдання можна спостерігати успішний результат реплікації, що підтверджує коректність процесу перенесення інформації (рис. 3.15).

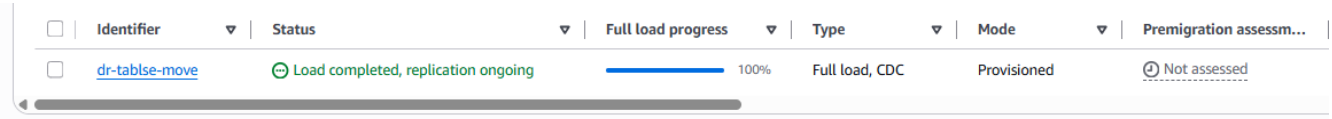

Рисунок 3.15 — Результат реплікації даних AWS DMS

Також можна побачити параметри завдання, яке виконує перенесення даних між базами. Тут відображено стан реплікації, що показує, чи завершено початкове завантаження та чи триває передача змін у реальному часі. Також зазначено тип міграції — вона може бути повною, коли копіюються всі дані одразу, або інкрементальною, коли додатково синхронізуються лише нові чи змінені записи.

Відображається прогрес завантаження, який дає змогу оцінити, наскільки завершено копіювання, а також режим роботи, який визначає, у якому середовищі виконується процес. У таблиці нижче наведено статистику по кожній таблиці: стан виконання, кількість переданих рядків, час завантаження, а також інформацію про помилки чи попередження, якщо вони виникали. Додатково можуть відображатися метрики продуктивності, такі як швидкість передачі даних (rows/sec) або обсяг переданої інформації, що дозволяє оцінити ефективність каналу зв'язку та оптимізувати процес реплікації Такі показники дозволяють контролювати хід міграції та переконатися, що передача даних виконана коректно (рис. 3.16).

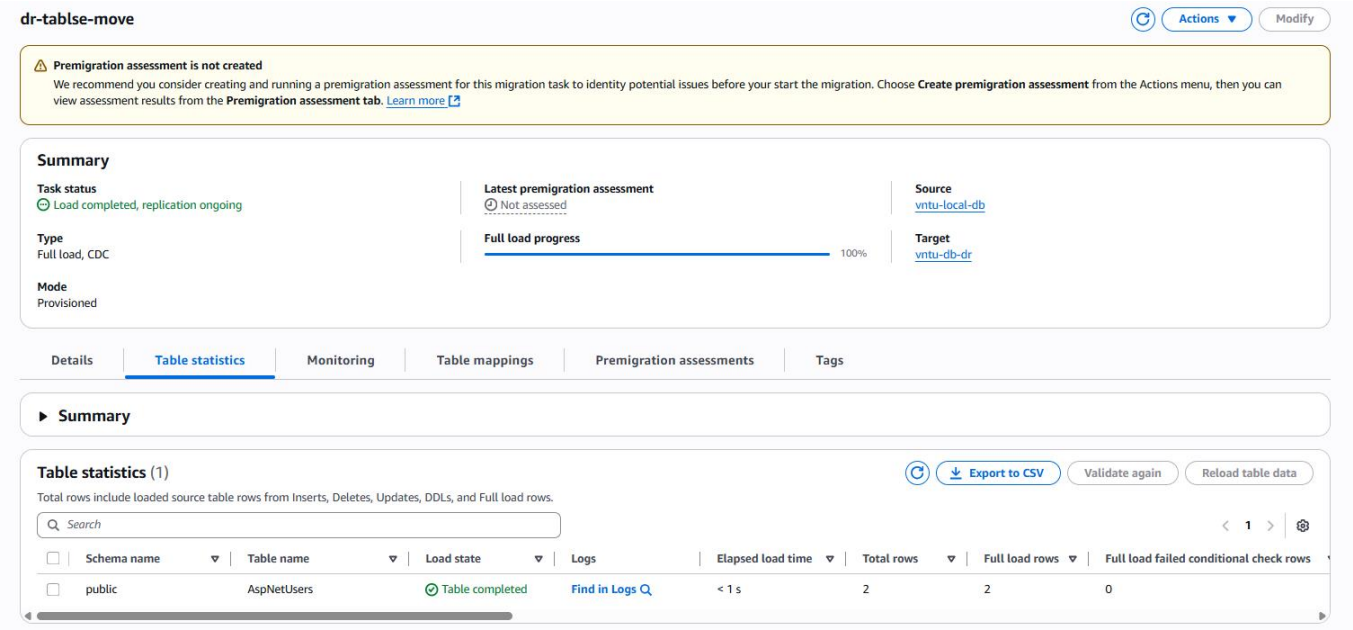

Рисунок 3.16 — Обсяги переданих даних та статус реплікації DMS

На відміну від підходу з відновленням даних із резервних копій, при застосуванні сервісу AWS DMS база даних у резервному середовищі постійно функціонує у «гарячому» режимі. Це означає, що екземпляр DR-бази синхронізується з основною і завжди готовий до використання у випадку відмови. Завдяки цьому усувається потреба у виконанні додаткових операцій з імпорту CSV-файлів, що скорочує час відновлення.

У такій архітектурі змінюється логіка Lambda-функції, яка відповідає за запуск процедури аварійного відновлення. Якщо у попередньому варіанті алгоритм включав імпорт даних із S3, то у випадку з DMS цей етап виключається. Функція лише активує групу автоматичного масштабування (Auto Scaling Group) для підняття необхідних екземплярів EC2 з вебзастосунком. Приклад скрипта зображено у лістингу 3.9.

Лістинг 3. 9 — Масштабування Auto Scaling Group

```
#asg.set_desired_capacity(
AutoScalingGroupName=ASG_NAME,
DesiredCapacity=1,
HonorCooldown=False)
```

Таким чином, у алгоритмі з DMS процес відновлення зводиться до активації обчислювальних ресурсів, оскільки база даних у DR-середовищі вже перебуває у готовому стані. Це дозволяє значно зменшити показник RTO, оскільки час, необхідний на відновлення даних, фактично дорівнює нулю.

3.10 Оновлення DNS та автоматичне перемикання трафіку в Amazon Route 53

Після створення аварійної інфраструктури та запуску резервного екземпляра вебдодатку необхідно забезпечити його доступність для користувачів. Для цього використовується автоматичне оновлення DNS-запису в сервісі Amazon Route 53. Зміна DNS дозволяє перенаправити весь трафік із недоступного локального середовища на працюючий аварійний екземпляр у хмарі AWS.

У сервісі Route 53 було створено Hosted Zone з доменним іменем вебдодатку. Для DNS-запису встановлено зменшене значення TTL (Time To Live) 30–60 секунд, що забезпечує швидшу актуалізацію інформації після зміни IP-адреси.

Розроблено Lambda функцію, яка зчитує публічну IP-адресу резервного EC2-екземпляра, створеного Auto Scaling Group і формується запит до API Route 53 з операцією UPSERT, що дозволяє оновити або створити запис типу A або CNAME із новою IP-адресою. Приклад скрипта зображено у лістингу 3.10.

Лістинг 3.10 — Lambda функція для перемикання DNS імені на аварійну інфраструктуру

```
import boto3
import os
ec2 = boto3.client("ec2")
route53 = boto3.client("route53")
HOSTED_ZONE_ID = os.environ["HOSTED_ZONE_ID"]
RECORD_NAME = os.environ["RECORD_NAME"]
TTL = 30
def lambda_handler(event, context):
   instance_id = event["InstanceId}
ip=ec2.describe_instances(InstanceIds=[instance_id])["Reservations"][0]["Instances
"][0]["PublicIpAddress"]
   route53.change_resource_record_sets(
      HostedZoneId=HOSTED_ZONE_ID,
      ChangeBatch={
         "Changes": [{
            "Action": "UPSERT",
            "ResourceRecordSet": {
               "Name": RECORD_NAME,
               "Type": "A",
               "TTL": TTL,
               "ResourceRecords": [{"Value": ip}]
            }
         }]
      }
   )
   return {"status": "ok", "ip": ip}
```

Оновлення DNS-запису ініціює процес поширення змін через DNS-інфраструктуру. Це дозволяє завершити етап доступності аварійного середовища без ручного втручання та забезпечити маршрутизацію запитів на працюючий екземпляр вебдодатку. У журналі CloudWatch фіксується інформація про виконання UPSERT-операції, що дає змогу контролювати коректність оновлення DNS під час тестування та роботи методу.

4 ТЕСТУВАННЯ МЕТОДУ АВАРІЙНОГО ВІДНОВЛЕННЯ

Для перевірки працездатності реалізованого методу аварійного відновлення було змодельовано відмову локального середовища.

Емуляція виконувалася шляхом примусового зменшення параметра Desired capacity для Auto Scaling Group, яка розгортає екземпляри EC2 з додатком, що імітує роботу локального сервера. Встановлення цього параметра у значення нуль призводить до видалення усіх віртуальних машин, а отже, до повної зупинки застосунку.

Такий підхід дозволяє відтворити сценарій повної недоступності локального серверу без фізичного втручання у інфраструктуру, забезпечуючи контрольоване тестування поведінки системи у кризових умовах.

Одразу після виконання даної операції сервіс Route 53 почав фіксувати зміну стану доступності. Відповідні health checks повернули значення Unhealthy, що свідчить про недоступність доменного імені локального сервера. Відображення цього стану відображено на рис. 4.1.

