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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ КОНВЕЄРНОЇ ЗБІРКИ ДИНАМІЧНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩ

Виконав студент 2 курсу, групи 1КІ–22м
спеціальності 123 — Комп'ютерна інженерія
Гуменюк О. В.
Керівник к.т.н., проф. каф. ОТ
Захарченко С. М
" 7 " грудня 2023р.
Опонент к.т.н., доц. зав. каф. МБІС
Карпінець В.В.
" 5 " грудня 2023р.

Допущено до захисту
д.т.н., проф. Азаров О.Д.
"11" 12 2023 р.

ВНТУ 2023

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Галузь знань — Інформаційні технології

Освітній рівень — магістр

Спеціальність — 123 Комп'ютерна інженерія

Освітньо–професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"26" вересня 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту **Гуменюку Олександру Володимировичу**

1 Тема роботи «Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ» керівник роботи Захарченко Сергій Михайлович професор, затверджено наказом вищого навчального закладу від 18.09.2023 року № 247.

2 Строк подання студентом роботи 09.12.23р.

3 Вихідні дані до роботи: ресурси розташовані у хмарному сервісі, горизонтальне масштабування за обсягом оперативної пам'яті та потужністю центрального процесора, обмеження по бюджету – 100 USD , максимальна кількість віртуальних обчислювальних машин – 10

4 Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз методів конвеєрної збірки динамічних

обчислювальних середовищ, дослідження методів збірки динамічних обчислювальних середовищ, розробка технології збірки динамічних обчислювальних середовищ, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): алгоритм горизонтального масштабування за пороговим значенням оперативної пам'яті, алгоритм горизонтального масштабування за пороговим значенням центрального процесора, алгоритм вертикального масштабування, відображення вартості динамічного обчислювального середовища.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1— Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Захарченко Сергій Михайлович проф.	1.09.2023	1.09.2023
4	Небава Микола Іванович проф., к.е.н	1.11.2023	1.11.2023

7 Дата видачі завдання 19.09.2023р.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	10.09.2023	Вик.
2	Огляд існуючих рішень	25.09.2023	Вик.
3	Дослідження технологій динамічних обчислювальних середовищ	5.10.2023	Вик.
4	Розробка технології динамічного обчислювального середовища на основі Microsoft Azure	12.10.2023	Вик.

Продовження таблиці 2

5	Розрахунок економічної частини	19.10.2023	Вик.
6	Оформлення пояснювальної записки	26.10.2023	Вик.
7	Виконання магістерської кваліфікаційної роботи	2.11.2023	Вик.
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	9.11.2023	Вик.
9	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	16.11.2023	Вик.
10	Перевірка «Антиплагіат»	23.11.2023	Вик.
11	Попередній захист	30.11.2023	Вик.

Студент _____ Гуменюк Олександр Володимирович

Керівник _____ проф. Захарченко Сергій Михайлович

АНОТАЦІЯ

УДК 004.75

Гуменюк О.В. Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2023 — 130с. На укр. мові. Бібліогр.: 36 назв; рис.: 29; табл. 9.

У роботі розглянуто технології реалізації конвеєрної збірки динамічних обчислювальних середовищ. Проведений аналіз сучасних підходів для побудови конвеєрної збірки, розглянуто різних підрядників, вибрано оптимальну технологію програмного побудування динамічного обчислювального середовища. Розроблено програмне рішення, з використанням хмарних конвеєрів збірок і без серверних програмних функцій.

Ключові слова: конвеєрна збірка, автоматичне масштабування, середовище розгортання, інфраструктура як код.

ABSTRACT

УДК 004.75

Humeniuk Oleksandr. Pipelines implementation technologies of dynamic computing environments. Master's thesis in the specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2023. In Ukrainian language. Bibliographer: 36 titles; fig.: 29; tabl. 9.

The paper considers the pipelines implementation technologies of dynamic computing environments. It contains the analysis of modern approaches to the construction of a pipelines, various cloud providers were considered, and the optimal technology for software construction of a dynamic computing environment was selected. A software solution has been developed, using cloud assembly pipelines and serverless cloud functions.

Key words: pipeline, autoscaling, computing environment, infrastructure as a code.

ЗМІСТ

					08–51МКР.004.00.000 ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Гуменюк О.В.		06.12.23	ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ КОНВЕЄРНОЇ ЗБІРКИ ДИНАМІЧНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩ ПОЯСНЮВАЛЬНА ЗАПИСКА	Літ.	Арк. 7 / Аркушів 130
Перевір.		Захарченко С.М.		07.12.23			
Опонент		Карпінець В.В.		08.12		ВНТУ, гр1КІ–22м	
Н. Контр.		Швець С.І.		22.12.23			
Затверд.		Азаров О.Д		27.12.23			

ВСТУП

Розробка ефективних методів конвеєрної збірки динамічних обчислювальних середовищ є **актуальною** задачею для сучасних науковців із усього світу.

Сьогодні, для продовження роботи над науковими відкриттями та розробками, необхідно аналізувати величезні масиви даних. Під час даного процесу виникає дві проблеми: час, необхідний для виконання аналізу та бюджет наукового дослідження.

Для того, щоб пришвидшити процес аналізу необхідно мати дуже потужні обчислювальні середовища, які мають велику вартість. І навпаки, для того щоб зменшити вартість аналізу отриманих результатів, можливо проводити аналіз з використанням менших потужностей. Відповідно, необхідно знайти баланс між цими двома характеристиками.

Придбання і встановлення обчислювальної техніки, її налаштування і підтримка в працюючому стані також вимагають багато часу, персоналу і коштів.

Метою даної роботи є аналіз і розробка ефективних методів конвеєрної збірки обчислювальних середовищ, які б могли б бути швидко створені, налаштовані і запущені. При цьому кількість обчислювальних ресурсів, повинна масштабуватись в залежності від навантаження. В ідеальних умовах, коли середовище не виконує ніяких завдань, його вартість повинна прямувати до нуля.

Для досягнення мети пропонується вирішити такі завдання:

— вибрати платформу, яка підтримує програмне створення динамічних обчислювальних середовищ;

— вибрати технології створення та налаштування конвеєрної збірки;

— налаштувати конвеєрну збірку;

— розробити каркас динамічного обчислювального середовища;

— налаштувати масштабування;

— проаналізувати час і кошти, необхідні для виконання обчислень.

Об’єктом дослідження є процес масштабування хмарних ресурсів

Предметом дослідження даної роботи є механізми масштабування хмарних обчислювальних ресурсів із використанням вертикального і горизонтального типів масштабування.

Практична цінність даної роботи полягає у виконанні обчислювальних процесів із використанням оптимальної кількості обчислювальних ресурсів і зі збереженням економічної ефективності.

Наукова новизна полягає у вдосконаленні методу масштабування обчислювальних ресурсів за рахунок уведення порогових значень, що дозволяє оптимізувати їх використання.

Апробацію результатів наукової роботи було проведено на двох наукових конференціях:

— «Молодь в науці: дослідження, проблеми, перспективи (МН–2024)», доповідь на тему “Горизонтальне масштабування хмарних обчислювальних ресурсів за допомогою порогових значень”;

— «LIII Науково–технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024)», доповідь на тему “Алгоритм масштабування хмарних обчислювальних ресурсів за допомогою порогових значень” .

1 АНАЛІЗ МЕТОДІВ КОНВЕЄРНОЇ ЗБІРКИ ДИНАМІЧНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩ

Сьогодні існує велика кількість рішень для збереження і обробки даних на віддалених серверах. Кожне із них має певний набір характеристик, які мають вплив на вибір середовища для розробки. Основними характеристиками є:

— вартість;

— час розгортання;

— здатність масштабуватись;

— швидкодія;

— надійність;

— наявність документації.

Відповідність даним характеристикам зумовлює вибір певного вибору постачальника віддаленого серверу.

1.1 Аналіз постачальників хмарної інфраструктури

Основними підрядниками хмарної інфраструктури є:

1.1.1 Microsoft Azure — це набір хмарних сервісів, створених Microsoft, які допомагають привести нові рішення в життя, вирішити сучасні виклики та створити майбутнє. Azure пропонує більше 200 продуктів і хмарних сервісів[1]

Основні функції Azure включають:

— побудова, запуск та управління додатками через кілька хмар, на власних серверах та на периферії, з використанням інструментів та фреймворків на вибір кінцевого користувача;

— безпека з самого початку, підтримується командою експертів та проактивне дотримання нормативних вимог;

— гібридна робота на власних серверах, через кілька хмар та на периферії;

— будування на своїх умовах з використанням всіх мов програмування та фреймворків;

— постійні оновлення та додавання нових функцій, надають можливість швидше виконувати програмні рішення.

Під час аналізу вартості хмарних послуг було визначено, що Microsoft Azure є найдешевшим варіантом (рисунок 1.1).

ТИП ВМ США LINUX	AWS ЗА ГОДИНУ	GOOGLE ЗА ГОДИНУ	AZURE ЗА ГОДИНУ	IBM ЗА ГОДИНУ	AWS ЗА ОЗП	GOOGLE ЗА ОЗП	AZURE ЗА ОЗП	IBM ЗА ОЗП
СТАНДАРТНИЙ 2 VCPU З ЛОКАЛЬНИМ SSD	$0,133	$0,136	$0,100	$0,137	$0,018	$0,018	$0,013	$0,017
СТАНДАРТНИЙ 2 VCPU БЕЗ ЛОКАЛЬНОГО SSD	$0,100	$0,095	$0,100	$0,112	$0,013	$0,013	$0,013	$0,014
БАГАТО ОЗП 2 VCPU З ЛОКАЛЬНИМ SSD	$0,166	$0,159	$0,133	$0,179	$0,011	$0,012	$0,008	$0,011
БАГАТО ОЗП 2 VCPU БЕЗ ЛОКАЛЬНОГО SSD	$0,133	$0,118	$0,133	$0,179	$0,009	$0,009	$0,008	$0,011
ПОТУЖНИЙ ЦП 2 VCPU З ЛОКАЛЬНИМ SSD	$0,105	$0,112	$0,085	$0,075	$0,028	$0,062	$0,021	$0,038
ПОТУЖНИЙ ЦП 2 VCPU БЕЗ ЛОКАЛЬНОГО SSD	$0,085	$0,071	$0,085	$0,075	$0,021	$0,039	$0,021	$0,038

Рисунок 1.1 — Аналіз вартості послуг різних постачальників хмарних сервісів

Azure також пропонує багато можливостей, включаючи програмне забезпечення як сервіс (SaaS), платформу як сервіс (PaaS) та інфраструктуру як сервіс (IaaS)[2]. Microsoft Azure підтримує багато мов програмування, інструментів та фреймворків, включаючи продукти Microsoft та стороннє програмне забезпечення[3].

1.1.2 Amazon Web Services (AWS) — це найбільш всеосяжний та широко використовуваний хмарний сервіс у світі, який пропонує понад 200 повноцінних сервісів з глобальних дата–центрів[4]. Мільйони клієнтів – включаючи найшвидше зростаючі стартапи, найбільші підприємства та провідні державні

установи – використовують AWS для зниження витрат, збільшення гнучкості та прискорення інновацій.[5]

Основними функціями AWS є:

— найбільша функціональність – AWS має значно більше сервісів та функцій у цих сервісах, ніж будь–який інший хмарний провайдер;

— найбезпечніший – AWS створений для того, щоб бути найбільш гнучким та безпечним хмарним обчислювальним середовищем, доступним сьогодні;

— найшвидші інновації – AWS надає можливість використовувати останні технології для експериментування та інновацій;

— найбагатший досвід роботи – AWS має неперевершений досвід, зрілість, надійність, безпеку та продуктивність, який дозволяє розробляти додатки критичної інфраструктури та з високими вимогами стабільності.

AWS також пропонує багато можливостей, включаючи програмне забезпечення як сервіс (SaaS), платформу як сервіс (PaaS) та інфраструктуру як сервіс (IaaS)[6]. Amazon Web Services підтримує багато мов програмування, інструментальних засобів та фреймворків.

Відповідно до ринкового дослідження AWS займає найбільшу частину ринку (рисунок 1.2)

1.1.3 Google Cloud Platform (GCP) — це набір хмарних обчислювальних сервісів, наданих Google, які дозволяють вам зберігати, керувати та аналізувати дані. Він також використовується для розробки, розгортання та масштабування додатків у середовищі Google[7].

Основні функції GCP включають[8]:

— Compute Engine – віртуальні машини, що працюють у дата–центрі Google;

— Cloud Storage – об'єктне сховище, що є безпечним, стійким та масштабованим;

— Cloud SDK – командні інструменти та бібліотеки для Google Cloud;

— Cloud SQL – реляційні бази даних для MySQL, PostgreSQL та SQL Server;

— Google Kubernetes Engine – кероване середовище для запуску контейнеризованих додатків;

— BigQuery – склад даних для бізнес–гнучкості та інформаційних висновків.

GCP пропонує багато можливостей, включаючи програмне забезпечення як сервіс (SaaS), платформу як сервіс (PaaS) та інфраструктуру як сервіс (IaaS). Google Cloud Platform підтримує багато мов програмування, інструментальних засобів та фреймворків[9].

Рисунок 1.2 —Доля ринки різних постачальників хмарних послуг

1.1.4 DigitalOcean був заснований у 2012 році, щоб задовольнити потреби розробників, які шукали прості та доступні хмарні обчислювальні рішення. Їх перший продукт, Droplet, — це проста у використанні віртуальна машина, яку можна запустити всього за кілька хвилин[10].

Основні функції DigitalOcean включають:

— Compute Engine – це віртуальні машини, що працюють у дата–центрі DigitalOcean;

— Cloud Storage – об'єктне сховище, що є безпечним, стійким та масштабованим;

— Cloud SDK – командні інструменти та бібліотеки для DigitalOcean;

— Cloud SQL – реляційні бази даних для MySQL, PostgreSQL та SQL Server;

— Google Kubernetes Engine – кероване середовище для запуску контейнеризованих додатків;

— BigQuery – склад даних для бізнес–гнучкості та інформаційних висновків[11].

Завдяки своїй критичній для місії інфраструктурі та повністю керованим пропозиціям, DigitalOcean допомагає розробникам, стартапам та малому та середньому бізнесу швидко створювати, розгортати та масштабувати додатки для прискорення інновацій та підвищення продуктивності та гнучкості[12].

Відповідно до заявлених критеріїв, Microsoft Azure найбільше відповідає ролі підрядника хмарної інфраструктури, оскільки надає можливість повністю побудувати програмне рішення за допомогою безсерверних систем, використання, яких забезпечує найоптимальніші можливості з точки зору необхідної потужності, а також вартості виконання обчислень

1.2 Аналіз методів конвеєрної збірки

Microsoft Azure надає широкий спектр можливостей для конвеєрної збірки інфраструктури обчислювальних середовищ, основними з яких є:

1.2.1 Azure Pipelines — це сервіс від Microsoft Azure, який дозволяє автоматизувати процеси збірки, тестування та розгортання коду. Він використовує YAML для опису робочих процесів і може бути інтегрований з GitHub, Bitbucket та іншими системами контролю версій[13].

Основними компонентами Azure Pipeline є[14]:

— Pipeline – це один або декілька етапів, які описують процес CI/CD;

— Jobs – визначає роботи, які складають роботу етапу;

— Jobs.deployment – це спеціальний тип роботи. Це набір кроків, які виконуються послідовно проти середовища;

— Jobs.job – робота – це набір кроків, які виконуються агентом або на сервері;

— Parameters – визначає параметри часу виконання, передані в конвеєр;

— Resources – ресурси вказують збудовані, репозиторії, конвеєри та інші ресурси, які використовуються конвеєром;

Azure Pipeline також має потужну систему виразів, яка дозволяє використовувати рядки, булеві значення або числові значення при створенні конвеєра. Вирази можуть бути використані у багатьох місцях, де потрібно вказати значення рядка, булевого або числового значення при створенні конвеєра. Найпоширеніше використання виразів — це умови для визначення того, чи повинна запускатися робота або крок[15].