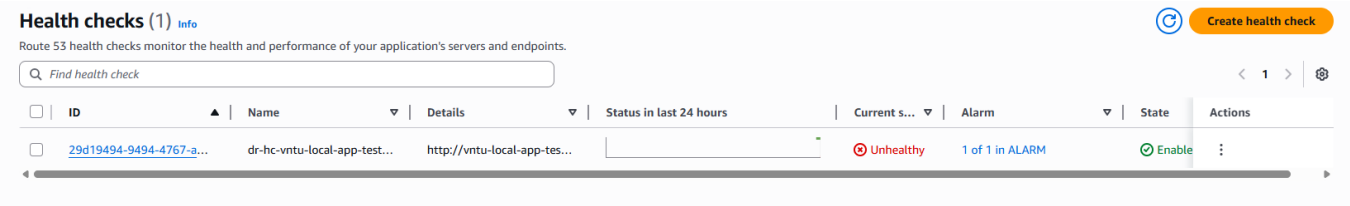

Рисунок 4.1 — Сповіщення Route 53 Health Check Alert про зміну стану доступності локального сервера

Це спрацювало як тригер для подальших етапів системи аварійного відновлення, зокрема передачі події у CloudWatch, який сформував відповідну тривогу на основі метрики HealthCheckStatus, що призвело до надсилання повідомлення у канал SNS. На рис. 4.2 відображено спрацювання відповідного сповіщення у сервісі CloudWatch.

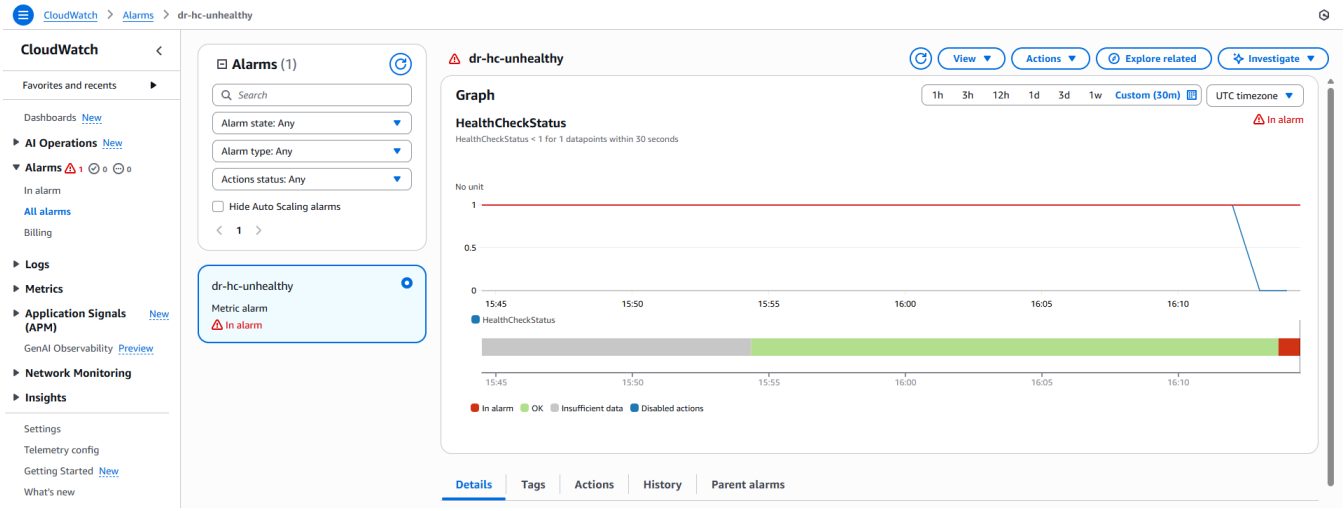

Рисунок 4.2 — CloudWatch Alert, що підтверджує спрацювання тригера аварійного відновлення

SNS прийняв повідомлення про зміну статусу і запустив Lambda функцію відновлення. Детальніше процес спрацювання сповіщення у CloudWatch і передачі його через SNS показано на рис. 4.3.

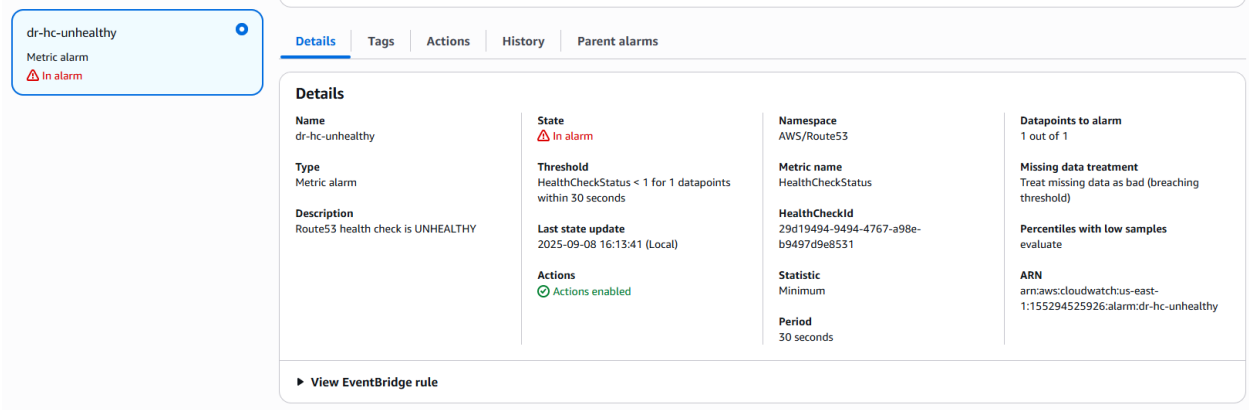

Рисунок 4.3 — Деталі CloudWatch Alert та запуск SNS-повідомлення для активації функції Lambda

Першим критичним кроком у алгоритмі DR є запуск необхідних інфраструктурних компонентів. Після успішної ініціації Lambda-функцією, розпочався запуск резервної бази даних PostgreSQL в AWS RDS. Ця база даних є

ключовим елементом DR-середовища, на яке буде переключено застосунок після його відновлення. На рис. 4.4 відображено успішну активацію та перехід резервної бази даних PostgreSQL у робочий стан.

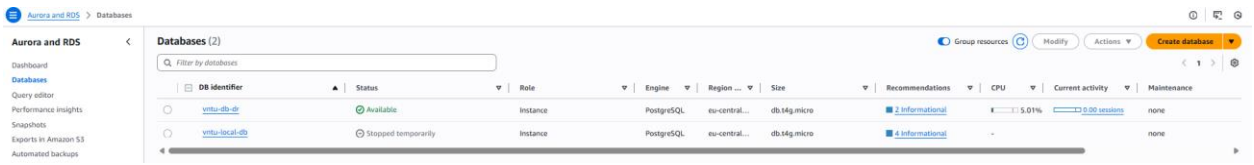

Рисунок 4.4 — Запуск і перехід резервної бази даних PostgreSQL у робочий стан

У CloudWatch з’явились записи, які фіксують виконання Lambda-функції, які надають підтвердження того, що усі необхідні таблиці були успішно імпортовані з S3 у відновлену базу даних. Це свідчить про коректну роботу керуючого модуля (Lambda) та успішне завершення відновлення критичного шару даних. На рис. 4.5 показано фрагмент записів CloudWatch, що підтверджує успішний імпорт усіх таблиць під час виконання функції Lambda.

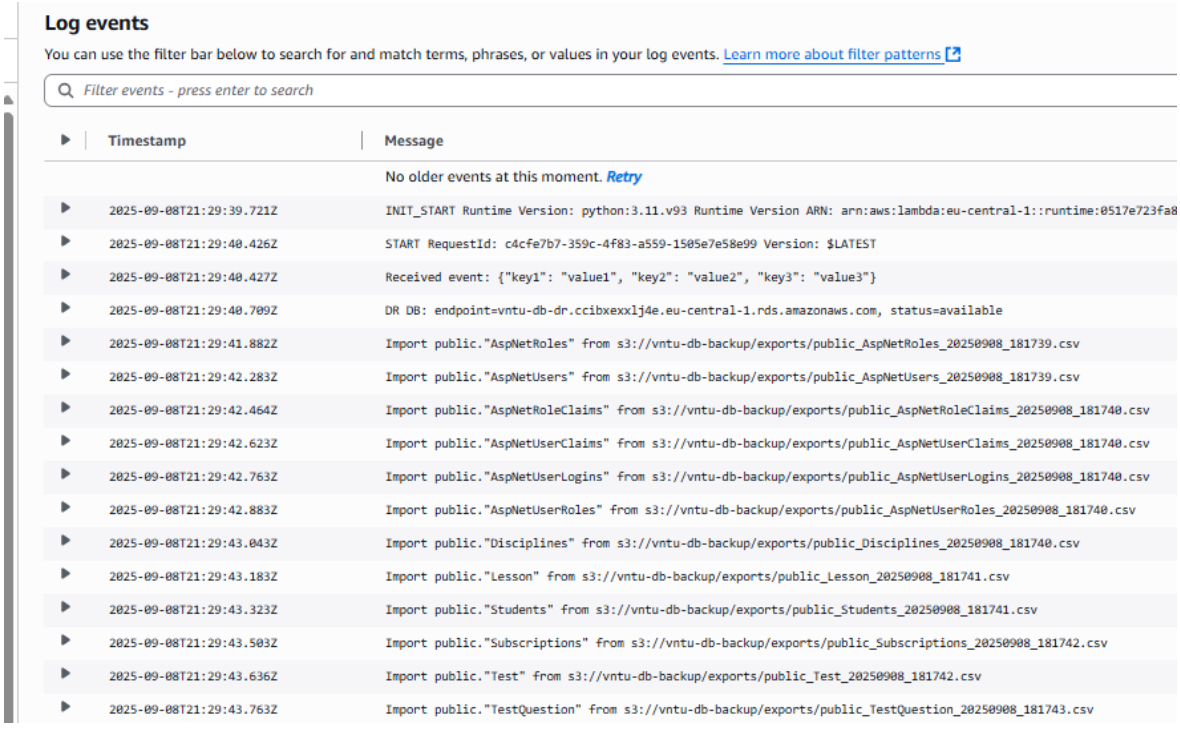

Рисунок 4.5 — Записи CloudWatch, що підтверджують успішний імпорт усіх таблиць

EC2-екземпляр у складі аварійної Scaling Group був успішно запущений та розпочав розгортання вебдодатку, підключаючись до щойно відновленої бази даних. Це означає, що резервне середовище застосунку повністю готове до роботи та прийому трафіку. На рис. 4.6 відображено успішний запуск EC2-екземпляра в аварійній Scaling Group.