Основний алгоритм роботи із Azure Pipeline показаний на рисунку 1.3

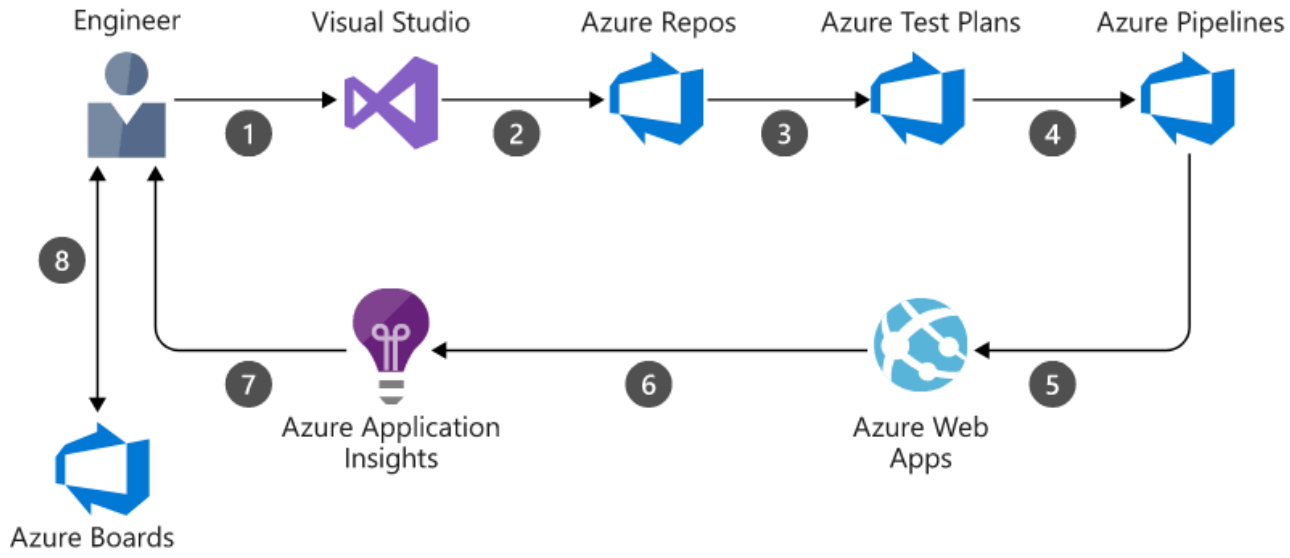


Рисунок 1.3 — Алгоритм роботи із Azure Pipeline.

1.2.2 Azure CLI — це інструмент командного рядка, який дозволяє налаштовувати та керувати ресурсами Azure з багатьох оболонок. Він використовується для створення, розгортання та управління ресурсами Azure[16].

Основні функції Azure CLI включають[17]:

— форматування виводу – Azure CLI пропонує три основні формати виводу: json, table та tsv;

— передача значень іншій команді – якщо значення використовується більше одного разу, його можна призначити змінній;

— використання лапок у параметрах – це дозволяє використовувати спеціальні символи у параметрах;

— використання дефісів у параметрах – це дозволяє використовувати дефіси у назвах параметрів;

— Azure CLI також надає можливість створювати та керувати Azure Function apps за допомогою простих команд. Що дозволяє створювати функціональні додатки, які дозволяють інтеграцію та розгортання[18].

1.2.3 Azure Resource Manager API — це сервіс, який дозволяє створювати, оновлювати та видаляти ресурси Azure. Він використовується для управління ресурсами Azure за допомогою RESTful API[19].

Основні функції Azure Resource Manager API включають:

— створення, оновлення та видалення ресурсів – можливо створювати, оновлювати та видаляти ресурси Azure за допомогою HTTP–запитів до кінцевих точок API;

— управління групами ресурсів – можливо створювати групи ресурсів, які містять пов’язані ресурси, щоб керувати їми як одним цілим;

— управління доступом – можливо управляти доступом до ресурсів за допомогою Azure Role–Based Access Control (RBAC);

— управління шаблонами – можливо створювати шаблони для автоматизації розгортання[20].

Azure Resource Manager API також надає можливість виконувати операції з ресурсами в межах певного простору імен, такого як Microsoft.Compute для віртуальних машин. Кожен простір імен має свої власні операції, які можна виконувати з ресурсами[21].

Серед розглянутих методів, Azure Pipelines є найбільш оптимальним, оскільки дозволяє описати весь процес конвеєрної збірки мовою YAML, що в свою чергу дає можливість працювати із декількома динамічними обчислювальними середовищами та зберігати програмний код конвеєрної збірки з використанням системи контролю версій

1.3 Аналіз методів побудови динамічних обчислювальних середовищ

Для побудови динамічного обчислювального середовища найчастіше використовується підхід «Інфраструктура як код». Це спосіб постачання та керування обчислювальними та мережевими ресурсами методом їх опису у вигляді програмного коду, на відміну від налаштовування необхідного обладнання власноруч чи з допомогою інтерактивних інструментів[22].

Цей підхід дозволяє автоматизувати та стандартизувати процеси розгортання та управління інфраструктурою, що сприяє швидкості, ефективності та надійності. IT інфраструктура, керована таким чином, охоплює як фізичні сервери, так і віртуальні машини, а також пов'язані з ними ресурси.[23]

У підході використовуються як виконувані скрипти, так і декларативні визначення, шаблони, які можуть перебувати в системі контролю версій. Термін найчастіше використовується для позначення декларативного опису IT інфраструктури[24].

Azure підтримує наступні IAC моделі:

1.3.1 Terraform — це інструмент для автоматизації інфраструктури, який дозволяє безпечно та передбачувано створювати та керувати інфраструктурою в будь–якій хмарі[25].

Основні функції Terraform:

— інфраструктура як код — надає можливість використовувати інфраструктуру як код для автоматизації створення інфраструктури, включаючи сервери, бази даних, політики брандмауера та майже будь–який інший ресурс;

— багатохмарне забезпечення — надає можливість розгортати безсерверні функції з AWS Lambda, керувати ресурсами Microsoft Azure Active Directory, забезпечувати балансування навантаження в Google Cloud та багато іншого;

— керування Kubernetes — надає можливість створювати та керувати кластерами Kubernetes на AWS, Microsoft Azure або Google Cloud та взаємодіяти з персональним кластером за допомогою провайдера Kubernetes Terraform;

— керування мережевою інфраструктурою — надає можливість автоматизувати ключові мережеві завдання, такі як оновлення цільових груп балансувальника навантаження або застосування політик брандмауера;

— керування віртуальними образами — надає можливість створювати та керувати віртуальними образами за допомогою Terraform та Packer;

— інтеграція з існуючими робочими процесами — надає можливість автоматизувати розгортання інфраструктури через існуючі робочі процеси CI/CD;

— застосування політики як код — дозволяє застосовувати політики, перш ніж користувач створить інфраструктуру за допомогою політики Sentinel як код;

— введення секретів у Terraform — дозволяє використовувати HashiCorp Vault для автоматизації використання динамічно генерованих секретних даних та облікових даних у конфігураціях Terraform [26].

Terraform працює за принципом “Write, Plan, Apply”. Це дозволяє визначити ресурси, Terraform створює план виконання, описуючи інфраструктуру, яку вона створить, оновить або знищить на основі наявної інфраструктури та конфігурації. При затвердженні Terraform виконує запропоновані операції у правильному порядку, дотримуючись будь–яких залежностей ресурсу [27].

1.3.2 Azure Resource Manager шаблони (ARM шаблони) — це файли JavaScript Object Notation (JSON), які визначають інфраструктуру та конфігурацію для проекту[28]. Ось деякі з основних функцій ARM шаблонів:

— інфраструктура як код — ARM шаблони дозволяють створювати та розгортати всю інфраструктуру Azure в декларативному стилі;

— повторюваність — надає можливість розгортати інфраструктуру протягом всього життєвого циклу розробки та гарантувати, що ресурси розгортаються у послідовний спосіб;

— бізнес–процеси — користувачам не потрібно турбуватися про складність послідовних операцій;

— модульні файли — надає можливість розбити шаблон на менші повторно використовувані компоненти та поєднати їх під час розгортання;

— створення будь–якого ресурсу Azure — можливо використовувати нові служби та функції Azure в користувацьких шаблонах відразу після їх випуску;

— масштабованість — можливо додавати скрипти PowerShell або Bash до користувацьких шаблонів за допомогою скриптів розгортання;

— тестування — надає можливість тестувати користувацькі шаблони за допомогою набору інструментів ARM Template Toolkit (arm–ttk), щоб переконатися, що шаблон відповідає рекомендованим критеріям;

— перегляд змін — можливо переглядати зміни перед розгортанням шаблону за допомогою операцій моделювання[29].

ARM шаблони працюють за принципом “Написати, Планувати, Застосувати”. Користувач визначає ресурси, ARM створює план виконання, описуючи інфраструктуру, яку вона створить, оновить або знищить на основі наявної інфраструктури та конфігурації. При затвердженні ARM виконує запропоновані операції у правильному порядку, дотримуючись будь–яких залежностей ресурсу[30].

1.3.3 Bicep — це доменно–специфічна мова (DSL), яка використовує декларативний синтаксис для розгортання ресурсів Azure [31].

Основними функціями Bicep є [32]:

— підтримка всіх типів ресурсів та версій API — Bicep негайно підтримує всі версії попереднього перегляду та GA для служб Azure;

— простий синтаксис — порівняно з еквівалентним шаблоном JSON, файли Bicep більш стислі та легші для читання;

— досвід авторства — використовуючи розширення Bicep для VS Code для створення ваших файлів Bicep;

— повторювані результати — надає можливість повторно розгортати інфраструктуру протягом життєвого циклу розробки і гарантує впевненість, що ресурси розгортаються у послідовний спосіб.

Bicep працює за принципом “Написати, Планувати, Застосувати”. Для цього необхідно визначити ресурси, Bicep створює план виконання, описуючи

інфраструктуру, яку вона створить, оновить або знищить на основі наявної інфраструктури та конфігурації[33].

Відповідно до аналізу є два основних компоненти, які можуть бути використані в якості IAC фреймворку — Bicep та Terraform. На рисунку 1.4 зображена порівняльна таблиця основних характеристик даних бібліотек.

Рисунок 1.4 — Порівняння характеристик Bicep та Terraform

Оскільки Bicep працює на основі ARM шаблонів і використовує всі їхні переваги, додаючи власні, Bicep буде використовуватись у даній роботі. Також на відміну від Terraform, Bicep має тісну інтеграцію із Azure Cloud, що дозволяє легше відслідковувати хід процесу розгортання і знаходити помилки, які можуть виникати під час даного процесу. Також у Bicep відсутній файл стану, що у свою чергу виключає необхідність виділення ресурсів для його збереження, а також можливість його пошкодження

1.4 Аналіз моделей розміщення коду та вибір компонентів Azure для динамічного обчислювального середовища

Microsoft Azure надає можливість працювати за двома моделями, які безпосередньо впливають на ціну і масштабованість ресурсів.

Серверний сервіс план дозволяє мати більший контроль над ресурсами, а також надає можливість налаштувати та керувати серверами, включаючи обслуговування та оновлення. Це може бути корисним для додатків, які потребують особливих конфігурацій або високого рівня безпеки.

Серверний сервіс план Azure, такий як Azure App Service, дозволяє масштабувати створені додатки, використовуючи різні опції масштабування. Надається можливість вибрати масштабування вгору, щоб збільшити потужність сервера, або масштабування вшир, щоб збільшити кількість серверів.

Масштабування вгору дозволяє збільшити потужність сервера, додаючи більше ЦПУ, пам'яті або дискового простору. Це може бути корисним для додатків, яким потрібно багато ресурсів для обробки великої кількості даних або для виконання складних обчислень.

Масштабування вшир дозволяє збільшити кількість серверів, на яких працює додаток. Це може бути корисним для додатків, яким потрібно обслуговувати велику кількість одночасних користувачів.

Важливо зазначити, що масштабування може мати вплив на вартість сервіс плану Azure. Необхідно розглянути потреби та бюджет проекту перед тим, як змінювати параметри масштабування.

З іншого боку, без серверний сервіс план забезпечує автоматичне масштабування та управління, що дозволяє зосередитися на написанні коду, а не на управлінні інфраструктурою. Необхідно платити лише за те, що використовується, що може бути більш економічно ефективним для додатків з непередбачуваним трафіком

Безсерверний сервіс план Azure, з використанням таких компонентів як Azure Functions, дозволяє масштабувати додатки динамічно, в залежності від потреб. Користувач не повинен турбуватися про налаштування серверів для запуску коду: усі серверні налаштування, планування обчислювальних ресурсів цілком приховані від користувачів і керуються платформою

Це означає, що Azure автоматично забезпечує масштабування ресурсів для додатків на основі потреб. Наприклад, якщо додаток отримує велику кількість трафіку, Azure автоматично збільшить кількість ресурсів, щоб впоратися з цим навантаженням. Коли навантаження знижується, Azure зменшить кількість ресурсів, щоб оптимізувати використання та вартість.

Це робить безсерверну архітектуру ідеальною для додатків з непередбачуваним або змінним навантаженням, оскільки вона може швидко масштабуватися вгору або вниз для задоволення потреб.

На рисунку 1.5 зображено основні порівняльні характеристики серверної та безсерверної архітектури

Оскільки, необхідно забезпечити динамічне масштабування та найбільш економічний варіант, безсерверний сервіс план буде використаний у якості основного.

Базуючись на сервісному плані, можна вибрати решту основних компонентів, для динамічного обчислювального середовища:

— Azure Durable Functions;

— Azure Cosmos DB.

Azure Functions дозволяє запускати користувацький код в середовищі без сервера без необхідності спочатку створювати віртуальну машину (VM) або публікувати веб–додаток. Це дозволяє написати код функції на одній із багатьох мов програмування, використовуючи велику кількість інструментів розробки, а потім розгорнути код в хмару Azure.

Azure Functions надає нативну підтримку для розробки на C#, Java, JavaScript, PowerShell, Python, а також можливість використовувати більше мов, таких як Rust та Go.

Functions інтегрується безпосередньо з Visual Studio, Visual Studio Code, Maven та іншими популярними інструментами розробки для забезпечення безшовного налагодження та розгортань.

Functions також інтегрується з Azure Monitor та Azure Application Insights для надання повного обсягу телеметрії виконання та аналізу функцій у хмарі.

Усі перелічені фактори дозволяють створити свою першу функцію, завершивши одну з наших статей швидкого початку, щоб створити та розгорнути свої перші функції менше ніж за п'ять хвилин.

Azure Durable Functions — це розширення Azure Functions, яке дозволяє писати станові функції в середовищі обчислень без сервера. Воно дозволяє

визначати станові робочі процеси, пишучи функції оркестратора, та станові сутності, пишучи функції сутностей, використовуючи модель програмування Azure Functions.

	ВИДІЛЕНІ ПОТУЖНОСТІ	БЕЗСЕРВЕРНІ ОБЧИСЛЕННЯ
СЦЕНАРІЙ ВИКОРИСТАННЯ	ДЛЯ РЕСУРСІВ ІЗ ПОСТІЙНИМ НАВАНТАЖЕННЯМ	ДЛЯ РЕСУРСІВ ІЗ ЗМІННИМ АБО НЕВІДОМИМ НАВАНТАЖЕННЯМ
ЛЕГКІСТЬ УПРАВЛІННЯ ПРОДУКТИВНІСТЮ	ВИЩА	НИЖЧА
МАСШАТБОВАНІСТЬ	РУЧНА	АВТОМАТИЧНА
ШВИДКОДІЯ	ВИСОКА	НИЖЧА ПІСЛЯ ПЕРІОДІВ ПРОСТОЮ
ОПЛАТА	ЗА ГОДИНУ	ЗА СЕКУНДУ

Рисунок 1.5 — Порівняння серверної та безсерверної архітектури

Ось декілька основних характеристик Azure Durable Functions:

— станові функції — Azure Durable Functions дозволяють писати функції, які можуть зберігати стан між викликами, що дозволяє розробляти більш складні робочі процеси;

— оркестрація — можливо визначати робочі процеси, пишучи функції оркестратора. Оркестратори можуть викликати інші функції та координувати їх виконання;

— сутності — можливо визначати станові сутності, пишучи функції сутностей. Сутності можуть зберігати стан та обробляти команди;

— масштабування — Azure Durable Functions автоматично масштабуються для обробки навантаження;

— мова програмування — Azure Durable Functions працюють з усіма мовами програмування Azure Functions, але можуть мати різні мінімальні вимоги до кожної мови.

Azure Durable Functions — це потужний інструмент для розробки складних, станових робочих процесів у середовищі обчислень без сервера.

Azure Cosmos DB — це міжнародно розподілена служба бази даних з багатьма моделями. Вона дозволяє масштабувати пропускну здатність та зберігання еластично та незалежно через будь–яку кількість регіонів Azure у всьому світі. Використовуючи будь-який з кількох загальних API, можливо скористатися швидким доступом до даних за кілька мілісекунд.

Azure Cosmos DB пропонує одноцифровий час відгуку в мілісекундах, автоматичне та миттєве масштабування, а також гарантовану швидкість при будь–якому масштабуванні. Бізнес–континуїтет гарантується за допомогою доступності, підтриманої SLA, та безпеки рівня підприємства.

Розробка додатків швидше та продуктивніше завдяки:

— готовому розподілу даних у багатьох регіонах у всьому світі;

— використанню вихідних кодів API;

— SDK для популярних мов;

— використанню Retrieval Augmented Generation (RAG) для надання найбільш семантично релевантних даних для збагачення ваших додатків на основі AI, побудованих з моделей Azure OpenAI, таких як GPT–3.5 та GPT–4.

Як повністю керована служба, Azure Cosmos DB відповідає за адміністрування бази даних з автоматичним управлінням, оновленнями та латками. Вона також обробляє управління потужністю за допомогою економічно ефективних безсерверних та автоматичних опцій масштабування, які реагують на потреби додатка, щоб збалансувати потужність із попитом.

Azure Cosmos DB serverless — це новий тип облікового запису в Azure Cosmos DB. Коли ви створюєте обліковий запис Azure Cosmos DB, ви можете вибрати між опціями забезпеченого пропуску та безсерверними.

Ось декілька основних характеристик Azure Cosmos DB:

— використання за споживанням — з безсерверною опцією, плату буде нараховано лише за одиниці запиту (RUs), які споживають операції з базами даних, та за зберігання, яке споживають користувацькі дані;

— масштабування — безсерверні контейнери можуть обслуговувати тисячі запитів на секунду без мінімальної плати та без необхідності планування потужностей;

— вартість — необхідно заплатити лише за кількість RUs, які споживають операції з базами даних;

— сценарії використання — Azure Cosmos DB serverless найкраще підходить для сценаріїв, в яких очікується проміжний та непередбачуваний трафік та тривалі перерви з малою або з відсутньою активністю.

Щоб почати використовувати модель без сервера, потрібно створити новий безсерверний обліковий запис. Перехід існуючого облікового запису до або з безсерверної моделі наразі не підтримується.

2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДІВ МАСШТАБУВАННЯ РЕСУРСІВ В ДИНАМІЧНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩАХ

Економічна ефективність є однією з причин популярності хмарних сервісів. За рахунок ефективного використання ресурсів витрати можна ще більше зменшити, а втрати ресурсів можна мінімізувати. Вимоги до програмного забезпечення можуть відрізнятися в залежності від багатьох факторів (наприклад, навантаження на програму); користувач може запускати різні типи програм (від простого текстового редактора до складної програмної системи) у віртуальній машині. У таких випадках, якщо характеристики екземпляра віртуальної машини є статичними існує висока ймовірність невідповідності між даними характеристиками та вимогами програми до ресурсів. Якщо характеристики віртуальної машини більше, ніж вимоги до ресурсів програми, тоді ресурси будуть витрачені даремно; якщо параметри віртуальної машини менші за вимоги до ресурсів програми – продуктивність програми знизиться. Для вирішення цих проблем запропоноване автоматичне масштабування віртуальних машин на основі порогових значень, у яких віртуальні машини будуть динамічно масштабуватися на основі використання програмних ресурсів (ЦП і пам'ять).

2.1 Проектування системи масштабування пам’яті за пороговими значеннями

У хмарній парадигмі програмного забезпечення, інфраструктура та платформа надаються як послуги. Дана робота, розглядає інфраструктуру (віртуальні машини). Сервісні компанії надають віртуальні машини (VM) для кінцевого користувача. Користувач використовує екземпляр віртуальної машини для розміщення/запуску свого програмного забезпечення, і він заплатить певну суму відповідно до SLA (Угода про рівень обслуговування). Багато організацій переходять до приватної хмари; ефективне використання ресурсів, зниження вартості та легке обслуговування є однією з причин для цього. Співробітники в організаціях отримують екземпляри віртуальних машин. Вони мають увійти в ці

екземпляри, для того щоб використовувати їх. Незалежно від того, комерційна це хмара чи приватна, існує два можливих сценарії:

2.1.1 Користувач розміщує різні програми на віртуальній машині і при цьому може використовувати віртуальну машину для розміщення різних програм від простого текстового редактора до складних бухгалтерських програм. Якщо екземпляр віртуальної машини є статичним (зазвичай це так), користувач має вибрати екземпляр віртуальної машини таким чином, щоб відповідати максимальним потребам програми у ресурсах. У цьому випадку, якщо користувач використовує віртуальну машину для запуску програми, яка має максимальні потреби у ресурсах лише протягом 2 годин на день то протягом решти 22 годин ресурс буде витрачено даремно. Якщо вимога до ресурсів програми більша ніж кількість ресурсів віртуальної машини, то це призводить до деградації продуктивності програми.

2.1.2 Вимоги до програми змінюються з часом, при цьому можна розглянути програму бази даних, яка потребує більше ресурсів, коли транзакції відбуваються. Якщо транзакцій немає, це не потребує великих ресурсів. В випадку статичного екземпляру віртуальної машини, це призведе до втрати ресурсів. Для вирішення даної проблеми можливе перенесення програми з однієї віртуальної машини на іншу, але воно має багато недоліків. Це забирає багато часу, утомливо, економічно не ефективно і підвищує імовірність виникнення помилок. Якщо VM динамічно масштабується відповідно до вимог програми втрата ресурсів може бути мінімізована.

Для вирішення вищезазначених проблем було розроблено та перевірено поріг на основі механізму автоматичного масштабування віртуальної машини, у якому віртуальна машина автоматично налаштовується відповідно до вимог програми. Під час автоматичного масштабування використання ресурсу на основі порогового значення віртуальної машини відстежується. Якщо значення перевищують попередньо визначені порогові значення, то характеристики віртуальної машини будуть збільшуватися або зменшуватися динамічно без її вимкнення відповідно до потреб, що мінімізує втрату ресурсів.

У даній роботі розглядається використання оперативної пам’яті та процесора віртуальної машини. Коли збільшується потреба програми в ресурсах, завантаження оперативної пам’яті та ЦП VM збільшується. У якийсь момент потреба в ресурсах програми стане більшою в порівнянні з потужністю віртуальної машини в результаті продуктивність програми деградує і зрештою вона перестане працювати. Щоб уникнути цієї проблеми, коли використання процесора і пам’яті віртуальної машини перевищує попередньо визначене максимальне порогове значення автоматично збільшується ємність оперативної пам’яті та процесора віртуальної машини.

Коли потреба програми в ресурсах зменшиться, відповідно зменшиться потреба в оперативній пам’яті і процесорі. Це призведе до втрати ресурсів віртуальної машини коли потужність використовується не повністю. Щоб уникнути втрати ресурсів, коли використання ЦП і пам'яті віртуальної машини перевищує заздалегідь визначене мінімальне порогове значення, необхідно обсяг оперативної пам’яті та процесора віртуальної машини. Моніторинг і масштабування оперативної пам'яті і ЦП віртуальної машини є двома незалежними завданнями.

Вимоги до програми можуть змінюватися з часом, а також користувач може запускати різні програми (які мають інші вимоги до ресурсів) на віртуальній машині. У цих випадках фіксована ємність віртуальної машини може призвести до втрати ресурсів або деградації продуктивності програми. Це можна вирішити шляхом динамічного масштабування віртуальної машини відповідно до вимог до розміщеної програми. Під час автоматичного масштабування ресурсу на основі порогового значення відстежується використання віртуальної машини. Якщо показники перевищують попередньо встановлені порогові значення, тоді ємність VM буде динамічно збільшуватися або зменшуватися відповідно до потреб без вимикання віртуальних машин, що мінімізує втрату ресурсів. Загальний вигляд системи зображений на рисунку 2.1

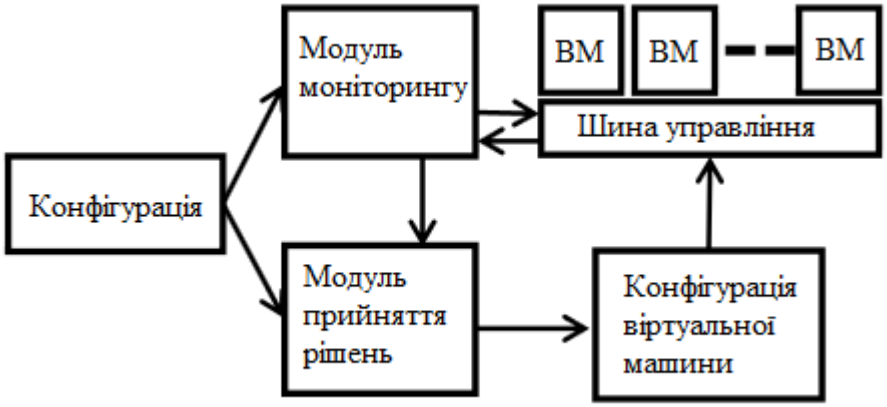


Рисунок 2.1 — Загальний вигляд системи масштабування

Система масштабування складається із наступних компонентів:

2.2.1 Модуль моніторингу — відстежує віртуальні машини; він зчитує використання ЦП і пам'яті і передає ці дані компоненту прийняття рішень. Він використовує шину управління для отримання значень ЦП і використання пам'яті віртуальних машин. За замовчуванням він відстежує всі активні віртуальні машини або є можливість контролю лише певних віртуальних машин, налаштувавши відповідні значення в конфігурації. Інтервал часу для надсилання запиту шині даних для отримання статистики віртуальних машин можна налаштувати в конфігурації.

Коли модуль моніторингу запускається, він зчитує всі значення конфігурації з конфігураційного файлу і відстежує віртуальні машини відповідно до даних значень.

2.2.2 Модуль прийняття рішень — отримує статистику віртуальної машини з модуля моніторингу, а також читає порогові значення з файлу конфігурації, порівнює їх із статистикою віртуальної машини та вирішує, чи потрібно масштабувати віртуальну машину вгору/вниз, і передає це рішення модуль конфігурації віртуальної машини. Інформація, що передається до модуля конфігурації віртуальної машини містить ідентифікатор віртуальної машини, яку необхідно масштабувати, тип масштабування: оперативна пам’ять ЦП і кількість необхідних ресурсів. Існує ймовірність того, що використання процесора та оперативної пам’яті віртуальної машини може перевищувати порогове значення

протягом кількох секунд і знову повернутися до нормальних значень. Якщо модуль монітора отримує ці значення, він ініціює збільшення/зменшення масштабу оперативної пам’яті/ЦП. У наступній ітерації модуль монітора знову отримує нормальні значення та ініціює зменшення/збільшення розміру оперативної пам’яті/ЦП віртуальної машини, що призводить до непотрібних операцій масштабування віртуальних машин. Для уникнення цієї проблеми, запроваджено конфігураційні значення, які називаються cpuiteration (min і max) і memoryiteration (min і max). Будь–яке додатне ціле число від 0 до n може бути встановлено для обчислення та збереження. Модуль прийняття рішень ініціює збільшення/зменшення ресурсів, лише якщо використання оперативної пам’яті та ЦП віртуальної машини перевищує порогові значення в послідовній кількості ітерацій, вказаних у конфігурації.

2.2 Алгоритми масштабування пам’яті та ресурсів процесора.

Було розроблено два окремих алгоритми для масштабування пам’яті та ЦП. Принцип роботи обох алгоритмів однаковий. У разі використання алгоритму масштабування пам’яті кожна віртуальна машина використовує пам’ять (Mx). Значення кількості зчитувань із пам’яті порівнюється з максимальним пороговим значенням пам’яті (Mmx). Якщо Mx більше або дорівнює Mmx, тоді лічильник максимального використання пам’яті (MTx) збільшується, якщо менше лічильник використання пам'яті (MTm) скидається.

Роботу алгоритму масштабування пам’яті показано на рисунку 2.2. Параметри алгоритму мають наступні значення:

— Mmx — максимальне значення пам’яті;

— Mmn — мінімальне значення пам’яті;

— T — інтервал для зчитування значення пам’яті;

— TTMt — порогове значення лічильника пам’яті;

— MTx — максимальна кількість зчитувань значення пам’яті;

— MTm — мінімальна кількість зчитувань значення пам’яті;

— Mx — використання пам’яті віртуальною машиною.

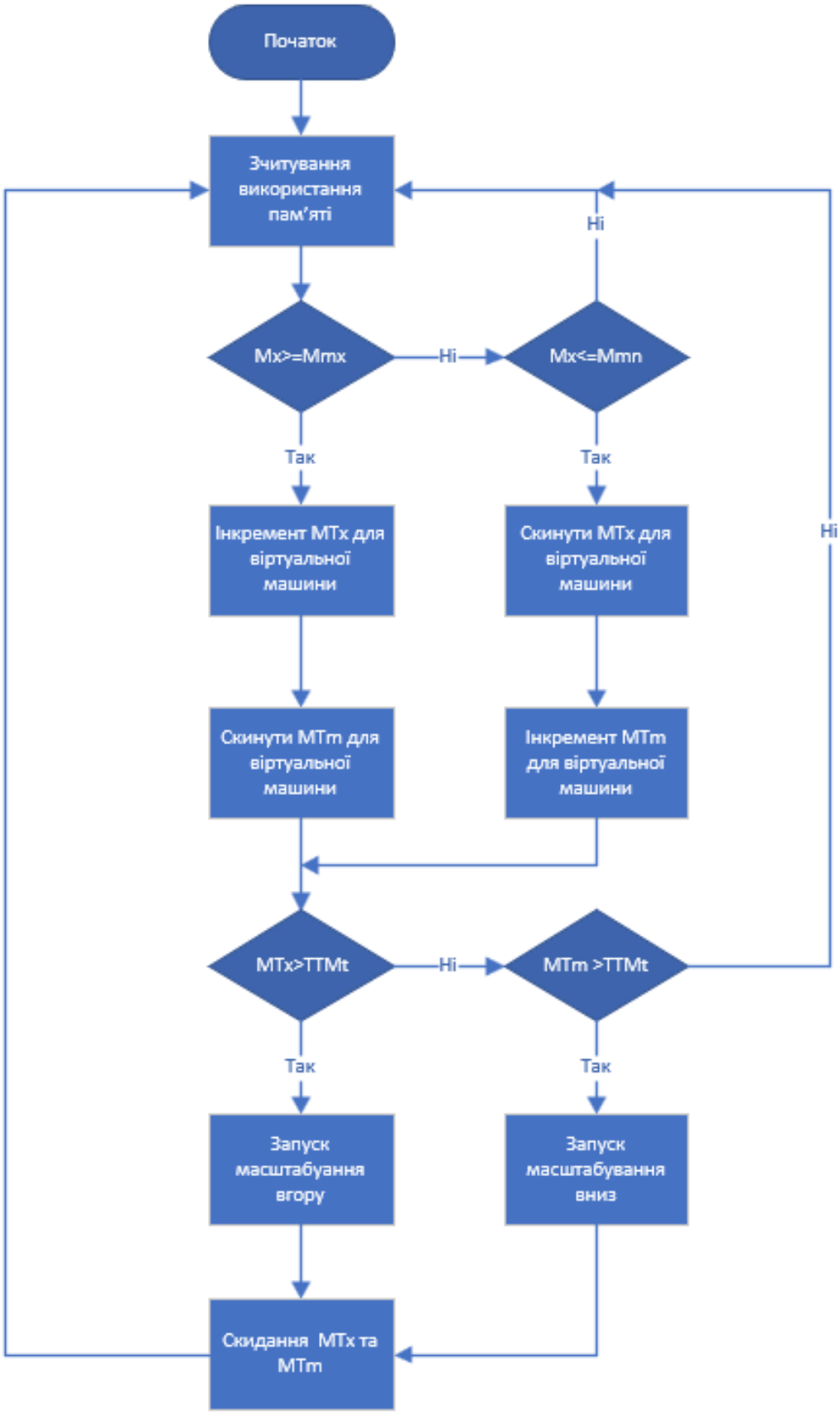


Рисунок 2.2 — Алгоритм масштабування обсягу пам’яті

Робота алгоритму масштабування ресурсів центрального процесору показана на рисунку 2.3

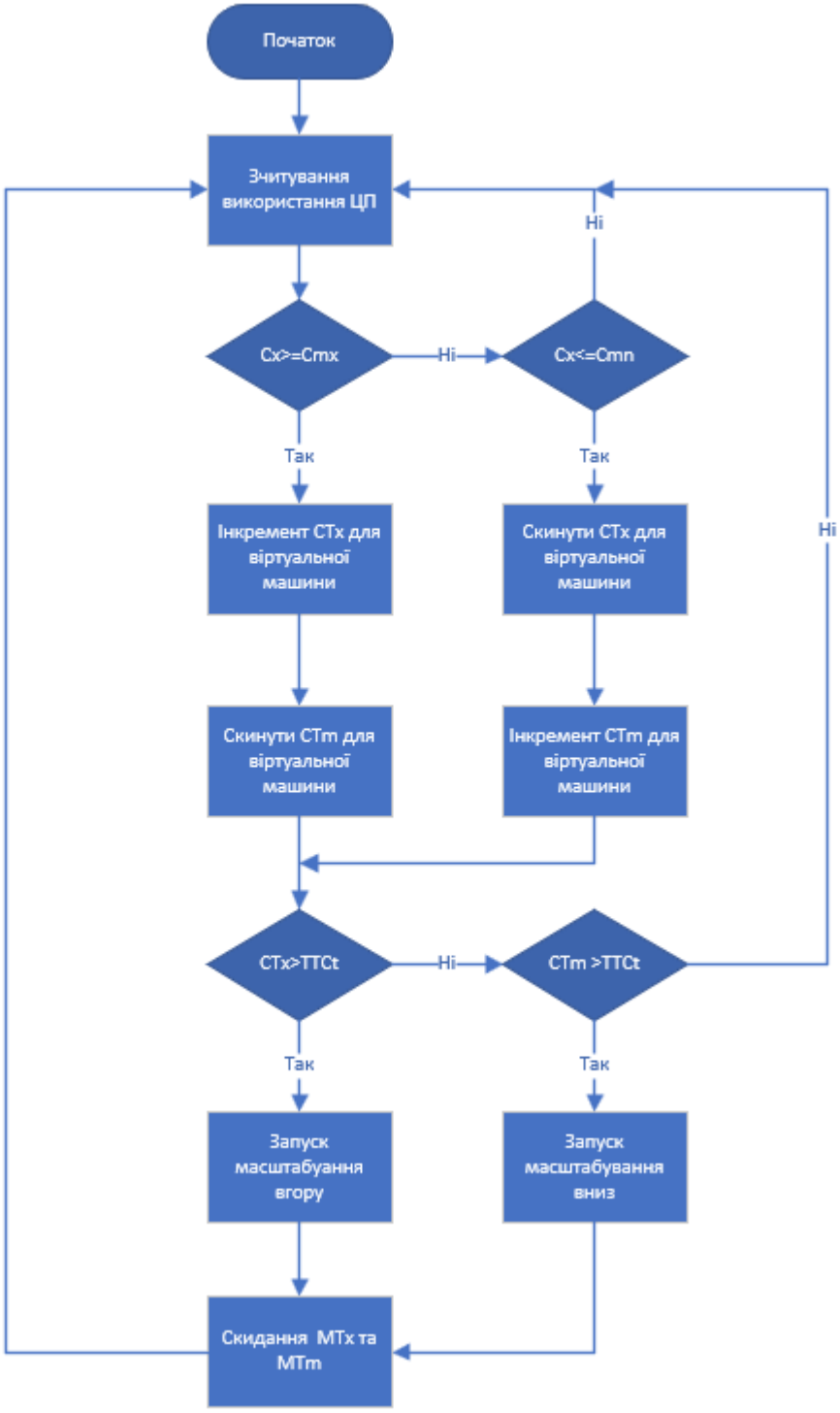


Рисунок 2.3 — Алгоритм масштабування ресурсів центрального процесора

Параметри алгоритму мають наступні значення:

— Cmx — максимальне значення ЦП;

— Cmn — мінімальне значення ЦП;

— T — інтервал для зчитування значення ЦП;

— TTCt — порогове значення лічильника ЦП;

— CTx — максимальна кількість зчитувань значення ЦП;

— CTm — мінімальна кількість зчитувань значення ЦП;

— Cx — використання ЦП віртуальною машиною.