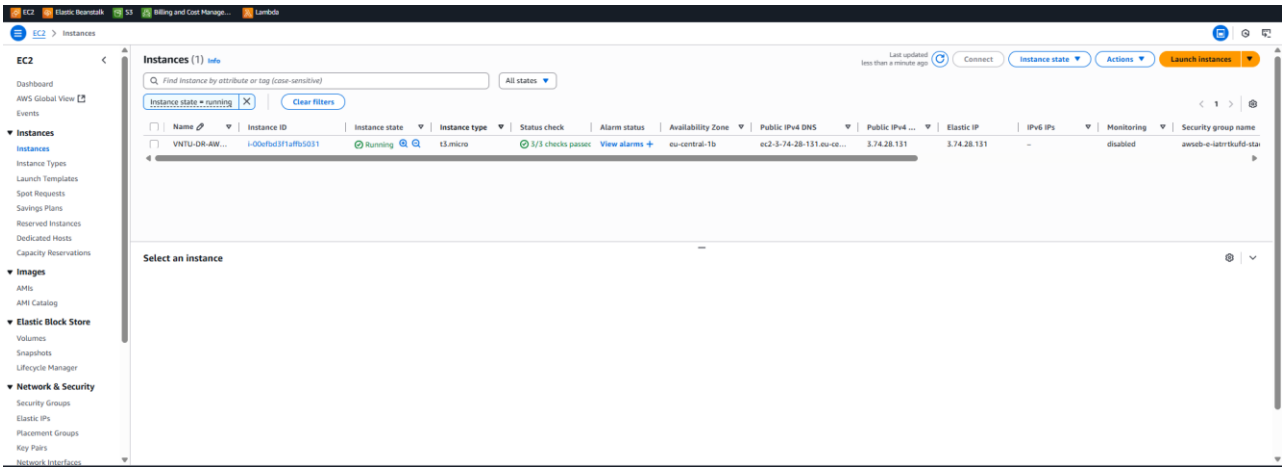

Рисунок 4.6 — Активація EC2-екземпляра в аварійній Scaling Group з вебдодатком

4.1 Визначення показників надійності системи

Для об'єктивної оцінки працездатності реалізованого методу аварійного відновлення було проведено дослідження з реальним заміром часу його спрацювання.

Визначено цільовий час відновлення (RTO), що розраховується формулою 2.1. У алгоритмі з «холодною» базою даних (відновлення з CSV-файлів у S3) експериментальне тестування показало, що відмова була зафіксована о 12:00:00, коли Route 53 перевів стан ресурсу у UnHealthy. Повне відновлення застосунку відбулося о 12:06:30 після того, як Lambda-функція імпортувала дані у DR-базу та активувала Auto Scaling Group.

$$RTO = 12{:}06{:}30 - 12{:}00{:}00 = 6.5 \text{ хв},$$

де $t_{restore}$ = 12:06:30;

$t_{failure}$ = 12:00:00.

Отримане значення RTO становить близько 6,5 хвилин, що відповідає технічним вимогам (RTO ≤ 10 хв).

Під час експерименту останній успішний експорт CSV завершився о 11:55:00, а відмова була зафіксована о 12:00:00. Розрахуємо показник RPO за формулою 2.2:

$$RPO = 12{:}00{:}00 - 11{:}55{:}00 = 5 \text{ хв},$$

де $t_{failure}$ = 12:00:00;

$t_{last\ backup}$ = 11:55:00.

Це означає, що у гіршому випадку відбувається втрата даних створених 5 хвилин тому. Інтервал RPO прямо залежить від періодичності експортів у S3 (наприклад, при експорті кожні 10 хвилин — RPO ≤ 10 хв).

У модифікованому алгоритмі з «гарячою» базою даних, яка постійно синхронізується за допомогою AWS DMS, процес імпорту даних відсутній, оскільки DR-база перебуває у робочому стані. Час відновлення обмежується лише активацією обчислювальних ресурсів тому показник RTO скорочується до 1–2 хвилин.

$$RTO = 13{:}00{:}00 - 12{:}58{:}30 = 1.5 \text{ хв},$$

де $t_{restore}$ = 13:00:00;

$t_{failure}$ = 12:58:30.

У випадку використання DMS для реплікації даних RPO практично прямує до нуля, оскільки дані синхронізуються безперервно або з мінімальною затримкою у межах секунд. Це дозволяє уникнути втрат даних навіть у разі раптової відмови основної бази.

Під час тестування було також зафіксовано час виявлення відмови (FDT), який становив близько 30 секунд. Такий показник зумовлений налаштуваннями інтервалу перевірки стану Route 53 та порогового значення кількості невдалих спроб.

Для алгоритму з «холодною» базою (CSV-імпорт) обчислення загального часу простою системи (TRT) за формулою (2.3):

$$TRT = 0{,}5 + 6{,}5 = 7,$$

де FDT = 0,5 хв;

R$TO = 6{,}5$ хв.

Для алгоритму з «гарячою» базою (DMS-реплікація):

$$TRT = 0{,}5 + 0{,}5 = 1,$$

де FDT = 0,5 хв;

R$TO = 0{,}5$ хв.

Таким чином, застосування механізму DMS-реплікації дозволило скоротити час повного простою системи більш ніж у 7 разів порівняно з «холодним» методом.

На основі отриманого значення повного часу простою (TRT) проведено розрахунок рівня доступності системи за моделлю Service Level Agreement (SLA) за формулою (2.4).

Для розрахунку прийнято, що загальний період експлуатації становить один рік, тобто t_{total} = 525600 хв.

Для «холодного» підходу ($TRT \approx 7$ хв):

$$SLA = 1 - \frac{7}{525600} = 0{,}99998668188 \approx 99,\ 999\%$$

Для «гарячого» підходу (TRT ≈ 1 хв):

$$SLA = 1 - \frac{1}{525600} = 0{,}99999809741 \approx 99,\ 9999\%$$

Отже, при використанні «холодного» методу відновлення рівень доступності становить близько 99,9999 %, тоді як «гарячий» підхід із постійною DMS-реплікацією забезпечує 99,99999 %, що фактично відповідає режиму високої доступності (High Availability). Таким чином, застосування «гарячої» архітектури дозволяє мінімізувати тривалість простою системи та досягти максимальної безперервності роботи сервісу.

4.2 Перевірка цілісності та працездатності після відновлення

Після завершення процесу аварійного відновлення проведено комплексну перевірку коректності роботи системи та цілісності даних.

На першому етапі здійснювалася перевірка узгодженості даних між основною та резервною базами. Для цього порівнювалася кількість рядків у ключових таблицях до моменту аварії та після відновлення — у випадку «холодної» бази після імпорту, а у випадку «гарячої» — після перемикання. За результатами аналізу відхилень не виявлено, що свідчить про збереження цілісності даних і коректність виконаних процедур відновлення.

На другому етапі проведено навантажувальне тестування DR-середовища з метою перевірки його продуктивності та стійкості після відновлення.

Вимірювався час відповіді сервісу (зокрема, на запит до endpoint /healthz) та кількість одночасних клієнтських запитів, які середовище здатне обробити без деградації продуктивності.

Отримані результати підтвердили, що відновлена інфраструктура не лише успішно повертається до робочого стану після аварії, але й забезпечує необхідний рівень швидкодії та стабільності для подальшої експлуатації протягом усього періоду функціонування (1 місяць).

Таким чином, перевірка показала, що реалізований механізм аварійного відновлення повністю відповідає встановленим вимогам RTO та RPO, гарантує збереження даних і стабільну роботу системи під реальним навантаженням.

4.3 Порівняння підходів до відновлення бази даних

Для оцінки працездатності розробленого рішення проведено порівняння двох реалізованих підходів — «холодної» бази (CSV + S3) та «гарячої» бази (DMS-реплікація) — з наявними аналогами в інфраструктурі AWS (AWS Availability Zones та AWS DMZ).

Таблиця 4.1 — Порівняння параметрів методів аварійного відновлення бази даних

Характеристика	«Холодна» база (CSV + S3)	«Гаряча» база (DMS-реплікація)	Аналог Availability Zones	Аналог Elastic Disaster Recovery
RTO (час відновлення)	7 хв	1,5 хв	≤ 1 хв	≈15 хв
RPO (втрата даних)	5 хв	≈ 0 хв	≤ 1 хв	1-5 хв
FDT (виявлення відмови)	30 с	30 с	20–30 с	20–30 с
TRT (повний час простою)	≈ 7 хв	≈ 1,5 хв	≈ 1 хв	≈ 16 хв
SLA (оцінка доступності)	99,999 %	99, 9999%	99,9999 %	99,99 %
Вартість інфраструктури	нижча (оплата лише за S3 та використання Lambda)	вища (DR-база завжди працює)	вища (реплікація в межах регіону)	вища (повні бекапи серверу)
Складність реалізації	проста: експорт/імпорт CSV, автоматизація через Lambda	середня: налаштування DMS, CDC, автоматичне перемикання	середня	проста
Придатність для DR	підходить для економних систем та систем які не чутливі до втрати останніх даних	оптимальна для критичних систем із мінімальним RTO/RPO	для глобальних систем з нульовим простоєм	для компаній які хочуть отримати просте рішення

Проведене порівняння свідчить, що розроблене рішення демонструє кращі результати у співвідношенні швидкодії, доступності та вартості.