2.3 Математична модель алгоритму масштабування за пороговими значеннями

Математична модель для горизонтального масштабування на основі декількох показників може бути побудована на основі правил прийняття рішень

Правило прийняття рішень — це відображення деякого набору інформації в певну дію, відповідно послідовність правил прийняття рішень $\eta = (q_0, q_1, q_2, ..)$ називається політикою. Найпоширенішою політикою є рандомізована політика, що залежить від історії. Така політика характеризуються єдиним детермінованим правилом прийняття рішень, яке відображає поточний стан у прийнятому рішенні. Таким чином, ми розглянемо відображення q з S до A таке, що виконується $q(x) = a$.

Відповідно до теми роботи математична модель будується на основі спеціальної форми політики: політики гістерезису. Оскільки розроблений алгоритм масштабування використовує два показники для моніторингу буде використано політику із подвійними пороговими значеннями. У такому випадку правило прийняття рішення визначається на основі показнику ресурсу до відповідних порогових значень $q(m, k) = k_1$ при $k_1 \in [1, \dots, K]$. Декілька типів ресурсів можуть досягати порогових значень зі стану k до k_1.

Політика із подвійними пороговими значеннями — це стаціонарна політика така як і правило прийняття рішень q(m, k), яка полягає у збільшені чи зменшені своїх аргументів в залежності від послідовності порогових значень для будь–яких k і k_1.у $[1, K]$, відповідно:

$$q(m, k) = k_1 \text{коли } l_{k_1}(k) \leq m < l_{k_1+1}(k) \qquad (2.1)$$

де $l_{k_1}(k) = \min\{m : q(m,k) \geq k_1\}$

Цей мінімум дорівнює ∞, якщо набір порожній. Для всіх k, $l_{K+1}(k) = \infty$ і $l_1(k) = 0$ (оскільки хоча б одне порогове значення повинне моніторитись).

Для декількох показників доцільно використовувати монотонну політику гістерезісу, яка являє собою підвид політики із декількома граничними значеннями

Монотонна політика гістерезісу — це політика, яка має декілька граничних значень існує декілька наборів цілих чисел l_k і L_k, для яких

$$l_k = l_k(K) = l_k(K-1) = \cdots = l_k(k) \qquad (2.2)$$
$$L_k = l_k(1) = l_k(2) = \cdots = l_k(k-1)$$

При $l_k \leq L_k$ для $k = 1, \ldots, K+1$; $l_k \leq L_{k+1}$, для $k = 1, \ldots, K$; $l_1 = L_1 = 0$ $l_{K+1} = L_{K+1} = \infty$.

Для усіх $(m,k) \in S$:

$$q(m,k) = \begin{cases} q(m,k-1), \text{ якщо } m < l_k \text{ і } k > 0 \\ k, \text{ якщо } l_k \leq m < L_{k+1} \\ q(m,k+1), \text{ якщо } m \geq L_{k+1} \text{ і } k < m \end{cases} \qquad (2.3)$$

Порогові значення lk вказують на те що деякі сервери повинні бути масштабовані при досягненні Lk+1 аналогових точок активації (рис 2.4).

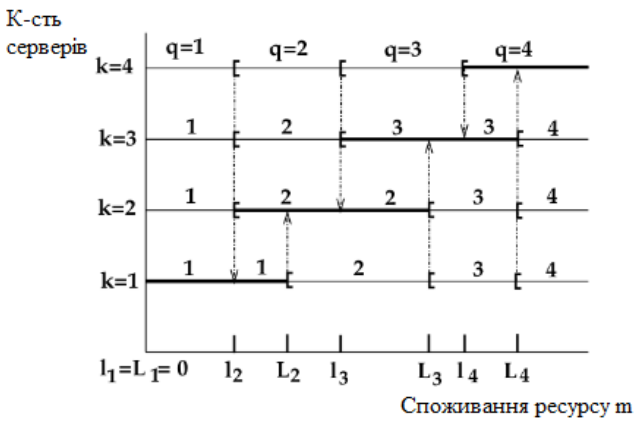


Рисунок 2.4 — Гістерезис алгоритму масштабування

2.4 Результати роботи алгоритму масштабування за пороговими значеннями

Хмарне середовище налаштовується за допомогою Xen Cloud Platform (XCP). XCP включає в себе Xen Hypervisor, стек інструментів Xen API, vSwitch тощо. XCP — корпоративна програма з відкритим кодом, яка являє собою платформу віртуалізації серверів і хмарних обчислень. Багато з існуючих IaaS провайдерів використовують налаштований XEN для створення інфраструктури віртуалізації. XCP надає гіпервізор Xen із підтримкою ряду гостьових операційних систем, включаючи підтримку мереж і сховищ Windows і Linux, інструменти керування в єдиному перевіреному інтсаляційному образі, який також називають пристроєм XCP. Він також підтримує динамічне масштабування віртуальної машини. XCP API використовуються для отримання статистики пам’яті та використання ЦП віртуальних машин, а також для масштабування віртуальних машин

XCP підтримує два типи пам’яті: статичну та динамічну. Кожен тип має мінімальне і максимальне значення. Максимальний обсяг статичної пам’яті визначає максимальний обсяг фізичної пам'яті, до якої гостьова операційна система може звертатися з моменту, коли гостьова система завантажується, до моменту, коли гостьова система вимкнеться. Неможливо змінити обсяг статичної пам'яті, коли віртуальна машина працює. У випадку динамічної пам'яті можливо збільшити/зменшити діапазон під час роботи віртуальної машини. XCP надає функцію під назвою контролер динамічної пам'яті. Використовуючи такі API, надані XCP, з’являється збільшувати/зменшувати динамічну пам’ять у межах допустимого діапазону, коли це необхідно.

Масштабування ЦП можна здійснити шляхом зміни ємності ЦП. Ємність ЦП додатково фіксує максимальну кількість ЦП, яку може споживати домен. Обмеження виражається у відсотках від одного фізичного ЦП: 100 — це 1 фізичний ЦП, 50 – половина ЦП, 400 — це 4 ЦП тощо... Значення за замовчуванням 0 означає відсутність верхнього обмеження . Вага ЦП віртуальної машини визначає, скільки центрального процесора виділено для цієї віртуальної машини. Домен із вагою 512 отримає вдвічі більше ЦП, ніж домен із вагою 256

на виділеному комп’ютері. Допустимі ваги варіюються від 1 до 65535, а за замовчуванням мають значення 256 . В даному випадку для масштабування використовується ємність ЦП.

Хмарний сервер налаштовано на 4–ядерній машині з процесором Intel Xeon W3250 з 2,67 ГГц, 12 ГБ оперативної пам'яті DDR3, 1 ТБ Жорсткий диск з 7200 об/хв. Машина, яка використовується для моніторингу використання пам’яті та ЦП віртуальними машинами має процесор Intel Core2 Duo з тактовою частотою 2,66 ГГц, 1 ГБ оперативної пам’яті, жорсткий диск 500 ГБ, під’єднаний через локальну мережу Ethernet. Використовується операційна система CentOS 5.7.

Для створення навантаження на віртуальні машини використовувались як інтенсивні обчислювальні програми, так і програми, що інтенсивно використовують пам'ять. Після запуску програм використання процесору і пам'яті віртуальних машин збільшилось. Було встановлено наступні коефіцієнти: коефіцієнт масштабування для пам’яті як 1,25, а для ЦП — 2, лічильник використання встановлено на 3 хвилини, тобто якщо ресурс використання перевищує верхнє порогове значення протягом 3 хвилин безперервно, потім відповідним віртуальним машинам буде виділено більше ресурсів, як зазначено в коефіцієнті масштабування (масштабування в гору). Якщо використання ресурсу нижче нижнього порогового значення протягом 3 хвилин безперервно, тоді ресурс буде вилучено з віртуальної машини відповідно до коефіцієнту масштабування (Масштабування вниз).

Були встановлені наступні порогові значення: верхній поріг — 80%, нижній — 25%.

Необхідно враховувати наступні особливості:

— масштабування вниз мінімізує витрату ресурсів;

— від час масштабування в гору необхідно переконатися, що продуктивність програми не погіршується;

— вибір правильних порогових значень важливий для коректної роботи алгоритму.

Нижче порогове значення призводить до підвищення коливання ємності VM, а вище порогове значення робить алгоритм менш чутливими до зміни ресурсу використання ВМ.

На Рисунках 2.5 і 2.6 показано збільшення та зменшення пам’яті відповідно. Рисунки 2.7 і 2.8 показують збільшення та зменшення масштабу ЦП відповідно. Оскільки було встановлено підрахунок використання до 3 хвилин, можна спостерігати масштабування на 4–й хвилині в усіх випадках.

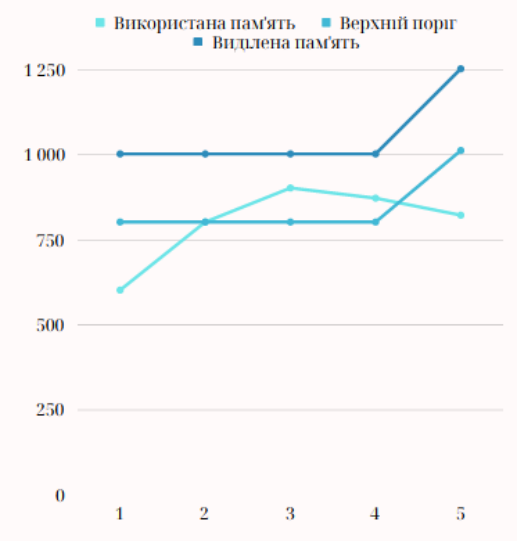


Рисунок 2.5 — Збільшення розміру оперативної пам’яті

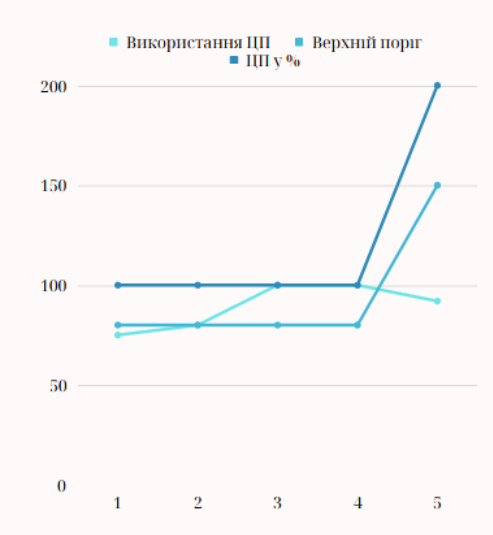


Рисунок 2.7 — Масштабування ЦП вгору

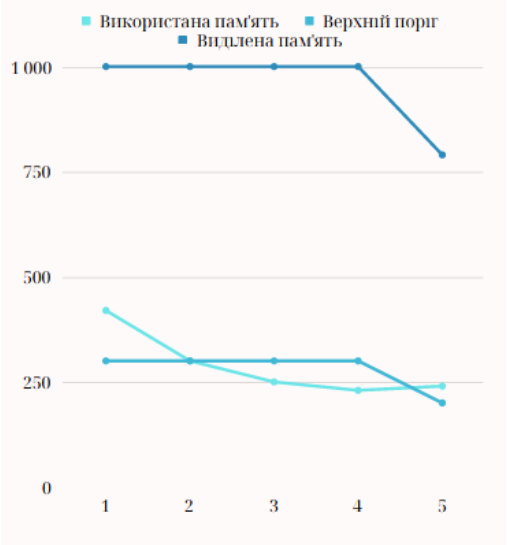


Рисунок 2.6 — Зменшення розміру оперативної пам’яті

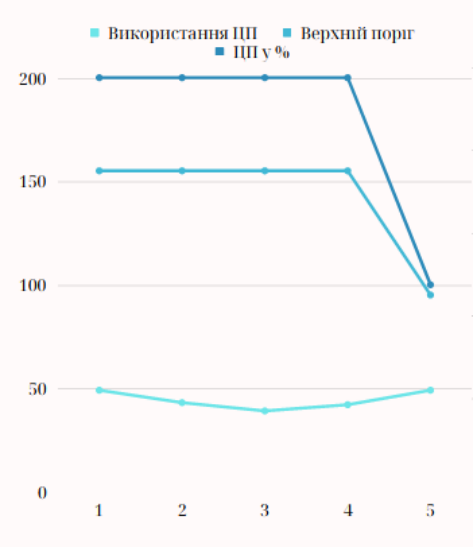


Рисунок 2.8 — Масштабування ЦП донизу

2.5 Розширення моделі масштабування з використанням порогових значень за допомогою горизонтального масштабування

Після застосування алгоритму масштабування за допомогою порогових значень, досі існує імовірність того, що складна програмна система використає всім можливості вертикального масштабування і потрібно буде додавати додаткові екземпляри віртуальних машин і розділяти обчислювальні завдання поміж ними.

Відповідно до цього пропонується розширити алгоритм масштабування за допомогою порогових значень можливістю вертикального масштабування, як це показано на рисунку 2.9. Якщо використання ресурсів віртуальної машини досягають максимального порогового значення протягом інтервалу TMIcx, якщо він більший максимального порогового інтервалу TMIcxm – запускається горизонтальне масштабування і додається новий екземпляр віртуальної машини. Значення лічильнику інтервалу скидається. В іншому разі перевіряється значення мінімального порогового інтервалу TMIcx , якщо воно менше за мінімальний пороговий інтервал TMIcxmn — запускається горизонтальне масштабування і екземпляр віртуальної машини видаляється. Значення лічильнику інтервалу скидається.

Застосування ефективних методів використання ресурсів може звести до мінімуму призвести втрату ресурсів. Автоматичне масштабування на основі порогового значення є одним із методів, при якому віртуальна машин динамічно масштабується відповідно до вимог програми до ресурсів, таким чином мінімізуючи використання ресурсів.

Вибір належних порогових значень є дуже важливим фактором успіху даного підходу. Нижче порогове значення призводить до частої зміни конфігурації віртуальної машини та більш високе значення знижує здатність віртуальної машини адаптуватися до нових вимог до ресурсів. Існує можливість використання декількох методів, щоб знайти оптимальні порогові значення. Наприклад, на основі історії, математичної моделі тощо.