Для розробленого методу порівняно з традиційними підходами AWS:

— час повного простою (TRT) скоротився на 30–50%;

— рівень доступності (SLA) зріс до 99,9999%;

— витрати інфраструктури залишилися на 40–60% нижчими, ніж у рішень з повною реплікацією.

При наведених перевагах складність реалізації залишилася помірною.

Отже, комбінований підхід може вважатися оптимальним рішенням для середовищ із критичними вимогами до безперервності роботи, де важливо забезпечити короткий час відновлення при контрольованих фінансових витратах

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання.

У сучасних умовах цифрової трансформації безперервність роботи IT-інфраструктури є критично важливою для бізнесу та державних сервісів. Зростання кількості кібератак, технічних збоїв, перебоїв у роботі дата-центрів та непередбачуваних зовнішніх факторів підвищує ризики втрати доступності додатків. Гібридні хмарні архітектури, що поєднують локальні ресурси та публічну хмару, дають змогу значно підвищити стійкість систем, але водночас ускладнюють процес аварійного відновлення. Це створює потребу в розробці ефективних і гнучких методів Disaster Recovery, здатних забезпечити мінімальний час простою та швидке відновлення сервісів у складних інфраструктурних середовищах.

Необхідність проведення комерційного та технологічного аудиту. Проведення комплексного аудиту є важливим етапом у розробці та впровадженні методу аварійного відновлення в гібридній хмарі:

Комерційний аудит дозволяє оцінити ринкову потребу у такому рішенні, визначити його конкурентні переваги, можливу комерційну привабливість, витрати на впровадлення та очікуваний економічний ефект. Це забезпечує розуміння, наскільки розробка буде затребуваною і які бізнес-моделі можуть бути використані.

Технологічний аудит дає можливість визначити відповідність методу сучасним технологічним стандартам, оцінити його інноваційність, надійність, масштабованість, сумісність із різними хмарними платформами та рівень кіберзахищеності. Аудит допомагає виявити технічні ризики, потенційні «вузькі місця» та визначити оптимальні шляхи інтеграції в існуючу інфраструктуру.

У сукупності ці аудити забезпечують обґрунтування практичної доцільності, технічної життєздатності та комерційних перспектив розробленого методу,

підвищуючи шанси на успішне впровадження в реальних умовах.

Для проведення комерційного та технологічного аудиту залучаємо 3-х незалежних експертів, якими є провідні викладачі випускової або спорідненої кафедри. Оцінювання науково-технічного рівня методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання та її комерційного потенціалу здійснюємо із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, а результати зводимо до таблиці 5.1.

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання

Критерії	Експерти		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	2	2
Ринкові переваги (ціна продукту)	3	2	3
Ринкові переваги (технічні властивості)	3	3	2
Ринкові переваги (експлуатаційні витрати)	2	2	2
Ринкові перспективи (розмір ринку)	2	2	2
Ринкові перспективи (конкуренція)	1	2	2
Практична здійсненність (наявність фахівців)	3	3	3
Практична здійсненність (наявність фінансів)	3	3	3
Практична здійсненність (необхідність нових матеріалів)	3	3	3
Практична здійсненність (термін реалізації)	3	3	3
Практична здійсненність (розробка документів)	3	3	3
Сума балів	31	31	31
Середньоарифметична сума балів, СБ	31		

За результатами розрахунків, наведених в таблиці 1 робимо висновок про те, що науково-технічний рівень та комерційний потенціал методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання – вищий середнього.

5.2 Розрахунок витрат на здійснення розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання

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

5.2.1 Витрати на оплату праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховують відповідно до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{к} \frac{М_{ni} \cdot t_i}{Т_р},$$

де к – кількість посад дослідників, залучених до процесу дослідження;

$М_{ni}$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$Т_р$ – число робочих днів в місяці; приблизно $Т_р$ = (21…23) дні, приймаємо 22 дні;

t_i – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 2.

Таблиця 5.2 – Витрати на заробітну плату дослідників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	42 000	1909	8	15273
Розробник	40 000	1818	20	36364
Консультанти	40 000	1818	5	9091
Всього:	60727			

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i,$$

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

t_i – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду С і можна визначити за формулою:

$$С_i = \frac{М_м \cdot К_i \cdot К_с}{Т_р \cdot t_{зм}},$$

де $М_м$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), у 2025 році $М_м$=8000 грн;

$К_i$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати, складає 1,1;

$Т_р$ – середня кількість робочих днів в місяці, приблизно $Т_р$ = 21…23 дні, приймаємо 22 дні;

$t_{зм}$ – тривалість зміни, год., приймаємо 8 год.

Таблиця 5.3 – Витрати на заробітну плату робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Підготовка інфраструктури	10	4	63,5	1,27	635
Розробка Terraform	20	5	68,5	1,37	1370
Тестування	10	4	63,5	1,27	635
Всього					2640

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$З_д = 0{,}1 \cdot (З_о + З_р) = 0{,}1 \cdot (60727 + 2640) = 6337 \text{ грн.}$$

5.2.2 Відрахування на соціальні заходи.

Нарахування на заробітну плату $Н_{зп}$ розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Н_{зп} = \beta \cdot (З_о + З_р + З_д) =$$
$$= 0{,}22 \cdot (60727 + 2640 + 6337) = 15335 \text{ грн.}$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_р$ – основна заробітна плата робітників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

5.2.3 Амортизація обладнання.

Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час (чи для) виконання даного етапу роботи.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц_б}{Т_в} \cdot \frac{t}{12},$$

де $Ц_б$ – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.;

t – термін використання основного фонду, місяці;

$Т_в$ – термін корисного використання основного фонду, роки.

Таблиця 5.4 – Амортизаційні відрахування за видами основних фондів

Найменування	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців	Сума амортизації, грн.
Ноутбук	25 000	3	12	8333,3
Всього	8333,3			

5.2.4 Витрати на електроенергію для науково-виробничих цілей.

Витрати на силову електроенергію $В_е$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

Таблиця 5.5 – Витрати на електроенергію

Найменування обладнання	Потужність, кВт	Тривалість годин роботи
Ноутбук	0,06	140

$$\text{Ве} = \sum \frac{W_i \cdot t_i \cdot \text{Ц}_\text{е} \cdot \text{К}_\text{впі}}{\text{ККД}} = \frac{0{,}06 \cdot 140 \cdot 4{,}32 \cdot 0{,}75}{0{,}98} = 27{,}8 \text{ грн.},$$

де W_i – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год.;

Це – вартість 1 кВт електроенергії, 4,32 грн.;

$\text{К}_\text{впі}$ – коефіцієнт використання потужності;

ККД – коефіцієнт корисної дії обладнання.

5.2.5 Інші витрати.

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

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

$$\text{Ів} = \left(\text{З}_\text{о} + \text{З}_\text{р}\right) \cdot \frac{\text{Н}_\text{ів}}{100\%} = (60727 + 2640) \cdot \frac{100}{100} = 63367{,}3 \text{ грн.},$$

де $\text{Н}_\text{ів}$ – норма нарахування за статтею «Інші витрати».

5.2.6 Накладні (загальновиробничі) витрати.

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на

підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…200% від суми основної заробітної плати дослідників та робітників за формулою:

$$\text{Внзв} = \left(З_о + З_р\right) \cdot \frac{Н_{нзв}}{100\%} = (60727 + 2640) \cdot \frac{200}{100} = 126734{,}55 \text{ грн.},$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

5.2.7 Витрати на проведення розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання .

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = 60727 + 2640 + 6337 + 15335 + 8333{,}3 + 27{,}8 + 63367{,}3 + 126734{,}5 = 283501{,}8 \text{ грн.}$$

5.2.8 Загальні витрати.

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи з розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = \frac{283501{,}8}{0{,}5} = 567003{,}6 \text{ грн.},$$

де η – коефіцієнт, що характеризує етап виконання науково-дослідної роботи. Оскільки, якщо науково-технічна розробка знаходиться на стадії розробки дослідного зразка, то η=0,5.

5.3 Розрахунок економічної ефективності науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання , є збільшення у потенційного інвестора величини чистого прибутку.

В даному випадку відбувається розробка засобу, тому основу майбутнього економічного ефекту буде формувати: ΔN – збільшення кількості споживачів, яким надається відповідна інформаційна послуга в аналізовані періоди часу; N – кількість споживачів, яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки; $Ц_б$ – вартість послуги у році до впровадження інформаційної системи; $\pm\Delta Ц_о$ – зміна вартості послуги (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу.

Можливе збільшення чистого прибутку у потенційного інвестора ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N_i)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right),$$

де $\pm\Delta Ц_о$ може мати як додатне, так і від'ємне значення (від'ємне – при зниженні ціни відносно року до впровадження цієї розробки, додатне – при зростанні ціни), 25000 грн.;

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки, 125 шт.;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, 15000 грн.;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів, 40000 грн.;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році0 шт., 20 шт., 45 шт..

Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки); λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість.

У 2025 році ставка податку на додану вартість становить 20%, а коефіцієнт λ =0,8333; ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ =0,2...0,5; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2025 році ϑ =18%.

Очікуваний термін життєвого циклу розробки 3 роки, тому:

1-й рік: $\Delta П1$=640369 грн., 2-й рік: $\Delta П2$=701844 грн., 3-й рік: $\Delta П3$=778688грн.