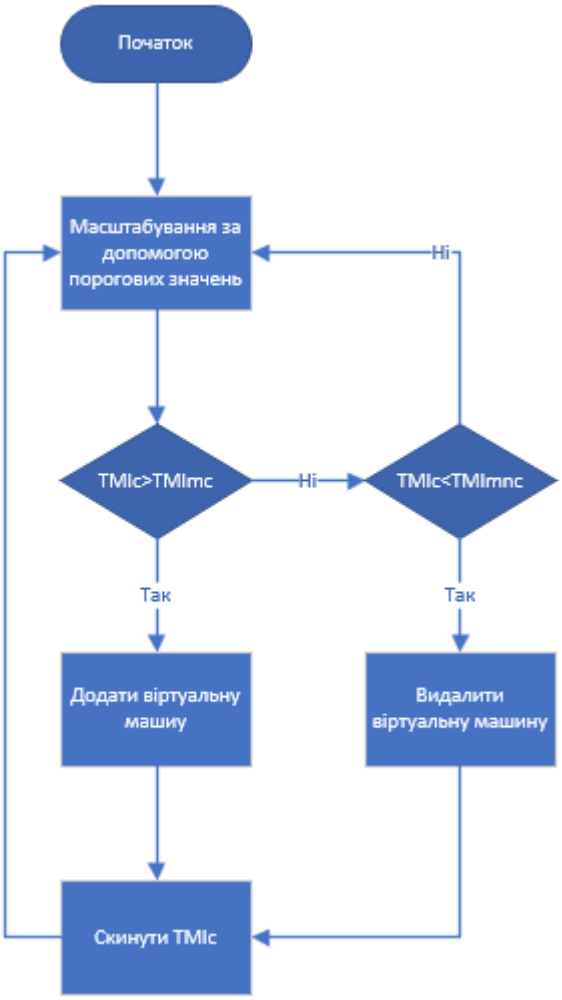


Рисунок 2.9 — Вертикальне масштабування

Наразі система динамічного масштабування базується на пороговому значенні, коли порогові значення є статичними та попередньо визначеними. Якщо протягом певного інтервалу система досягає порогових значення, то запускається горизонтальне масштабування.

3 РОЗРОБКА ТЕХНОЛОГІЇ КОНВЕЄРНОЇ ЗБІРКИ ДИНАМІЧНИХ СЕРЕДОВИЩ В ХМАРНОМУ СЕРЕДОВИЩІ MICROSOFT AZURE

Процес розробки даної технології було розділено на наступні етапи, які в подальшому розглядаються в наступних підрозділах:

1) розробка IAC рішення — Bicep компоненту, який відповідатиме за створення динамічного середовища;

2) реєстрація середовищ на порталі Azure DevOps;

3) розробка конвеєру збірки динамічних середовищ;

4) імплементація каркасного рішення динамічного обчислювального середовища із автоматичним масштабуванням;

5) розробка конвеєру збірки для каркасного рішення;

6) розробка конвеєру завантаження для каркасного рішення.

Черговість даних кроків є важливою, оскільки реалізація кожного кроку залежить від попереднього

3.1 Розробка IAC рішення – Bicep компоненту, який відповідатиме за створення динамічного середовища

Для збереження результатів роботи на порталі Azure DevOps створимо гіт репозиторій, який матиме назву DynamicComputingEnvironment.IaC. Даний репозиторій буде зберігати Bicep код, для створення динамічного обчислювального середовища, а також YAML код, необхідний для створення конвеєрів збірки і розгортання динамічного обчислювального середовища.

Основними компонентами даного модуля є файл main.bicep, який є точкою входу в програму, та набір модулів, які відповідають за створення кожного компоненту. Повний код IAC рішення наведений у Додатку В.

Основними модулями компонентів є:

— модуль сховища даних;

— модуль аналізу діагностичних даних;

— модуль сервіс плану;

— модуль сховища динамічної функції;

— модуль динамічної функції;

— модуль налаштування дозволів;

— модуль сховища конфіденційних даних.

Контейнерна діаграма bicep коду зображена на рисунку 3.1

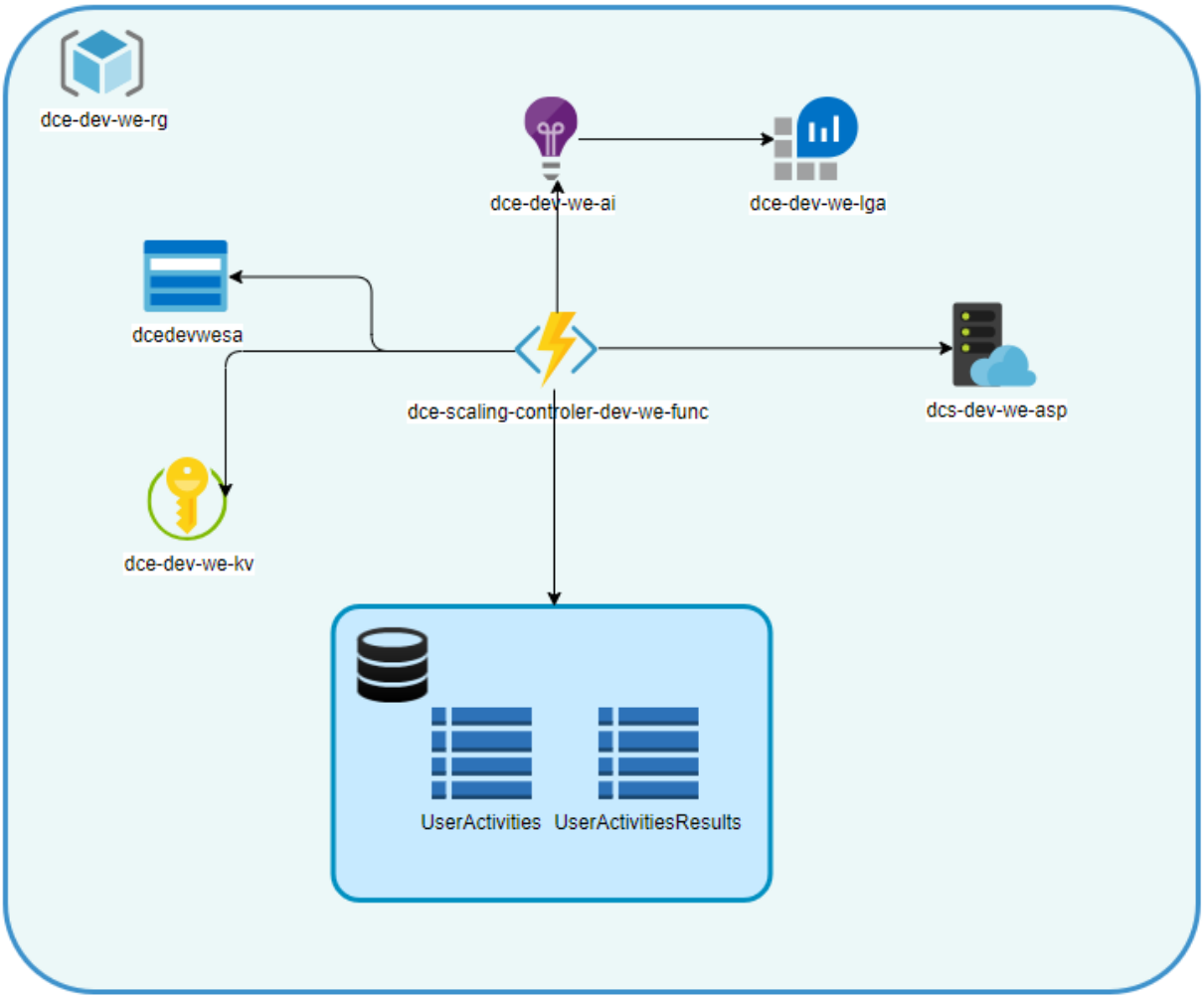


Рисунок 3.1 — Контейнерна діаграма динамічного обчислювального середовища

3.1.1 Модуль сховища даних

Модуль сховища даних відповідає за створення і налаштування бази даних Azure CosmosDb. Основними його функціями є: створення облікового запису CosmosDb, створення бази даних, створення контейнерів і налаштування їх партішин ключів та пошукових індексів, налаштування режиму роботи бази

даних та її пропускної спроможності. Як результат роботи даний модуль повертає рядок з’єднання, який необхідно використовувати для з’єднання із створеною базою даних. Налаштування облікового запису для модуля сховища даних показане у лістингу 3.1

Лістинг 3.1 — Налаштування облікового запису модуля сховища даних

```
param name string
param location string
param tags object
param throughputLimit int = 6000

resource cosmosDbAccount
'Microsoft.DocumentDB/databaseAccounts@2023-04-15' = {
  name: name
  location: location
  kind: 'GlobalDocumentDB'
  tags: tags
  identity: {
    type: 'SystemAssigned'
  }
  properties: {
    consistencyPolicy: {
      defaultConsistencyLevel: 'Session'
    }
    locations: [
      {
        locationName: location
        failoverPriority: 0
      }
    ]
    capacity: {
      totalThroughputLimit: throughputLimit
}
    databaseAccountOfferType: 'Standard'
    enableAutomaticFailover: false
  }
}
var connectionString =
listConnectionStrings(resourceId('Microsoft.DocumentDB/databas
eAccounts', name),
cosmosDbAccount.apiVersion).connectionStrings[0].connectionStr
ing
```

Продовження лістингу 3.1

```
output name string = cosmosDbAccount.name
output connectionString string = connectionString
```

Даний модуль створює обліковий запис у вказаному розташуванні, налаштовує пропускну спроможність. Як вхідні параметри він приймає: ім’я облікового запису, назву локації, список тегів та пропускну спроможність. Як результат роботи повертається ім’я створеного облікового запису, а також рядок з’єднання.

Модуль налаштування бази даних наведений у лістингу 3.2. Його основною задачею є створення бази даних та налаштування її пропускної спроможності

Лістинг 3.2 — Створення бази даних

```
param databaseName string
param accountName string
param location string
param tags object
param maxThroughput int = 5000

resource cosmosDatabase
'Microsoft.DocumentDB/databaseAccounts/sqlDatabases@2023-04-
15' = {
name: '${accountName}/${databaseName}'
  location: location
  tags: tags
  properties: {
    options: {
      autoscaleSettings: {
        maxThroughput: maxThroughput
      }
    }
    resource: {
      id: databaseName
  }
  }
}

output name string = databaseName
```

Даний модуль приймає на вхід наступні параметри: ім’я бази даних і облікового запису користувача, локацію список тегів а також пропускну спроможність бази даних. Як результат повертається ім’я створеної бази даних.

Модуль створення контейнерів наведений у лістингу 3.3, він використовується для створення контейнерів у вказаній базі даних

Лістинг 3.3 — Модуль створення контейнерів бази даних

```
param accountName string
param databaseName string
param collectionConfig array

var conflictResolutionPolicy = {
conflictResolutionPath: '/_ts'
  mode: 'LastWriterWins'
}

resource containers
'Microsoft.DocumentDB/databaseAccounts/sqlDatabases/containers
@2023-04-15' = [for config in collectionConfig: {
    name: '${accountName}/${databaseName}/${config.name}'
    properties: {
      resource: {
        id: config.name
        partitionKey: {
          paths: [config.partitionKey]
          kind: 'Hash'
        }


        conflictResolutionPolicy: conflictResolutionPolicy
        indexingPolicy: {
          automatic: true
          indexingMode: 'consistent'
               includedPaths: config.indexIncludedPaths
               excludedPaths: config.indexExcludedPaths
             }
           }
         }
}]
```

Даний модуль налаштовує ключ партиції, індексування контейнеру і спосіб вирішення конфліктів під час паралельних запитів і не повертає жодних значень

3.1.2 Модуль аналізу діагностичних даних

Обов’язком даного модуля є аналіз і відображення всіх логів, зовнішніх залежностей, запитів та помилок роботи програми. Лістинг 3.4 відображає основну конфігурацію модуля діагностичних даних

Лістинг 3.4 — Модуль діагностичних даних

```
param name string
param location string
param tags object
param workspaceName string

resource appInsights 'Microsoft.Insights/components@2020-
02-02' = {
  name: name
  location: location
  kind: 'web'
  properties: {
    Application_Type: 'web'
    publicNetworkAccessForIngestion: 'Enabled'

publicNetworkAccessForQuery: 'Enabled'
    WorkspaceResourceId: logAnalyticsWorkspace.id
  }
  tags: tags
}


resource logAnalyticsWorkspace
'Microsoft.OperationalInsights/workspaces@2022-10-01' = {
  name: workspaceName
  location: location
tags: tags
  properties: {
    retentionInDays: 30
    sku: {
```

Продовження лістингу 3.4

```
  name: 'PerGB2018'
    }
  }
}

output name string = appInsights.name
output instrumentationKey string =
appInsights.properties.InstrumentationKey
output logAnaliticsName string =
logAnalyticsWorkspace.name
```

Модуль приймає на вхід наступні параметри: ім’я ресурсу, назву робочого середовища, локацію та список тегів. Як результат роботи повертається інструментальний ключ, що використовується для зв’язку із іншими компонентами

3.1.3 Модуль сервіс плану

Модуль сервіс плану призначений для виділення ресурсів для виконання програмного коду, таких як: потужність процесора, кількість оперативної пам’яті, розмір жорсткого диску, масштабованість і тд. Модуль приймає на вхід наступні параметри: ім’я ресурсу, локація, список тегів, та параметри відповідальні за ресурси виконання. Результатом виконання даного модуля буде отримання ідентифікатора створеного ресурсу. Основний код даного модуля відображено у лістингу 3.5

Лістинг 3.5 — Модуль сервісного плану

```
param name string
param location string
param tags object
param skuName string
param tier string

resource appServicePlan 'Microsoft.Web/serverfarms@2021-
02-01' = {
  name: name
```

Продовження лістингу 3.5

```
location: location
  sku: {
    name: skuName
    tier: tier
  }
  tags: tags
}

output id string = appServicePlan.id
```

3.1.4 Модуль сховища динамічної функції

Модуль сховища динамічної функції необхідний для збереження проміжних даних виконання функції, а також для обміну повідомленнями про стан виконання асинхронних операцій. Даний модуль також може використовуватись для збереження файлових даних однією чи декількома функціями. Код даного модуля наведений у лістингу 3.6

Даний модуль приймає на вхід назву сховища, його локацію, список тегів, а також налаштування пов’язані із розміром сховища, його типом і ціною. Як результат роботи модуля, повертаються наступні параметри: рядок з’єднання, ім’я створеного ресурсу, посилання на файлове сховище та чергу повідомлень

Лістинг 3.6 — Модуль сховища динамічної функції

```
param name string
param location string
param tags object
param kind string = 'StorageV2'
param skuName string = 'Standard_LRS'
param publicNetworkAccess string = 'Disabled'

resource storageAccount
'Microsoft.Storage/storageAccounts@2022-09-01' = {
  name: name
  location: location
  kind: kind
  sku: {
    name: skuName
  }
```

Продовження лістингу 3.6

```
    tags: tags
      properties: {
    publicNetworkAccess : publicNetworkAccess
      }
    }
    var accountKey = listKeys(storageAccount.id,
storageAccount.apiVersion).keys[0].value
    output connectionString string =
'DefaultEndpointsProtocol=https;AccountName=${name};EndpointSu
ffix=${environment().suffixes.storage};AccountKey=${accountKey
}'
    output name string = storageAccount.name
    output blobUrl string =
'https://${storageAccount.name}.blob.core.windows.net'
    output queueUrl string =
'https://${storageAccount.name}.queue.core.windows.net'
```

3.1.5 Модуль динамічної функції

Модуль динамічної функції використовується для створення Azure Function ресурсу, що являється контейнером для виконання користувацького коду. Підтримується велика кількість мов програмування

На вхід даного модуля передаються наступні параметри: ім’я функції, локація, список тегів, назва середовища виконання, посилання на сервіс план, ім’я сховища даних, ім’я сховища конфіденційних даних, список конфігурації функції та рядків підключення.