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки методу аварійного відновлення додатків

у хмарному середовищі з використанням гібридної архітектури розгортання:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} = \frac{640369}{(1+0{,}1)^1} + \frac{701844}{(1+0{,}1)^2} + \frac{778688}{(1+0{,}1)^3} = 1747230{,}02 \text{ грн.},$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн.;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки (приймаємо Т=3 роки);

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

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

Далі розраховують величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання . Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot ЗВ = 2 \cdot 567003{,}606 = 1134007 \text{ грн.},$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання та її комерціалізацію.

Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай кінв=1...5, але може бути і більшим; ЗВ – загальні витрати на проведення науково-технічної розробки та

оформлення її результатів, грн.

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання становитиме:

$$Е_{абс} = ПП - PV = 1747230{,}02 - 1134007 = 613223 \text{ грн.},$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання , грн.;

PV – теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс}>0$, то можемо припустити про потенційну зацікавленість у розробці методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання .

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність Ев або показник внутрішньої норми дохідності вкладених інвестицій та порівняти її з так званою бар’єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання вкладати буде економічно недоцільно.

Внутрішня економічна дохідність інвестицій Ев, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання , розраховується за формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{Еабс}{PV}} = \sqrt[з]{1 + \frac{613223}{1134007}} = 0{,}51,$$

де $Т_ж$ – життєвий цикл розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання , роки.

Далі розраховуємо період окупності інвестицій $Т_о$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання :

$$Т_о = \frac{1}{E_в} = \frac{1}{0{,}51} = 1{,}95 \text{ роки.}$$

Оскільки $Т_о$<1…3-х років, то це свідчить про комерційну привабливість науково-технічної розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання і може спонукати потенційного інвестора профінансувати впровадження цієї розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання та виведення її на ринок.

Проведений комерційний та технологічний аудит розробки методу аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання підтвердив її доцільність та перспективність. За результатами оцінювання трьома незалежними експертами науково-технічний рівень і комерційний потенціал розробки визначено як вищий за середній (середньоарифметична оцінка – 31 бал), що свідчить про достатню технічну зрілість концепції та її конкурентоспроможність на ринку.

Розрахунок витрат показав, що загальна вартість виконання науково-дослідної

роботи становить 5670,0 тис. грн (з урахуванням коефіцієнта етапу виконання). Подальший аналіз економічної ефективності засвідчив позитивний фінансовий результат можливої комерціалізації: чистий приведений дохід становить 613,2 тис. грн, а період окупності інвестицій – лише 1,95 року, що менше за типовий допустимий діапазон 1–3 роки. Внутрішня норма дохідності (0,51) також перевищує мінімально прийнятний рівень.

Отже, розроблений метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання є економічно вигідним, технічно здійсненним і комерційно привабливим для потенційного інвестора. Це підтверджує доцільність його подальшого впровадження та просування на ринку сучасних IT-рішень для забезпечення безперервності бізнес-процесів.

ВИСНОВКИ

В магістерській кваліфікаційній роботі проведено аналіз сучасних технологій розгортання систем у хмарних середовищах, які забезпечують доступність вебдодатків, мають високу гнучкість та масштабованість. Однак, вони мають високі фінансові витрати в довгостроковій перспективі.

Проаналізовано технологої розгортання на локальнуму сервері та визначено причини можливих аварій інфраструктури у сучасних умовах, серед яких основними є вимкнення електроенергії, апаратні збої, кібератаки та людські помилки.

Визначено об’єкти аварійного відновлення якими є база даних та вебсервіс.

Проведено аналіз існуючих хмарних сервісів аварійного відновлення, зокрема AWS Elastic Disaster Recovery та Azure Site Recovery, який показав, що при забезпеченні повного відновлення даних і працездатності серверів, їх використання потребує значних часових і фінансових ресурсів.

В роботі розроблено метод автоматичного аварійного відновлення вебдодатків, який передбачає моніторинг працездатності локального середовища, автоматичне переключення трафіку на резервну хмарну інфраструктуру та синхронізацію критичних даних між середовищами. Метод забезпечує підвищення стабільності та зменшення часу недоступності вебсервісів шляхом застосування гібридної архітектури розгортання додатків у хмарному середовищі AWS.

Розроблено «гарячу» модифікацію методу, яка використовує сервіс AWS Database Migration Service для постійної реплікації даних на відміну від початкового варіанту, у якому метод базується на періодичному експорті критичних таблиць у сховище Amazon S3.

Виконано розгортання вебдодатку .NET Core та СУБД PostgreSQL у хмарному середовищі AWS. Створено високодоступну інфраструктуру з використанням сервісів Amazon RDS для розміщення бази даних, Amazon S3 для зберігання резервних копій, AWS Lambda для автоматизації процесів резервування та

відновлення, SNS для надсилання сповіщень, Route 53 Health Check для моніторингу працездатності сервісів і CloudWatch Alarm для виявлення збоїв і запуску алгоритму відновлення. Усі компоненти інфраструктури створено з використанням підходу Infrastructure as Code засобами Terraform, що забезпечує повторюваність, керованість і можливість швидкого розгортання рішень у різних середовищах.

Проведене тестування розробленого методу показало, що середній час відновлення системи становить близько 7 хвилин при допустимій втраті даних до 5 хвилин. При використані модифікації методу з «гарячою» базою даних час відновлення системи — близько 1,5 хвилини при майже нульовій втраті даних. Це забезпечує підвищення рівня доступності SLA з 99,99 % до 99,9999 %, що відповідає вимогам до систем високої доступності.

Запропонований підхід може бути використаний для побудови високодоступних систем малого та середнього бізнесу, а також як основа для подальших досліджень у сфері керування процесами аварійного відновлення в хмарних середовищах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Марценюк Д. В., Войцеховський О. В., Войцеховська О. В. Метод аварійного відновлення вебдодатків у хмарному середовищі AWS // Матеріали міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» (Вінниця, 2026 р.) : збірник доповідей [Електронний ресурс]. Вінниця: ВНТУ, 2026. — Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/viewFile/26501/21841. (дата звернення: 11.28.25).

2. Disaster Recovery: кому який варіант підходить — [Електронний ресурс]. — Режим доступу: https://ko.com.ua/disaster_recovery_yakij_dlya_chogo_144160 (дата звернення: 10.09.2025).

3. Що таке IT-інфраструктура — [Електронний ресурс]. — Режим доступу: https://cases.media/en/article/sho-take-it-infrastruktura-yakoyu-vona-buvaye-i-z-chogo-skladayetsya (дата звернення: 10.09.2025).

4. Довгий, С. О., Копійка, О. В. IT-інфраструктура як базова складова цифрової трансформації: монографія. — [Електронний ресурс]. — Режим доступу: itgip.org/wp-content/uploads/2023/05/monografiya3.pdf (дата звернення: 15.09.2025).

5. Mell P., Grance T. The NIST Definition of Cloud Computing (NIST Special Publication 800-145). — Gaithersburg, MD : NIST, 2011. — 7 p.

6. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection — Information security management systems — Requirements. — Geneva : ISO, 2022.

7. VMware. On-Premises vs Cloud Deployment Models. — [Електронний ресурс]. — Режим доступу: https://www.vmware.com (дата звернення: 20.09.2025).

8. Microsoft. Cloud Computing Models: SaaS, PaaS, IaaS. — [Електронний ресурс]. — Режим доступу: https://azure.microsoft.com (дата звернення: 25.09.2025).

9. Amazon Web Services. AWS Well-Architected Framework. — [Електронний

ресурс]. — Режим доступу: https://aws.amazon.com/architecture/well-architected (дата звернення: 28.09.2025).

10. Amazon Web Services. Elastic Disaster Recovery (DRS) User Guide. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/drs (дата звернення: 28.09.2025).

11. Amazon Web Services. Amazon Route 53 Documentation. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/route53 (дата звернення: 06.10.2025).

12. Amazon Web Services. AWS Lambda Developer Guide. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/lambda (дата звернення: 06.10.2025).

13. Amazon Web Services. Amazon Simple Notification Service (SNS) Documentation. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/sns (дата звернення: 06.10.2025).

14. Amazon Web Services. AWS Database Migration Service User Guide. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/dms (дата звернення: 10.10.2025).

15. Amazon Web Services. Amazon S3 User Guide. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/s3 (дата звернення: 10.10.2025).

16. AWS Architecture Blog: Pilot Light та Warm Standby. — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-iii-pilot-light-and-warm-standby/? (дата звернення: 15.10.2025).

17. The role of RPO and RTO in disaster recovery planning. — [Електронний ресурс]. — Режим доступу: https://irek.ase.md/xmlui/handle/123456789/2074 (дата звернення: 15.10.2025).

18. Tencent Cloud Techpedia – “How to measure detection delay and repair time in equipment risk identification. — [Електронний ресурс]. — Режим доступу:

https://www.tencentcloud.com/techpedia/125791 (дата звернення: 15.10.2025)

19. HashiCorp. Terraform Documentation. — [Електронний ресурс]. — Режим доступу: https://developer.hashicorp.com/terraform/docs (дата звернення: 18.10.2025).

20. AWS DMS — Ongoing replication. — [Електронний ресурс]. — Режим доступу:https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Task.CDC.html?utm_source (дата звернення: 18.10.2025).

21. Amazon Web Services (AWS): Benefits of Cloud Infrastructure. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/six-advantages-of-cloud-computing.html (дата звернення: 20.10.2025).

22. Amazon Relational Database Service (RDS). AWS Documentation. — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html (дата звернення: 20.10.2025).

23. Understanding Automated Backups with Amazon RDS. AWS Whitepaper. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html (дата звернення: 22.10.2025).