Даний модуль створює функції і налаштовує її рівень доступу до інших ресурсів. Лістинг 3.7 ілюструє створення модулю роботи динамічної функції

Як результат роботи, повертається посилання на функцію

Лістинг 3.7 — Модуль динамічної функції

```
param name string
param location string
param tags object
param environmentType string
param appServicePlanId string
param storageAccountName string
```

Продовження лістингу 3.7

```
param keyVaultName string
param appSettings array
param connectionStrings array = []

resource function 'Microsoft.Web/sites@2022-03-01' = {
  name: name
  location: location
  tags: tags
  kind: 'functionapp'
  identity: {
   type: 'SystemAssigned'
  }
  properties: {
    serverFarmId: appServicePlanId
    httpsOnly: true
    siteConfig: {
      use32BitWorkerProcess: false
      http20Enabled: true
      scmType: 'VSTSRM'
      netFrameworkVersion: 'v7.0'
      appSettings: appSettings
      connectionStrings: connectionStrings
    }
  }
}

module grantPermissions './permissions/grant-
permissions.bicep' = {
  name: '${name}-grant-permissions'
  params: {
    serviceName: function.name
    appId: function.id
    principalId: function.identity.principalId
    environmentType: environmentType
    keyVaultName: keyVaultName
    storageAccountName: storageAccountName
  }
}

output endpoint string =
'https://${function.properties.defaultHostName}'
```

3.1.6 Модуль роботи із конфіденційними даними

Даний модуль використовується для роботи із конфіденційними даними такими, як паролі, ключі, рядки з’єднання та сертифікати. На вхід подається: ім’я ресурсу, локація та список тегів, як результат роботи повертається ім’я створеного ресурсу. Код даного модуля наведений на лістингу 4.8

Лістинг 3.8 — Модуль роботи із конфіденційними даними

```
param name string
param location string
param tags object

resource keyVault 'Microsoft.KeyVault/vaults@2022-07-01' =
{
  name: name
  tags: tags
  location: location
  properties: {
    tenantId: subscription().tenantId
    enableRbacAuthorization: true
    enabledForTemplateDeployment: true
    sku: {
      name: 'standard'
      family: 'A'
    }

    networkAcls: {
      bypass: 'AzureServices'
      defaultAction: 'Allow'
    }
    publicNetworkAccess: 'enabled'
    enableSoftDelete: true
  }
}

output name string = keyVault.name
```

Для наповнення ресурсу конфіденційними даними використовується модуль наведений у лістингу 3.9

Лістинг 3.9 — Наповнення сховища конфіденційними даними

```
param keyVaultName string

@secure()
param someSecret string

@secure()
param cosmosDbConnectionString string

resource CosmosDbConnectionStringSecret
'Microsoft.KeyVault/vaults/secrets@2022-07-01' = {
  name: '${keyVaultName}/CosmosDBConnectionString'
  properties: { value: cosmosDbConnectionString }
}

resource SomeSecret
'Microsoft.KeyVault/vaults/secrets@2022-07-01' = {
  name: '${keyVaultName}/SomeSecret'
  properties: { value: someSecret }
}
```

Деякі конфіденційні дані, можуть бути отримані шляхом створення ресурсів. Наприклад, рядки з’єднань. Для ресурсів, значення яких отримується від зовнішніх систем, імплементовано механізм коли ці дані додаються до головного сховища, і під час запуску IAC конвеєра копіюються до сховища конкретного середовища.

3.1.7 Модуль налаштування доступу

Модуль налаштування доступу налаштовує необхідні доступим між створеними ресурсам, такими як сховище, база даних, сховище конфіденційних даних та функція.

3.2 Реєстрація середовищ на порталі Azure DevOps

Для забезпечення оптимальної роботи рекомендується зареєструвати настумні обчислювальні середовища:

— Dev — використовуються розробниками для написання і відлагодження роботи коду, немає інтеграції із іншими системами;

— Tst — використовується тестувальниками для перевірки роботи коду, містить тестові дані та немає інтеграції з іншими системами;

— Int — використовується тестувальниками для перевірки роботи коду, містить тестові дані та інтегрований із зовнішніми системами;

— Pre — використовується тестувальниками для перевірки роботи коду, містить реальні дані, інтегрований із зовнішніми системами і максимально наближений до реального середовища;

— Prod — середовище роботи реальних користувачів.

Для всіх середовищ, окрім Dev, налаштовується підтвердження розгортання, оскільки написаний код може містити помилки і неточності. Підтвердження розгортання надається у випадку коли весь функціонал або зміни розроблені і їхня якість підтверджена.

3.3 Розробка конвеєру збірки динамічних середовищ

Конвеєр збірки динамічних обчислювальних середовищ необхідний для того, щоб перевірити на помилки, переглянути зміни в середовищі та запустити створення або оновлення існуючого динамічного середовища. Він складається із двох компонентів: шаблону розгортання для одного середовища та вхідного конвеєра, який перечислює всі середовища, де потрібно розгорнути інфраструктуру. Інфраструктура розгортається послідовно на кожному із середовищ. Кожне розгортання інфраструктури повинно бути підтверджене. Повний код конвеєру збірки динамічних середовищ наведений у Додатку В.

Основний код шаблону розгортання наведений у лістингу 3.10. Використання шаблону для розгортання середовища наведено у лістингу 3.11

Лістинг 3.10 — Перегляд та застосування змін інфраструктури обчислювального середовища

```
- task: AzureCLI@2
  displayName: Preview changes
inputs:
    azureSubscription: '${{
variables.AZURE_SRVC_CONNECTION }}'
```

Продовження лістингу 3.10

```
    scriptType: ps
        scriptLocation: inlineScript
        inlineScript:  >
          az deployment group what-if
          --resource-group ${{ variables.RESOURCE_GROUP }}
          --template-file ${{ variables.BICEP_FILE }}
          --parameters  environmentType=${{ parameters.envType
}} `
          region=${{ variables.REGION }} `
          location=${{ variables.LOCATION }} `
          masterKeyVaultName=${{
variables.MASTER_KEY_VAULT_NAME }} `
          masterKeyVaultRg=${{ variables.MASTER_KEY_VAULT_RG
}}

    - task: AzureCLI@2
      displayName: Deploy
      condition: and(succeeded(),
ne(variables['Build.Reason'], 'PullRequest'))
      inputs:
        azureSubscription: '${{
variables.AZURE_SRVC_CONNECTION }}'
        scriptType: ps
        scriptLocation: inlineScript
        inlineScript:  >
          az deployment group create
          --resource-group ${{ variables.RESOURCE_GROUP }}
          --template-file ${{ variables.BICEP_FILE }}
          --parameters  environmentType=${{ parameters.envType
}
    } `
          region=${{ variables.REGION }} `
          location=${{ variables.LOCATION }} `
          masterKeyVaultName=${{
variables.MASTER_KEY_VAULT_NAME }} `
          masterKeyVaultRg=${{ variables.MASTER_KEY_VAULT_RG}}
```

Лістинг 3.11 — Розгортання динамічного обчислювального середовища

```
name: $(Build.SourceBranchName)$(Rev:.r)

stages:
```

Продовження лістингу 3.11

```
    - stage: 'Dev'
      displayName: 'Deploy to dev'
      jobs:
        - template: 'pipelines/templates/iac-template.yml'
          parameters:
            envType: 'dev'

    - stage: 'Tst'
      displayName: 'Deploy to tst'
      condition: and(succeeded(),
ne(variables['Build.Reason'], 'PullRequest'))
      dependsOn: 'Dev'
      jobs:
        - template: 'pipelines/templates/iac-template.yml'
          parameters:
            envType: 'tst'
```

3.4 Імплементація каркасного рішення динамічного обчислювального середовища із автоматичним масштабуванням

Для імплементації каркасного рішення динамічного обчислювального середовища буде використовуватись C#, як мова програмування та Azure Durable Function, як тип програми. В якості шаблону проекту, буде використовуватись Clean Architecture шаблон із наступними проектами:

— Application – містить реалізацію основної логіки програми, обробку запитів та команд;

— Infrastructure – містить імплементацію основних інфраструктурних компонентів, мережеві запити, асинхронні повідомлення, зв’язок із базою даних і тд;

— Function – точка входу в програму, налаштовує проект та його конфігурацію, містить тригери функцій–оркестраторів, функції-оркестратори та функції активностей.

Обробка запитів та команд відбувається зі використанням шаблону медіатору, відповідно до якого необхідно зареєструвати всі команди, запити та компоненти відповідальні за їхню обробку. Коли створений запит передається

медіатору, він вибирає відповідний компонент для його обробки із списку зареєстрованих, створює його об’єкт та викликає інтерфейсний метод обробки Handle(), на вхід якого передається відповідна команда чи запита, та токен відміни.

3.4.1 Імплементація проекту інфраструктури

Відповідно до завдання проект інфраструктури буде містити тільки імплементацію запитів до Cosmos DB, як основної бази даних (Додаток В) та імплементацію репозиторіїв для роботи із активностями.

Першим репозиторієм, який використовується для отримання результатів активностей користувача, є UserActivityRepository, в ньому зберігаються необроблені дані із типом UserActivityDocument. Реалізація UserActivityRepository показана у лістингу 3.12

Лістинг 3.12 — Імлементація UserActivityRepository

```
    public interface IUserActivityRepository:
IDocumentDbRepository<UserActivityDocument>
    {}

    public class UserActivityRepository:
DocumentDbRepository<UserActivityDocument>,
IUserActivityRepository
    {

    public UserActivityRepository(
               IConfiguration configuration,
               IServiceProvider serviceProvider):base(
                serviceProvider,
                configuration["CosmosDB:Database:Name"],

  configuration["CosmosDB:Database:Collections:0"])
        {}
    }
```

Наступний репозиторій UserActivityResultRepository використовується для збереження результатів аналізу і показаний на лістингу 3.13

Лістинг 3.13 — Імплементація UserActivityResultRepository

```
public class UserActivityResultRepository :
DocumentDbRepository<UserActivityDocument>,
IUserActivityResultRepository
{
    public UserActivityResultRepository(IConfiguration
configuration, IServiceProvider serviceProvider) :
base(serviceProvider, configuration["CosmosDB:Database:Name"],
configuration["CosmosDB:Database:Collections:1"])
    {

    }
}


public interface IUserActivityResultRepository :
IDocumentDbRepository<UserActivityDocument>
{

}
```

Обидва компоненти наслідують DocumentDbRepository, що надає можливість використовувати його базові методи для отримання і збереження результатів. У конструктор базового типу передається ім’я бази даних, ім’я колекції, а також компонент, який повертає залежності. Типом для необроблених і проаналізованих даних є UserActivityDocument (лістинг 3.14)

Лістинг 3.14 — Імплементація UserActivityDocument

```
public class UserActivityDocument : DbEntity
{
    [JsonProperty("actions")]
    public int Actions { get; set; }

    [JsonProperty("state")]
    public string State { get; set; }
}
```

3.4.2 Імплементація проекту Application

Проект Application реєструє наступні обробники для команд та запитів:

3.4.2.1 UserActivityQueryHandler – отримує сторінку даних фіксованої кількості відповідно до заданого токена продовження, якщо токен пустий, повертає сторінку даних і список всіх токенів (лістинг 3.15)

Лістинг 3.15 — Імплементація UserActivityQueryHandler

```
    public async Task<UserActivityDocumentsVm>
Handle(UserActivitiesQuery request, CancellationToken
cancellationToken)
    {

        _logger.LogInformation("Getting user activities");

        Expression<Func<UserActivityDocument, bool>>
predicate = document => true;

        (List<string> continuationTokens,
IEnumerable<UserActivityDocument> userActivities) =

    await
_userActivityRepository.GetItemsByPageAsync(_configuration.Get
Value<int>("CosmosDbProcessMaxItems"),
request.ContinuationToken, predicate);

        var result = new UserActivityDocumentsVm()
        {
            UserActivities = userActivities,
            ContinuationTokens = continuationTokens ?? new
List<string>()
        };

        if (result.ContinuationTokens.Any())
result.ContinuationTokens.RemoveAt(0);

        _logger.LogInformation("{itemsCount} user activities
{pagesCount}", userActivities?.Count(),
$"{(result.ContinuationTokens.Any() ? $" and
{result.ContinuationTokens.Count} pages were taken to handle"
: "were taken to handle")}");

        return result;
    }
```

3.4.2.2 AnalizeUserActivityCommandHandler — використовується для імплементації аналізу активностей користувача (лістинг 3.16)

Лістинг 3.16 — Імплементація AnalizeUserActivityCommandHandler

```
    public class AnalizeUserActivityCommandHandler :
IRequestHandler<AnalizeUserActivityCommand,
List<UserActivityDocument>>
    {
        public async Task<List<UserActivityDocument>>
Handle(AnalizeUserActivityCommand request, CancellationToken
cancellationToken)
        {
            foreach (var activity in request.UserActivities)
            {
                await Task.Delay(30);
            }

            return request.UserActivities;
        }
    }
```

3.4.2.3 SaveActivityResultCommandHandler– використовується для збереження результатів аналізу (лістинг 3.17)

Лістинг 3.17 — Імплементація SaveActivityResultCommandHandler

```
    public class SaveActivityResultCommandHandler :
IRequestHandler<SaveActivityResultCommand, Unit>
    {
        private readonly IUserActivityResultRepository
_userActivityResultRepository;

        public
SaveActivityResultCommandHandler(IUserActivityResultRepository
userActivityResultRepository)
        {
            _userActivityResultRepository =
userActivityResultRepository; }

    public async Task<Unit> Handle(SaveActivityResultCommand
request, CancellationToken cancellationToken)
        {
```

Продовження лістингу 3.17

```
    await
_userActivityResultRepository.CreateItemsAsyncBatch(request.Us
erActivities, t => t.Id);
            return Unit.Value;
    }
    }
```

3.4.3 Імплементація проекту Function

Проект Function має 3 вхідних функції:

— GeneratorExecutionStatus – приймає на вхід HTTP запит із ідентифікатором екземпляру, як результат виконання повертається список посилань для перевірки статусу та контролю поточного екземпляру;

— ActivityHttpTrigger – HTTP точка входу, запускає процес аналізу;

— ActivityTimerTrigger – точка входу яка, запускається із певним інтервалом.

ActivityHttpTrigger і ActivityTimerTrigger запускають QueryActivitiesOrchestrator, основна задача якого, це отримання великих масивів даних розбиття їх на частини і відправка на опрацювання в ProcessActivitiesOrchestrator підоркестратор. ProcessActivitiesOrchestrator отримує масив даних, опрацьовує його, та зберігає результати.

Операції отримання, обробки та збереження даних виконуються за допомогою функцій активностей. Операція отримання даних показана на лістингу 3.18

Лістинг 3.18 — Отримання активності користувача

```
    [Function("GeUserActivities")]
    public async Task<UserActivityDocumentsVm>
GetCosmosDbDocuments([ActivityTrigger] string
continuationToken, FunctionContext executionContext)
        {

    var logger =
executionContext.GetLogger(nameof(ActivityFunctions));
        var query = new UserActivitiesQuery()
```

Продовження лістингу 3.18

```
{ ContinuationToken = continuationToken };
try
{
     return await _mediator.Send(query);
}
catch (Exception ex)
{
logger.LogError(ex, "Failed to get user activity\nError –
{message}", ex.Message); }
```

3.5 Розробка конвеєру збірки для каркасного рішення

Конвеєр збірки каркасного рішення виконує дві функції: зібрати артефакт на основі скомпільованого вихідного коду і завантажити його на середовище, яке використовується для перевірки програмістом.

Вхідний файл зберігається в репозиторії каркасної функції. Завдання вхідного файлу підключити шаблон із IAC репозиторію і передати йому необхідні параметри, як це зображено у лістингу 3.19

Лістинг 3.19 — Вхідний файл каркасної функції

```
name: $(Build.SourceBranchName)$(Rev:.r)
trigger:
- release*

resources:
  repositories:
    - repository: templates
      type: git
      name:
DynamicComputingEnvironments/DynamicComputingEnvironments.IAC
      ref: release/1.0.0

variables:
  - template: 'build-variables.yml'
stages:

 - template: pipelines/templates/ci-cd-stage-
template.yml@templates
     parameters:
```

Продовження лістингу 3.19

```
srvcName: '${{ variables.SRVC_NAME }}'
      srvcType: '${{ variables.SRVC_TYPE }}'
publishProj: '${{ variables.PUBLISHING_PROJECT }}'
projectParentFolder: '${{ variables.PROJECT_FOLDER }}'
```

Вхідний файл каркасної функції підключає файл шаблону збірки і завантаження коду на середовище для програміста. Також є перевірка для процесу завантаження коду, чи даний запуск не є пул реквестом, в такому разі відбувається лише збірка, без завантаження коду на середовище розробника, як показано у лістингу 3.20

Лістинг 3.20 — Шаблон збірки і завантаження коду

```
parameters:
  srvcName: ''
  srvcType: ''
  publishProj: ''
  projectParentFolder: ''

stages:
- stage: Build
  displayName: 'Build Solution'
  jobs:
    - template: ci-template.yml@templates
      parameters:
        publishingProject: '${{ parameters.publishProj }}'
        projectParentFolder: '${{
parameters.projectParentFolder }}'

- ${{ if ne(variables['Build.Reason'], 'PullRequest') }}:
  - stage: DeployDev
    displayName: 'Deploy to dev'
    condition: succeeded()
    dependsOn: Build
    variables:

- template: '../variables/dev-variables.yml'
    jobs:
      - template: cd-${{ parameters.srvcType }}-
template.yml@templates
```

Продовження лістингу 3.20

```
 parameters:
envType: dev
srvcName: '${{ parameters.srvcName }}'
region: '${{ variables.REGION }}'
publishingProject: '${{ parameters.publishProj }}'
```

Шаблон збірки проекту містить основні кроки для компіляції проекту і верифікації якості коду, такі як: завантаження всіх залежностей коду, компіляція коду, перевірка стилю написання коду, запуск модульних тестів та перевірка відсотку покриття коду тестами (лістинг 3.21)

Лістинг 3.21 — Компіляція проекту каркасної функції

```
parameters:
  buildConfiguration: 'Release'
  unitTestsPath: ''

steps:
  - task: UseDotNet@2
    displayName: 'Install .NET 7 SDK'
    inputs:
      packageType: 'sdk'
      version: '7.0.x'
      includePreviewVersions: true

  - task: DotNetCoreCLI@2
    displayName: 'Restore Nuget Packages'
    inputs:
      command: 'restore'
      projects: '**/*.csproj'
      feedsToUse: 'select'

  - task: DotNetCoreCLI@2
    displayName: 'Build Service'
    inputs:
    command: 'build'
      projects: '**/*.csproj'
          arguments: '--configuration ${{
     parameters.buildConfiguration }} --no-restore'
  - task: DotNetCoreCLI@2
    displayName: 'Run Unit Tests With Coverage'
```

Продовження лістингу 3.21

```
    inputs:
             command: 'test'
         projects: '${{ parameters.unitTestsPath }}'
             arguments: '--configuration ${{
   parameters.buildConfiguration }} /p:CollectCoverage=true
   /p:CoverletOutputFormat=cobertura --no-build'
```

Результатом роботи збірки каркасної функції буде публікація артефакту, який містить скомпільований код програми, а також може містити файли необхідні для конвеєру завантаження під час наступних кроків. (лістинг 3.22)

Лістинг 3.22 — Публікація артефактів проекту

```
    - task: DotNetCoreCLI@2
      displayName: 'Publish service'
      condition: and(succeeded(),
ne(variables['Build.Reason'], 'PullRequest'))
      inputs:
        command: 'publish'
        publishWebProjects: false
        zipAfterPublish: true
        projects: '**/${{ parameters.publishingProject
}}.csproj'
        arguments: '-c $(buildConfiguration) -o
$(Build.StagingDirectory)/Publish --no-build'
        modifyOutputPath: true

    - task: PublishPipelineArtifact@1
      displayName: 'Publishing Artifact ...'
      condition: and(succeeded(),
ne(variables['Build.Reason'], 'PullRequest'))
      inputs:
      targetPath: '$(Build.StagingDirectory)/Publish'
      publishLocation: 'pipeline'
      artifactName: 'Artifacts'
```

Після завершення процесу публікації автоматично запускається процес завантаження артефактів у обчислювальне середовище розробника. Даний процес ідентичний відповідним процесам для інших середовищ і буде розглянутий детально у розділі 3.6

3.6 Розробка конвеєру завантаження для каркасного рішення

Для конвеєру завантаження точкою входу є файл dce–controller–prd.yml. Даний файл підключає IAC репозиторій, який містить необхідні шаблони. Також додається файл із змінними, які описують конфігурацію розгортання для кожного із середовищ. Заключним етапом є виклик шаблону розгортання із передачею йому відповідних параметрів (лістинг 3.23).