24. HashiCorp. Terraform: Infrastructure as Code. [Електронний ресурс]. — Режим доступу: https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code (дата звернення: 25.10.2025).

25. Deploying Web Applications with AWS Elastic Beanstalk. AWS Documentation. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html (дата звернення: 30.10.2025).

26. Amazon Simple Storage Service (S3). AWS Documentation. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/s3/ (дата звернення: 30.10.2025).

27. Building Decoupled Applications with Amazon Simple Notification Service (SNS). AWS Documentation. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/sns/latest/dg/sns-system-to-system-messaging.html (дата звернення: 02.11.2025).

28. Configuring Health Checks in Amazon Route 53. AWS Documentation. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html (дата звернення: 02.11.2025).

29. Serverless Compute with AWS Lambda. AWS Documentation. [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/lambda/(дата звернення: 05.11.2025).

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. О. Д. Азаров
«03» жовтня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи
«Метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання»

Науковий керівник: к.т.н., доц. каф. ОТ
_____ Войцеховська О.В.

Студент групи 2КІ-24м,
_____Марценюк Д.В.

Вінниця 2025

1 Підстава виконання магістерської кваліфікаційної роботи

1.1 Актуальність розробки полягає у потребі мінімізації ризиків втрати даних і часу простою в разі технічних збоїв чи кібератак шляхом впровадження методу аварійного відновлення додатків у хмарному середовищі.

1.2 Наказ про затвердження теми МКР

2 Мета і призначенням МКР

2.1 Метою дослідження є підвищення стабільності та зменшення часу недоступності web-сервісів під час непередбачуваних ситуацій, шляхом використання гібридної архітектури розгортання додатків у хмарному середовищі Amazon Web Services.

2.2 Призначення розробки полягає у створенні та впровадженні методу автоматизованого аварійного відновлення вебдодатків у хмарному середовищі Amazon Web Services з використанням гібридної архітектури розгортання.

3 Вихідні дані для виконання МКР

3.1 Проведення огляду технологій аварійного відновлення інформаційної інфраструктури.

3.1 Розробка методу аварійного відновлення додатків у хмарному середовищі.

3.2 Практична реалізація методу аварійного відновлення.

3.3 Тестування методу аварійного відновлення.

3.4 Виконання розрахунків для доведення доцільності нової розробки.

4 Технічні вимоги до виконання МКР

МКР повинна задовольняти таку вимогу — реалізація методу автоматичного аварійного відновлення вебдодатків у хмарному середовищі Amazon Web Services з використанням гібридної архітектури розгортання.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в табл. А.1.

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання	Очікувані результати
1	Постановка мети та задач роботи	08.09.2025	Аналітичний огляд джерел, задачі досліджень
2	Аналітичний огляд технологій аварійного відновлення інформаційної інфраструктури	11.09.2025	Розділ 1
3	Розробка методу аварійного відновлення додатків у хмарному середовищі	01.10.2025	Розділ 2
4	Практична реалізація методу аварійного відновлення	21.10.2025	Розділ 3
5	Тестування методу аварійного відновлення	24.10.2025	Розділ 4
6	Підготовка економічної частини	03.11.2025	Розділ 5
7	Оформлення пояснювальної записки, графічного матеріалу та презентації	17.11.2025	ПЗ, графічний матеріал і презентація

6 Матеріали, що подаються до захисту МКР

До захисту МКР подаються: пояснювальна записка МКР, ілюстративні та графічні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР.

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається

на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— Методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп'ютерна інженерія». Кафедра обчислювальної техніки ВНТУ 2022.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод аварійного відновлення додатків у хмарному середовищі з використанням гібридної архітектури розгортання _

Тип роботи: _____ магістерська кваліфікаційна робота _____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) __2,5_____%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., д.т.н., проф., зав. кафедри ОТ_ _____
(прізвище, ініціали, посада) (підпис)

_____Мартинюк Т. Б., д.т.н., проф. кафедри ОТ __ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С. М., проф. каф. ОТ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____Войцеховська О. В., доц. каф. ОТ____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Марценюк Д. В._____
(підпис) (прізвище, ініціали)

ДОДАТОК В

Схема архітектури методу аварійного відновлення додатку

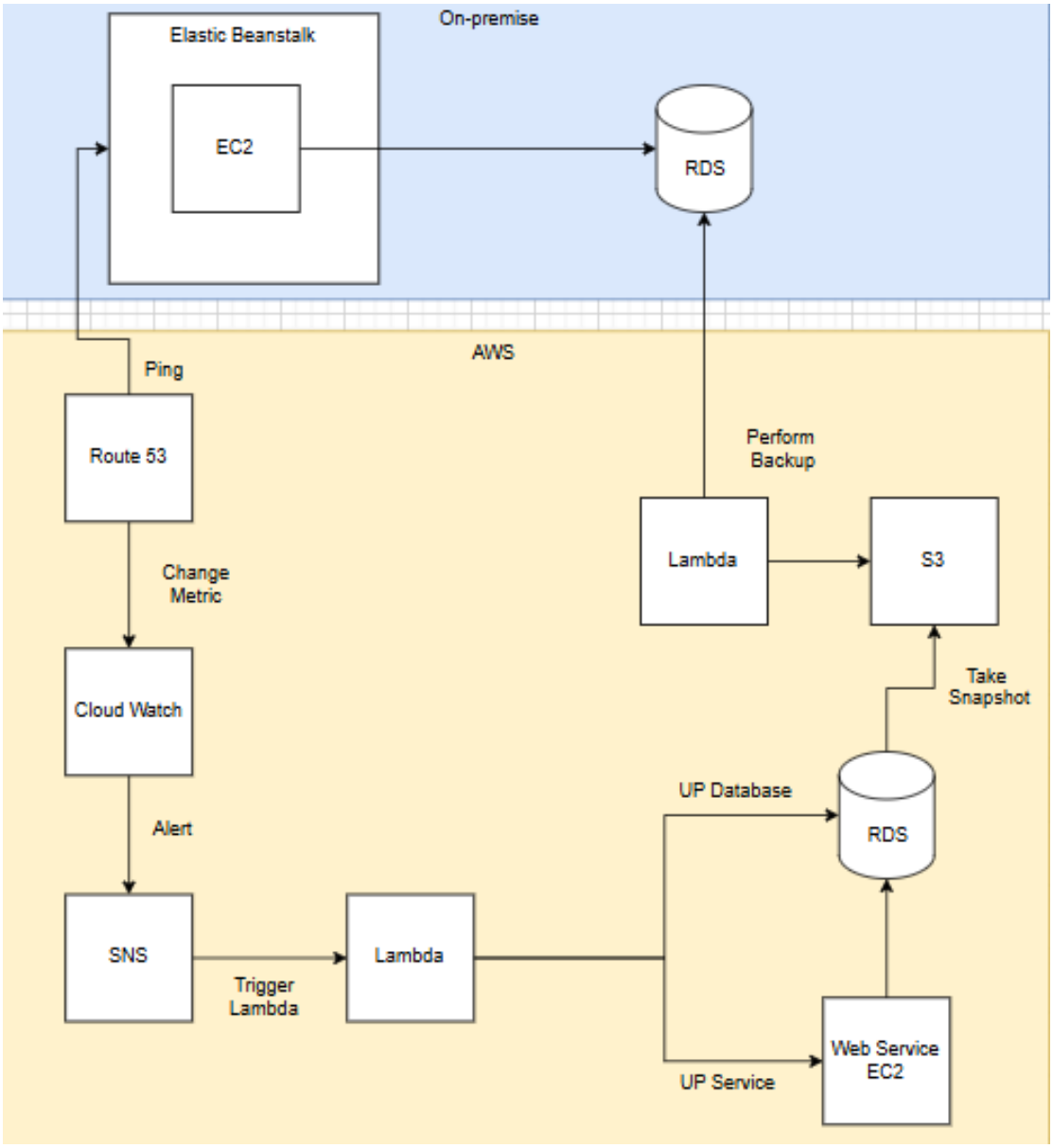

Рисунок В.1 — Схема архітектури методу аварійного відновлення додатку

ДОДАТОК Г

Схема архітектури методу аварійного відновлення з використанням DMS

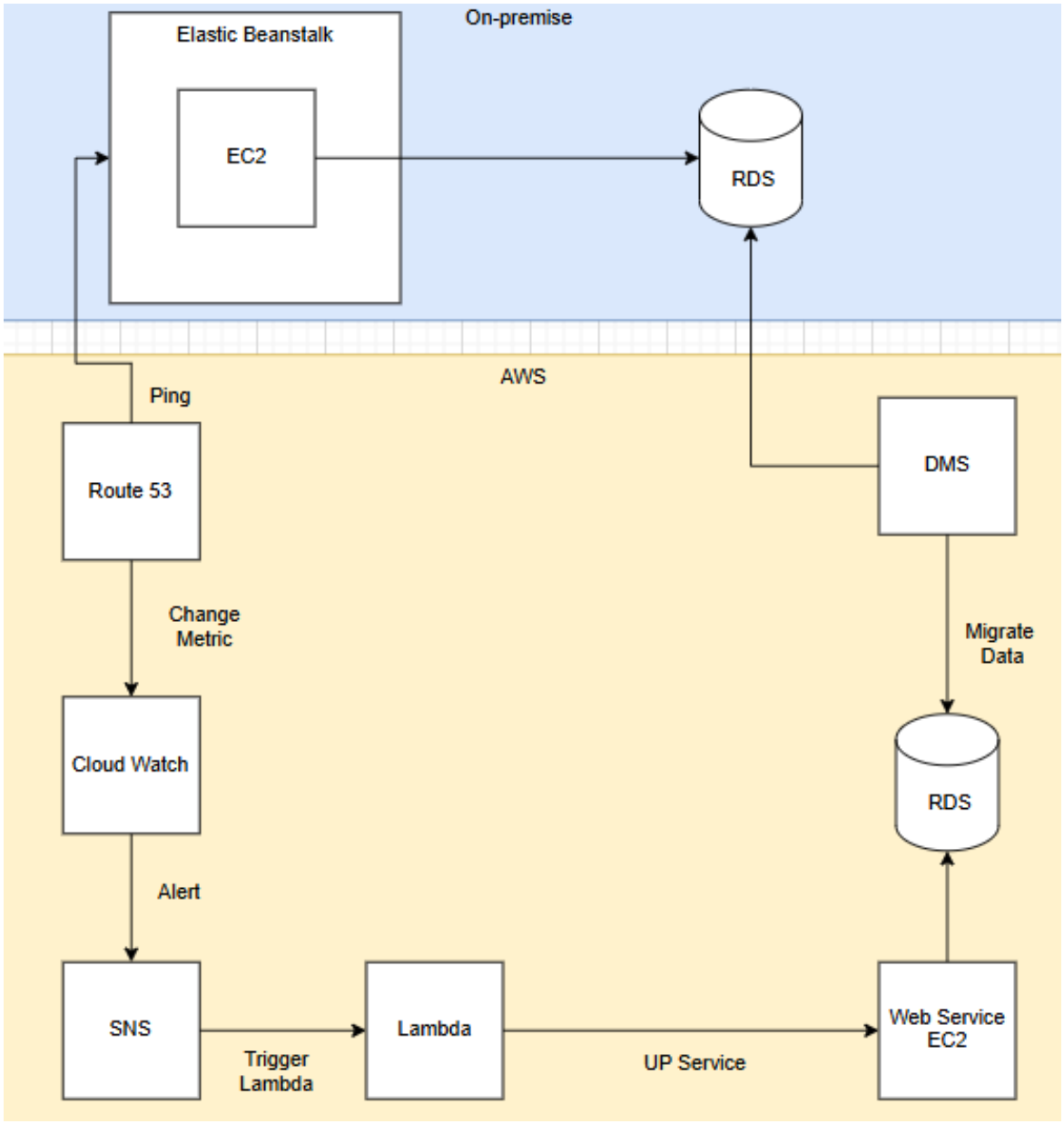

Рисунок Г.1 — Схема архітектури методу аварійного відновлення з використанням DMS

ДОДАТОК Д

Блок-схема алгоритму роботи методу автоматичного аварійного відновлення додатку

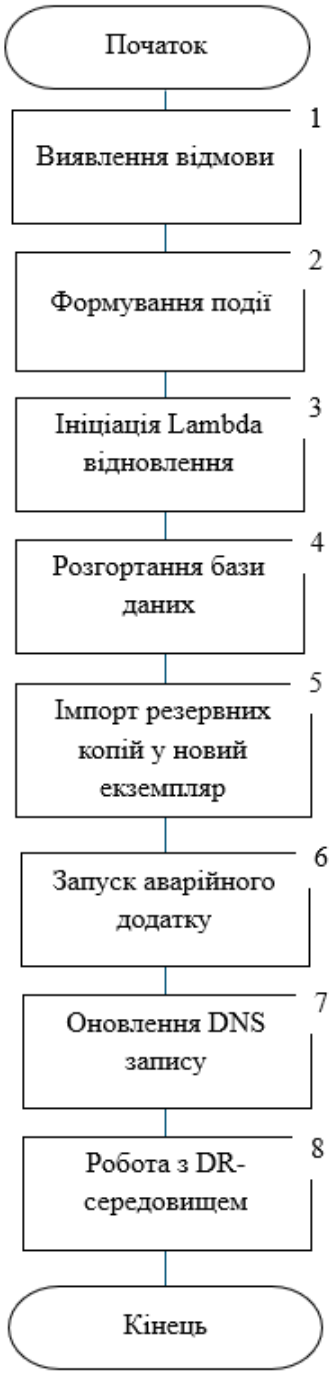

Рисунок Д.1 — Блок-схема алгоритму роботи методу автоматичного аварійного відновлення додатку

ДОДАТОК Е

Блок-схема алгоритму роботи модифікації методу автоматичного аварійного відновлення додатку з використанням «гарячої» бази даних

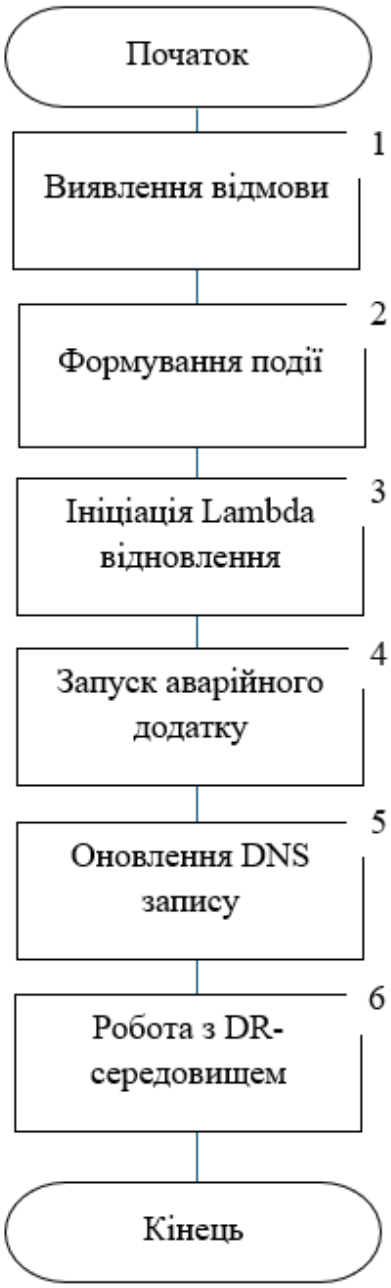

Рисунок Е.1 — Блок-схема алгоритму роботи модифікації методу автоматичного аварійного відновлення додатку з використанням «гарячої» бази даних

ДОДАТОК Ж

Вікно створення Lambda функції для відновлення додатку

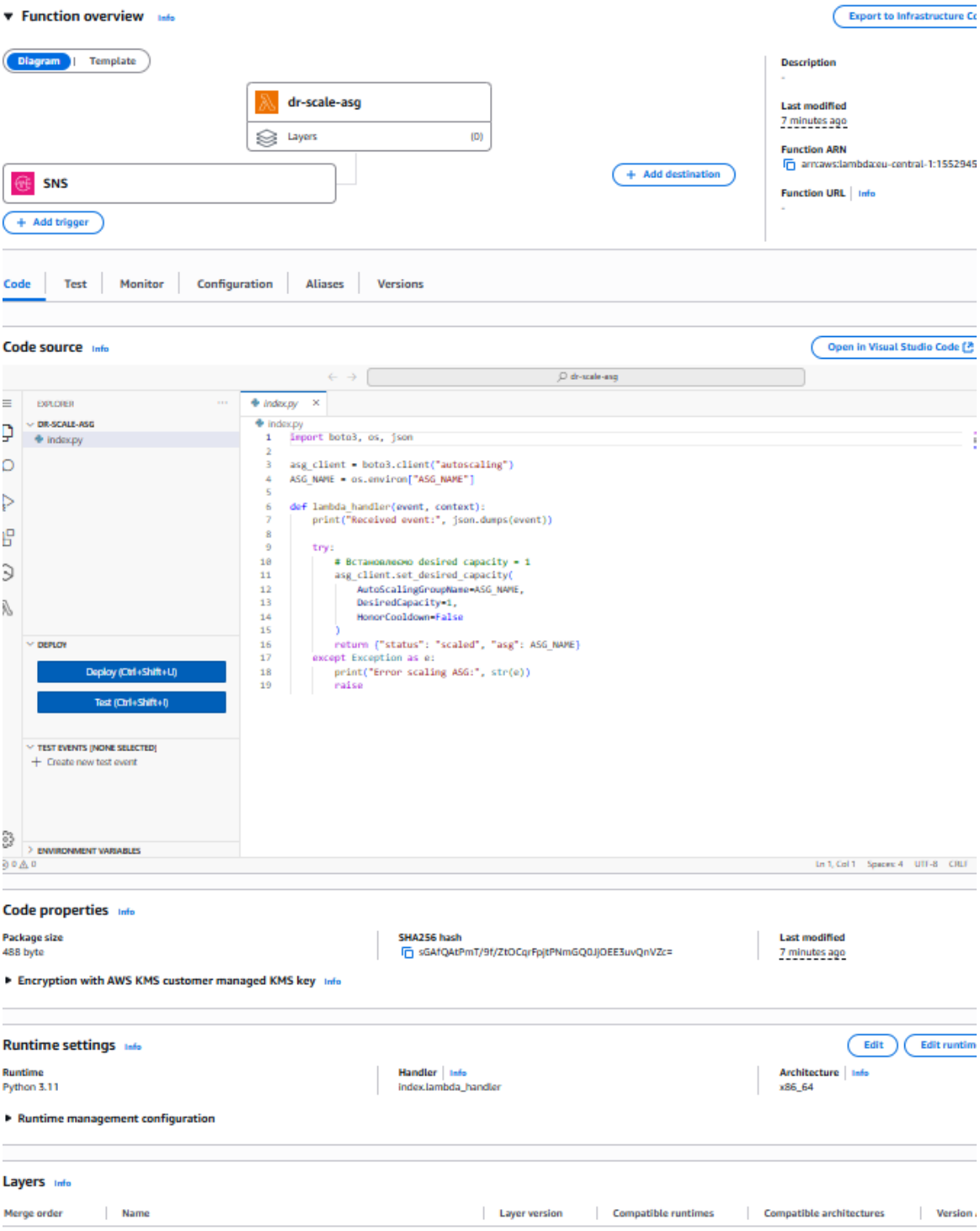

Рисунок Ж.1 — Вигляд вікна створення Lambda функції для аварійного відновлення додатку