Лістинг 3.23 — Конвеєр розгортання каркасної функції

```
name: $(Build.SourceBranchName)$(Rev:.r)
resources:
  repositories:
    - repository: templates
      type: git
      name:
DynamicComputingEnvironments/DynamicComputingEnvironments.IAC
      ref: release/1.0.0
variables:
  - template: 'build-variables.yml'
stages:
  - template: 'pipelines/templates/ci-cd-prd-
template.yml@templates'
    parameters:
      srvcName: '${{ variables.SRVC_NAME }}'
      srvcType: '${{ variables.SRVC_TYPE }}'
      publishProj: '${{ variables.PUBLISHING_PROJECT }}'
```

Шаблон розгортання містить список обчислювальних середовищ, на які потрібно завантажити код каркасної функції. Кожне середовище має залежність від попереднього. Тобто спочатку необхідно завантажити код на тестове середовище, після чого з`являється можливість завантаження на середовище інтеграції. Також для завантаження коду на середовище, необхідно пройти процедуру підтвердження дозволу. В даному випадку, коли програмне забезпечення протестовано і виникає необхідність завантаження, програміст розпочинає процес, а інша людина з відповідним рівнем доступу повинна відкрити процес збірки і надати дозвіл на продовження. Тільки після цього

процес завантаження розпочнеться. На лістингу 3.24 показано процес розгортання коду на тестове та інтеграційні обчислювальні середовища.

Лістинг 3.24 — Шаблон розгортання коду

```
parameters:
  srvcName: ''
  srvcType: ''
  publishProj: ''
stages:
- stage: 'Tst'
  displayName: 'Deploy to tst'
variables:
      - template: '../variables/tst-variables.yml'
  jobs:
    - template: 'cd-${{ parameters.srvcType }}-
template.yml@templates'
      parameters:
        envType: tst
        srvcName: '${{ parameters.srvcName }}'
        region: '${{ variables.REGION }}'
        publishingProject: '${{ parameters.publishProj }}'
- stage: 'Int'
  displayName: 'Deploy to int'
  condition: succeeded()
  dependsOn: 'Tst'
  variables:
      - template: '../variables/int-variables.yml'
  jobs:
    - template: 'cd-${{ parameters.srvcType }}-
template.yml@templates'
      parameters:
        envType: int
        srvcName: '${{ parameters.srvcName }}'
region: '${{ variables.REGION }}'
        publishingProject: '${{ parameters.publishProj }}'
```

Кожне окреме середовище використовує шаблон на основі типу ресурсу і підключає файл із конфігурацією для розгортання. Для процесу розгортання використовується дві основні задачі: завантаження артефакту та його розгортання у середовищі (лістинг 3.25)

Лістинг 3.25 — Розгортання артефакту у середовищі

```
    - task: DownloadPipelineArtifact@2
      inputs:
        source: '$(source)'
        project: $(System.TeamProjectId)
        pipeline: '[DCE]${{ parameters.publishingProject }}-
dev'
        runVersion: 'latestFromBranch'
        runBranch: $(Build.SourceBranch)
    - task: AzureFunctionApp@1
      displayName: 'Azure Function App Deploy'
      inputs:
        azureSubscription: '${{
variables.AZURE_SRVC_CONNECTION }}'
        appType: 'functionApp'
        appName: 'dce-${{ parameters.srvcName }}-${{
parameters.envType }}-${{ parameters.region }}-af'
        package: '$(Pipeline.Workspace)/Artifacts/*.zip'
        deploymentMethod: 'auto'
```

4 ВЕРИФІКАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Після додавання контейнерів збірки і розгортання та налаштування списку динамічних обчислювальних середовищ, портал Azure Devops відображає список конвеєрів як показано на рисунку 4.1.

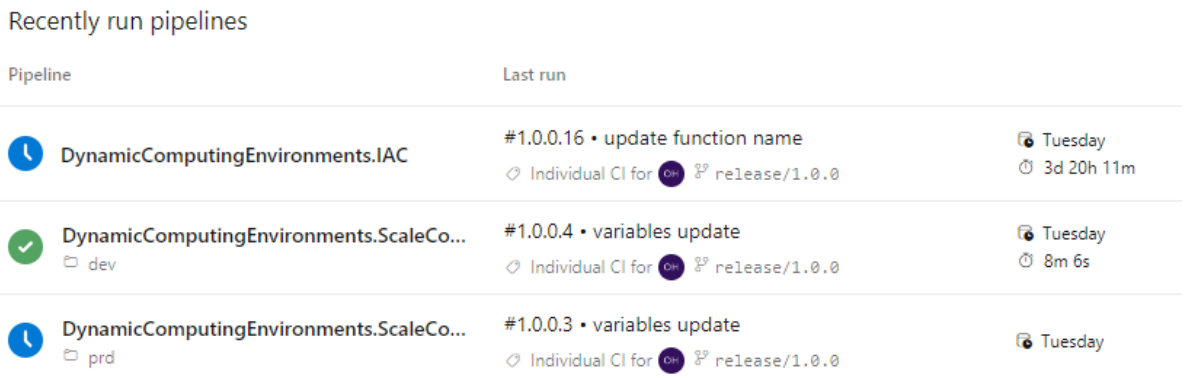


Рисунок 4.1 — Список конвеєрів збірки і розгортання

Для верифікації отриманих результатів необхідно підтримати підтвердження успішності виконання даних конвеєрів, правильної роботи каркасної функції, масштабування ресурсів та вартості розміщення основних компонентів.

4.1 Верифікація результатів роботи IAC конвеєра

Для перевірки роботи IAC конвеєра його потрібно запусти із списку конвеєрів і дочекатись завершення роботи. У випадку успішного запуску і відпрацювання список запусків повинен мати запуск з успішним індикатором, як це зображено на рисунку 4.2

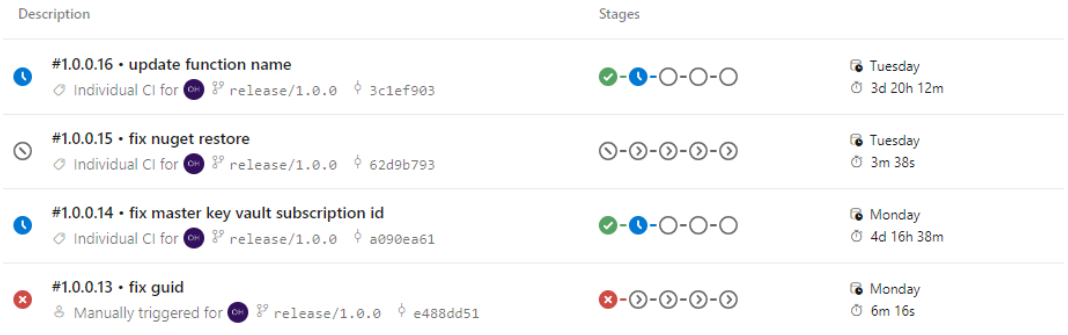


Рисунок 4.2 – Список запусків IAC конвеєра

Для перегляду деталей конкретного запуску необхідно натиснути на нього, після чого буде відображений список динамічних обчислювальних середовищ створених або оновлених за допомогою даного конвеєра. Рисунок 4.3 відображає даний список, а також роботу механізму підтвердження розгортання на інтеграційне середовище

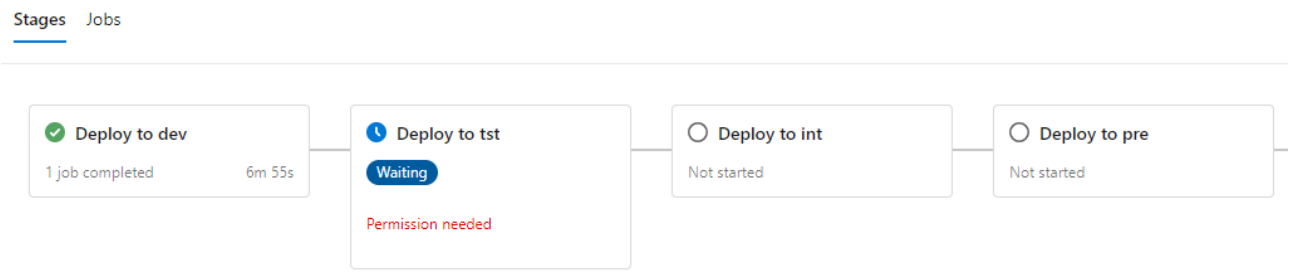


Рисунок 4.3 — Розгортання IAC коду у середовищі розробника

Для перегляду конкретних дій виконаних у середовищі потрібно натиснути на його назву, після чого відбудеться навігація на покрокові результати роботи конвеєра. Дане вікно містить повний журнал операцій для кожного виконаного кроку, що дозволяє відслідковувати результат і виявляти помилки під час виконання (рисунок 4.4)

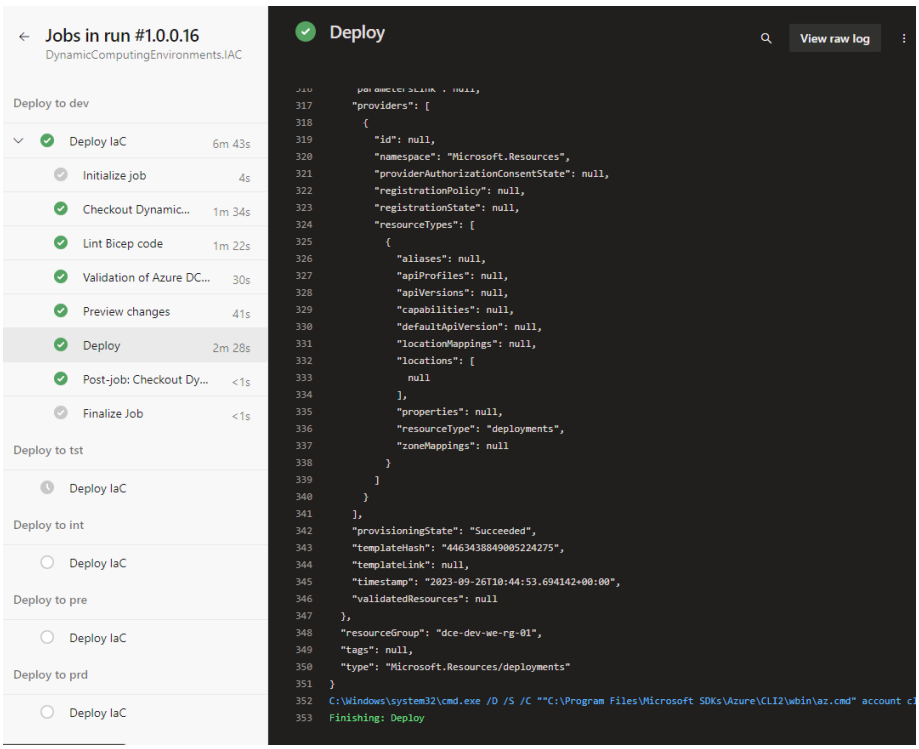
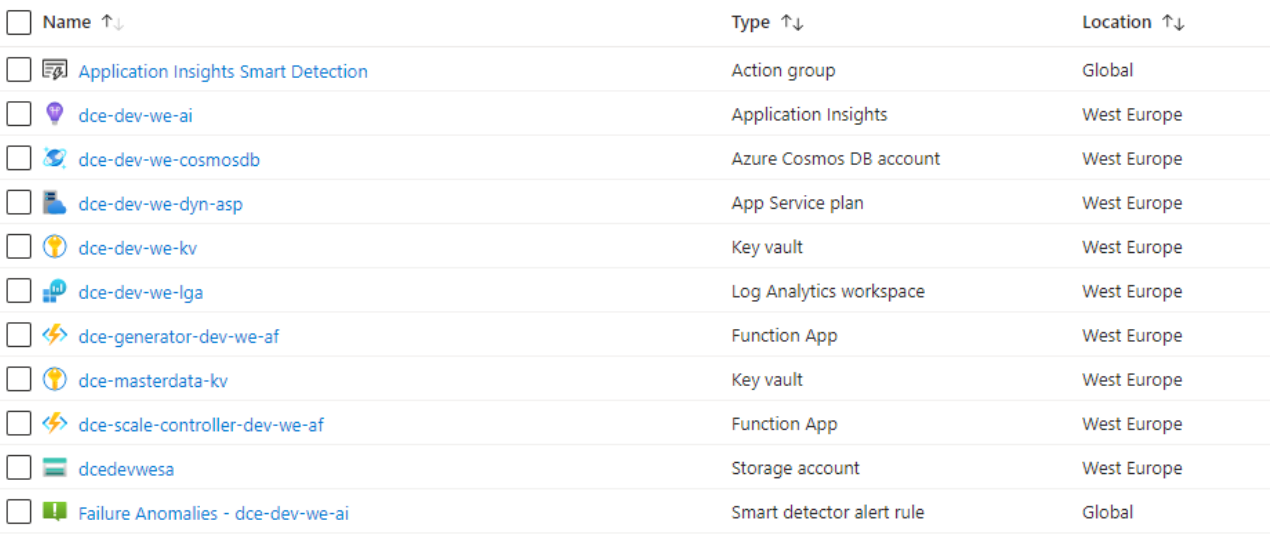


Рисунок 4.4 — Покроковий результат роботи конвеєра

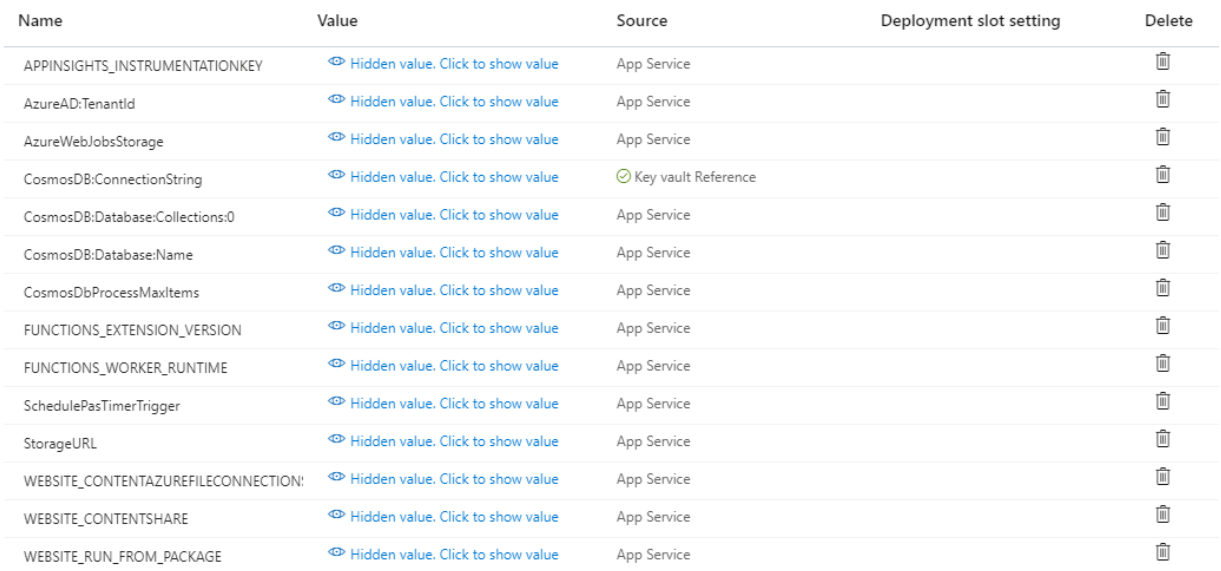
Після верифікації роботи IAC конвеєра потрібно перейти у Azure Portal, та перевірити наявність створених ресурсів та їх конфігурації. Для даної перевірки була створена ресурс група із ім`ям dce–dev–we–rg–01, в рамках якої і відбувалось розгортання. Список створених ресурсів показаний на рисунку 4.5