ДОДАТОК И

Лістинги скриптів для розгортання хмарної інфраструктури

Лістинг И.1 — Lambda функціїя для резервування бази даних інфраструктуру

```
import os, json, time
import boto3
import pg8000
ASG_NAME = "awseb-e-iatrrtkufd-stack-AWSEBAutoScalingGroup-
98ZDKVNgexFd"
# DR RDS (знаходимо endpoint по ідентифікатору)
DR_DB_IDENTIFIER = "vntu-db-dr"
DB_NAME = "postgres"
DB_USER = "masteruser"
DB_PASSWORD = "admin123!"  # додається в env змінні Lambda
# S3 з CSV-бекапами (export з aws_s3.query_export_to_s3)
S3_REGION = "eu-central-1"
S3_BUCKET = "vntu-db-backup"
S3_PREFIX = "exports"  # файли типу:
exports/AspNetRoles_YYYYMMDD_HHMMSS.csv
# Таблиці для відновлення: file_prefix має відповідати іменам файлів
TABLES = [
   {"table": 'public."AspNetRoles"', "file_prefix": "public_AspNetRoles_"},
   {"table": 'public."AspNetUsers"', "file_prefix": "public_AspNetUsers_"},
   {"table": 'public."AspNetRoleClaims"', "file_prefix":
"public_AspNetRoleClaims_"},
   {"table": 'public."AspNetUserClaims"', "file_prefix":
"public_AspNetUserClaims_"},
   {"table": 'public."AspNetUserLogins"', "file_prefix":
"public_AspNetUserLogins_"},
   {"table": 'public."AspNetUserRoles"', "file_prefix":
"public_AspNetUserRoles_"},
   {"table": 'public."Disciplines"', "file_prefix": "public_Disciplines_"},
   {"table": 'public."Lesson"', "file_prefix": "public_Lesson_"},
   {"table": 'public."Students"', "file_prefix": "public_Students_"},
   {"table": 'public."Subscriptions"', "file_prefix": "public_Subscriptions_"},
   {"table": 'public."Test"', "file_prefix": "public_Test_"},
   {"table": 'public."TestQuestion"', "file_prefix": "public_TestQuestion_"},
   {"table": 'public."TestAnswer"', "file_prefix": "public_TestAnswer_"},
   {"table": 'public."TestTicket"', "file_prefix": "public_TestTicket_"},]
GZIP_FILES = False
```

```
# ============================================
asg = boto3.client("autoscaling")
rds = boto3.client("rds")
s3  = boto3.client("s3")
def _get_db_endpoint(db_identifier: str):
resp = rds.describe_db_instances(DBInstanceIdentifier=db_identifier)
inst = resp["DBInstances"][0]
return inst["Endpoint"]["Address"], inst["DBInstanceStatus"]
def _wait_db_available(db_identifier: str, timeout_sec=900):
start = time.time()
while True:
_, status = _get_db_endpoint(db_identifier)
if status == "available":
return
if time.time() - start > timeout_sec:
raise TimeoutError(f"DB {db_identifier} not available after {timeout_sec}s
(status={status})")
time.sleep(10)
def _latest_csv_key(prefix: str) -> str: """
Знаходимо найсвіжіший CSV під s3://S3_BUCKET/S3_PREFIX/{prefix}..."""
full_prefix = f"{S3_PREFIX}/{prefix}"
paginator = s3.get_paginator("list_objects_v2")
latest = None
for page in paginator.paginate(Bucket=S3_BUCKET, Prefix=full_prefix):
for obj in page.get("Contents", []):
key = obj["Key"]
if not (key.lower().endswith(".csv") or key.lower().endswith(".csv.gz")):
continue
if latest is None or obj["LastModified"] > latest["LastModified"]:
latest = obj
if not latest:
raise RuntimeError(f"No CSV files under s3://{S3_BUCKET}/{full_prefix}")
return latest["Key"]
def _import_table(conn, table_qualified: str, key: str):"""
TRUNCATE → aws_s3.table_import_from_s3 до таблиці."""
opts = ["format csv", "header true"]
if GZIP_FILES or key.lower().endswith(".gz"):
opts.append("gzip true")
copy_opts = "(" + ", ".join(opts) + ")"
cur = conn.cursor()
# очистка таблиці перед імпортом
cur.execute(f'TRUNCATE TABLE {table_qualified} CASCADE;')
```

```
sql = """
SELECT aws_s3.table_import_from_s3(
%s, %s, %s,
aws_commons.create_s3_uri(%s, %s, %s));"""
cur.execute(sql, (table_qualified, '', copy_opts, S3_BUCKET, key, S3_REGION))
def lambda_handler(event, context):
print("Received event:", json.dumps(event))
# 0) переконаймося, що DR RDS доступний (якщо stopped — стартуємо)
endpoint, status = _get_db_endpoint(DR_DB_IDENTIFIER)
print(f"DR DB: endpoint={endpoint}, status={status}")
if status == "stopped":
print("Starting DR DB instance...")
rds.start_db_instance(DBInstanceIdentifier=DR_DB_IDENTIFIER)
_wait_db_available(DR_DB_IDENTIFIER)
endpoint, status = _get_db_endpoint(DR_DB_IDENTIFIER)
print(f"DR DB started. status={status}")
if status != "available":
raise RuntimeError(f"DR DB is not available (status={status})")
```

Лістинг И.2 — Скрипт для створення канала повідомлень SNS

```
terraform {
required_version = ">= 1.5.0"
required_providers {
aws = {
source  = "hashicorp/aws"
version = "~> 5.0"}}}
provider "aws" {
region = var.region}
variable "region" {
description = "AWS region"
type        = string
default     = "eu-central-1"}
variable "sns_topic_name" {
description = "SNS DR topic"
type        = string
default     = "dr-route53-alerts"}
resource "aws_sns_topic" "dr_alerts" {
name = var.sns_topic_name}
output "sns_topic_arn" {
value       = aws_sns_topic.dr_alerts.arn
description = "SNS topic For DR"}
```

Лістинг И.3 — Скрипт для створення Route 53

```
required_version = ">= 1.5.0"
required_providers {
aws = {
source = "hashicorp/aws"
version = "~> 5.0" } } }
provider "aws" { region = var.region }
variable "region" {
type       = string
default    = "us-east-1"
variable "target_fqdn" {
type       = string
default = "vntu-local-app-test-dev.eba-3pi28mzv.eu-central1.elasticbeanstalk.com"}
variable "protocol" {
type       = string
default    = "HTTP"}
variable "port" {
type       = number
default    = 80}
variable "resource_path" {
type       = string
default    = "/"}
variable "request_interval_seconds" {
type       = number
default    = 30}
variable "failure_threshold" {
type       = number
default    = 3}
resource "aws_route53_health_check" "app_hc" {
type              = var.protocol
fqdn              = var.target_fqdn
port              = var.port
resource_path     = var.resource_path
request_interval  = var.request_interval_seconds
failure_threshold = var.failure_threshold
measure_latency   = true
regions           =  ["us-east-1", "us-west-1", "us-west-2"]
enable_sni = var.protocol == "HTTPS" ? true : null
tags = {Name = "dr-hc-${var.target_fqdn}"}}
```

Лістинг И.4 — Скрипт для створення Cloud Watch

```
terraform {
required_version = ">= 1.5.0"
required_providers {
aws = { source = "hashicorp/aws", version = "~> 5.0" }}}
provider "aws" {
region = var.region}
provider "aws" {
alias  = "us_east_1"
region = "us-east-1"}
variable "region" {
type        = string
default     = "eu-central-1"}
resource "aws_cloudwatch_metric_alarm" "hc_alarm" {
provider            = aws.us_east_1
alarm_name          = "dr-hc-unhealthy"
alarm_description   = "Route53 health check is UNHEALTHY"
namespace           = "AWS/Route53"
metric_name         = "HealthCheckStatus"
statistic           = "Minimum"
period              = 30
evaluation_periods  = 1
threshold           = 1
comparison_operator = "LessThanThreshold"
dimensions = {
HealthCheckId = "29d19494-9494-4767-a98e-b9497d9e8531"}
alarm_actions           = ["arn:aws:sns:us-east-1:155294525926:alerts"]
ok_actions              = ["arn:aws:sns:us-east-1:155294525926:alerts"]
insufficient_data_actions = ["arn:aws:sns:us-east-1:155294525926:alerts"]}
```

Лістинг И.5 — Скрипт створення Lambda функції аварійного відновлення

```
conn = pg8000.connect(
host=endpoint, port=5432, database=DB_NAME,
user=DB_USER, password=DB_PASSWORD, ssl_context=True)
try: conn.autocommit = True
cur = conn.cursor()
cur.execute("CREATE EXTENSION IF NOT EXISTS aws_commons;")
cur.execute("CREATE EXTENSION IF NOT EXISTS aws_s3;")
imported = []
```

```
for item in TABLES:
table = item["table"]
key = _latest_csv_key(item["file_prefix"])
print(f"Import {table} from s3://{S3_BUCKET}/{key}")
_import_table(conn, table, key)
imported.append({"table": table, "s3_key": key})
#asg.set_desired_capacity(
AutoScalingGroupName=ASG_NAME,
DesiredCapacity=1,
HonorCooldown=False)
return {"status": "ok", "db_endpoint": endpoint, "imported": imported, "asg":
ASG_NAME}
finally:
conn.close()
```