Name	Type	Location
Application Insights Smart Detection	Action group	Global
dce-dev-we-ai	Application Insights	West Europe
dce-dev-we-cosmosdb	Azure Cosmos DB account	West Europe
dce-dev-we-dyn-asp	App Service plan	West Europe
dce-dev-we-kv	Key vault	West Europe
dce-dev-we-lga	Log Analytics workspace	West Europe
dce-generator-dev-we-af	Function App	West Europe
dce-masterdata-kv	Key vault	West Europe
dce-scale-controller-dev-we-af	Function App	West Europe
dcedevwesa	Storage account	West Europe
Failure Anomalies - dce-dev-we-ai	Smart detector alert rule	Global

Рисунок 4.5 — Вміст ресурс групи dce–dev–we–rg–01

Оскільки основним компонентом інфраструктури є каркасна функція, необхідно також відкрити її конфігурацію і підтвердити правильність встановлених значень, оскільки вони були згенеровані і встановлені за допомогою IAC конвеєру (рисунок 4.6)



Name	Value	Source	Deployment slot setting	Delete
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to show value	App Service		
AzureAD:TenantId	Hidden value. Click to show value	App Service		
AzureWebJobsStorage	Hidden value. Click to show value	App Service		
CosmosDB:ConnectionString	Hidden value. Click to show value	Key vault Reference		
CosmosDB:Database:Collections:0	Hidden value. Click to show value	App Service		
CosmosDB:Database:Name	Hidden value. Click to show value	App Service		
CosmosDbProcessMaxItems	Hidden value. Click to show value	App Service		
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to show value	App Service		
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to show value	App Service		
SchedulePasTimerTrigger	Hidden value. Click to show value	App Service		
StorageURL	Hidden value. Click to show value	App Service		
WEBSITE_CONTENTAZUREFILECONNECTION!	Hidden value. Click to show value	App Service		
WEBSITE_CONTENTSHARE	Hidden value. Click to show value	App Service		
WEBSITE_RUN_FROM_PACKAGE	Hidden value. Click to show value	App Service		

Рисунок 4.6 — Конфігурація каркасної функції

4.2 Верифікація роботи конвеєра збірки та розгортання каркасної функції

Для верифікації роботи конвеєра збірки та розгортання каркасної функції необхідно виконати схожі дії, як для IAC конвеєра. Потрібно запустити конвеєр збірки, після чого в списку запусків повинен з`явитись новий запуск із успішним індикатором виконання (рисунок 4.7)

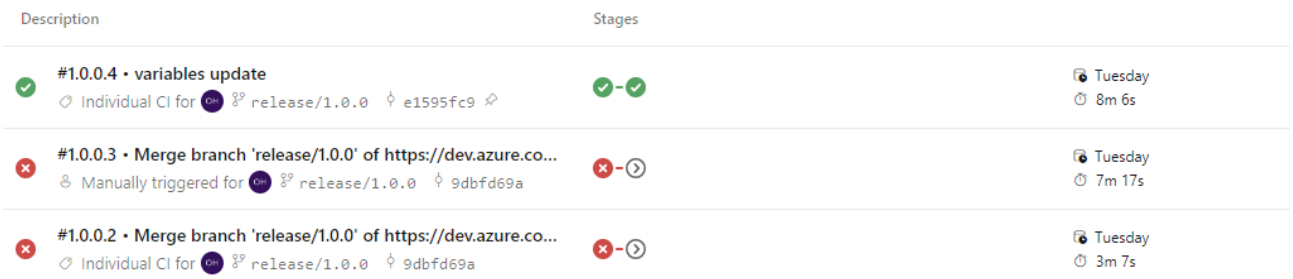


Рисунок 4.7 — Список запусків конвеєра збірки і розгортання каркасної функції

Для того, щоб переглянути деталі конкретного запуску, необхідно обрати його із списку і натиснути на нього, після чого відобразиться наступний екран, який показує список завдань, які були виконанні і їхній статус. Даний конвеєр має два завдання: збірка програмного коду і публікація його на середовище розробника (рисунок 4.8)

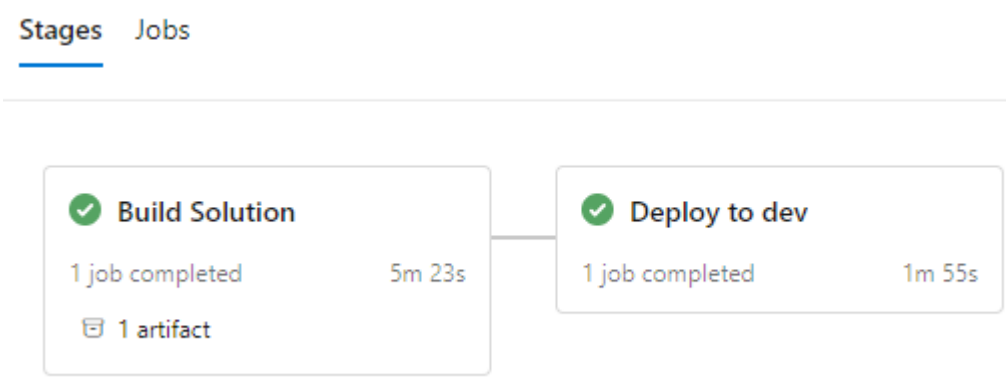


Рисунок 4.8 — Статус завдань збірки і розгортання каркасної функції

Останнім етапом буде перевірка покрокових інструкцій для кожного із завдань. Для цього необхідно натиснути на будь–яке завдання, відбудеться навігація на сторінку покрокових деталей. Дана сторінка буде містити усі деталі,

а також журнал подій, які відбулись під час роботи даного конвеєра (рисунок 4.9).

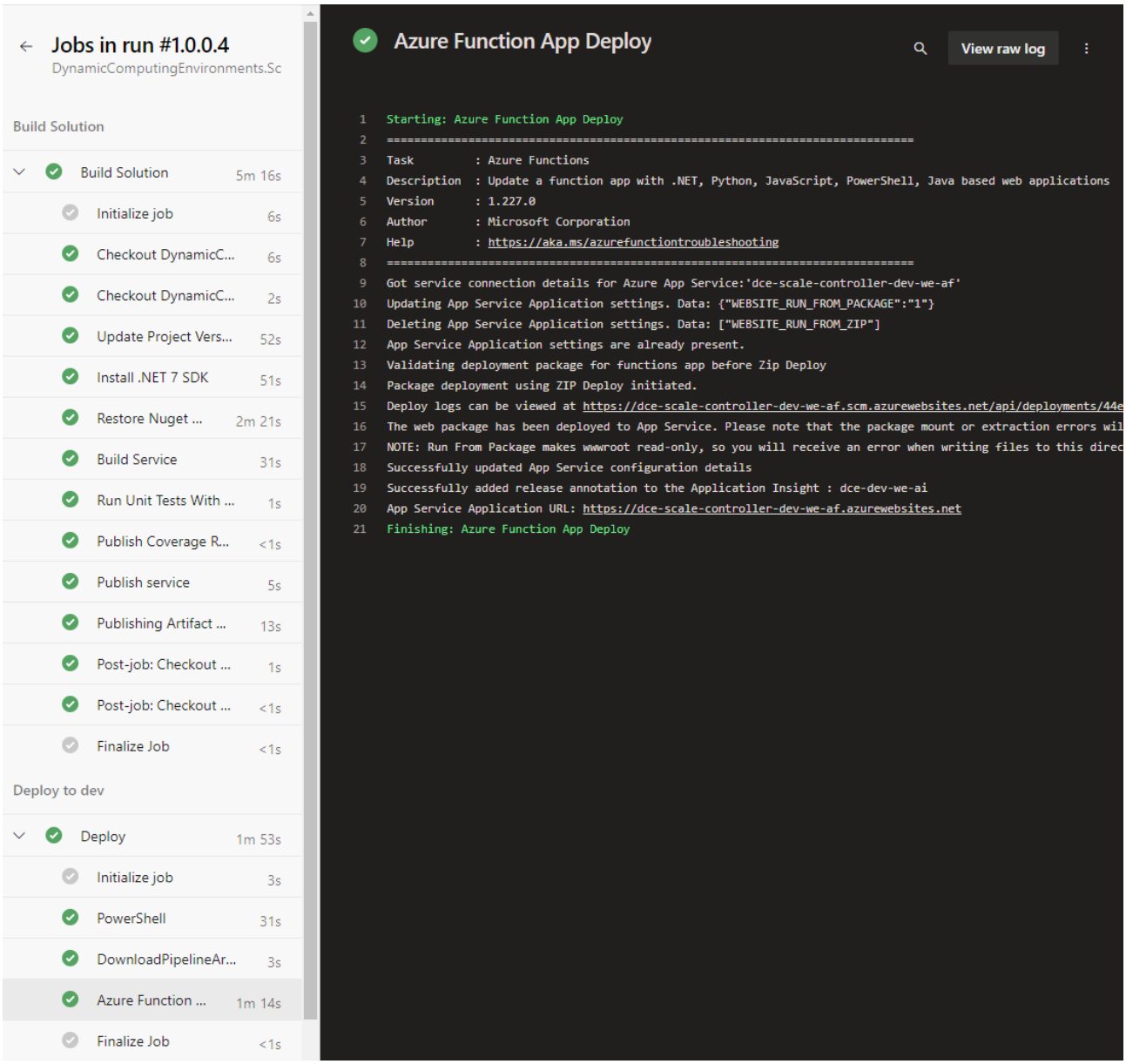


Рисунок 4.9 — Результат виконання конвеєру збірки і розгортання каркасної функції

4.3 Верифікація роботи каркасної функції

Після запуску каркасної функції відображається список усіх імплементованих функцій. У випадку помилок вони будуть показані у консолі червоним кольором. Запуск каркасної функції показаний на рисунку 4.10

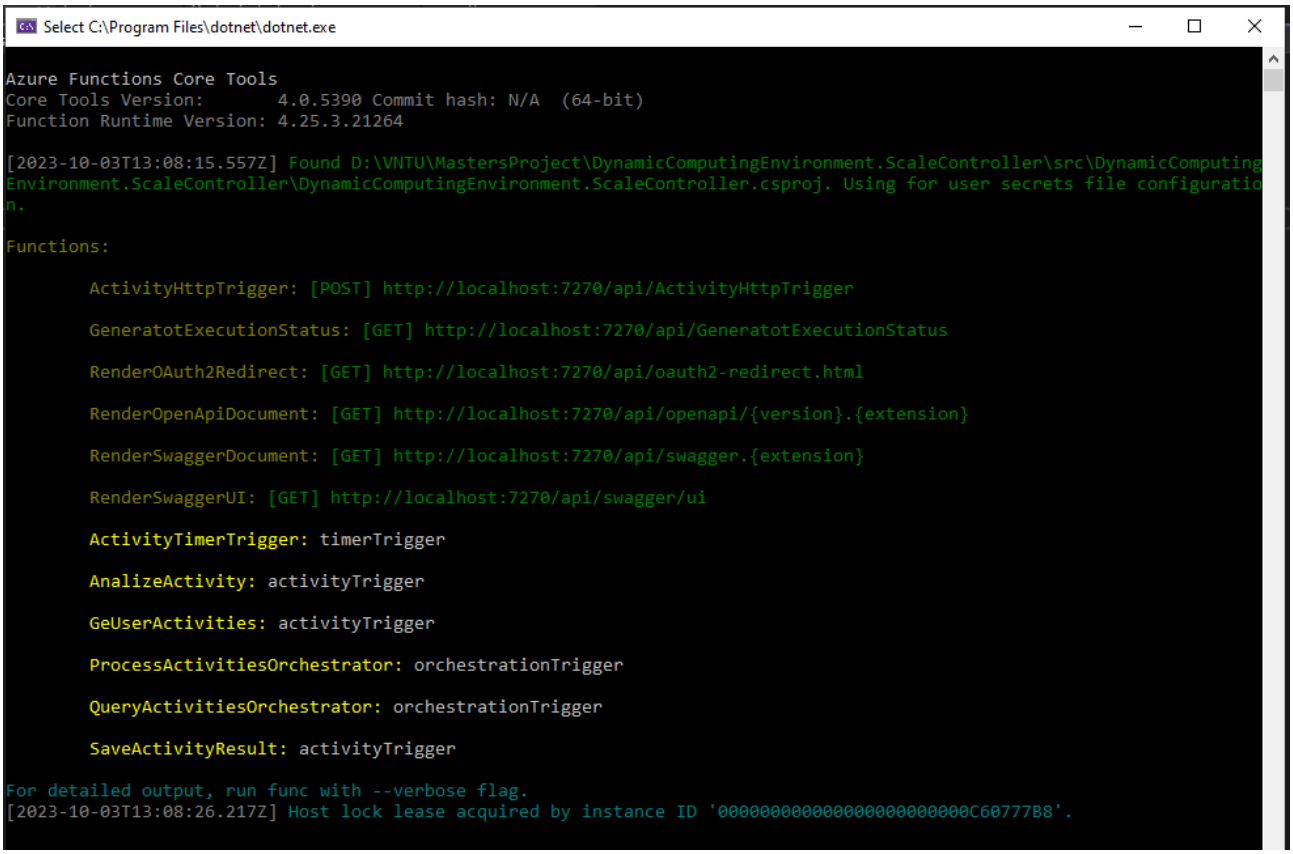


Рисунок 4.10 — Запуск каркасної функції

Для запуску процесу аналізу за допомогою HTTP виклику потрібно запустити ActivityHttpTrigger. Як результат виклику буде повернено список посилань, для контролю поточного запуску, як це показано на рисунку 4.11

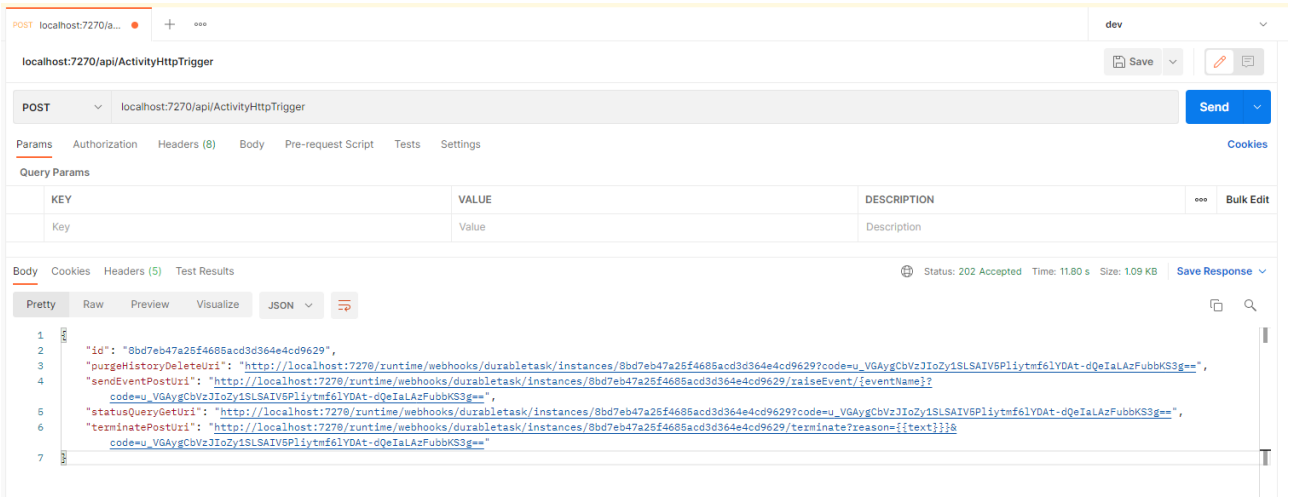


Рисунок 4.11 — Запуск процесу аналізу

Для перевірки поточного статусу виконання необхідно викликати statusQueryGetUrl. Статус буде відображено, відповідно до рисунку 4.12

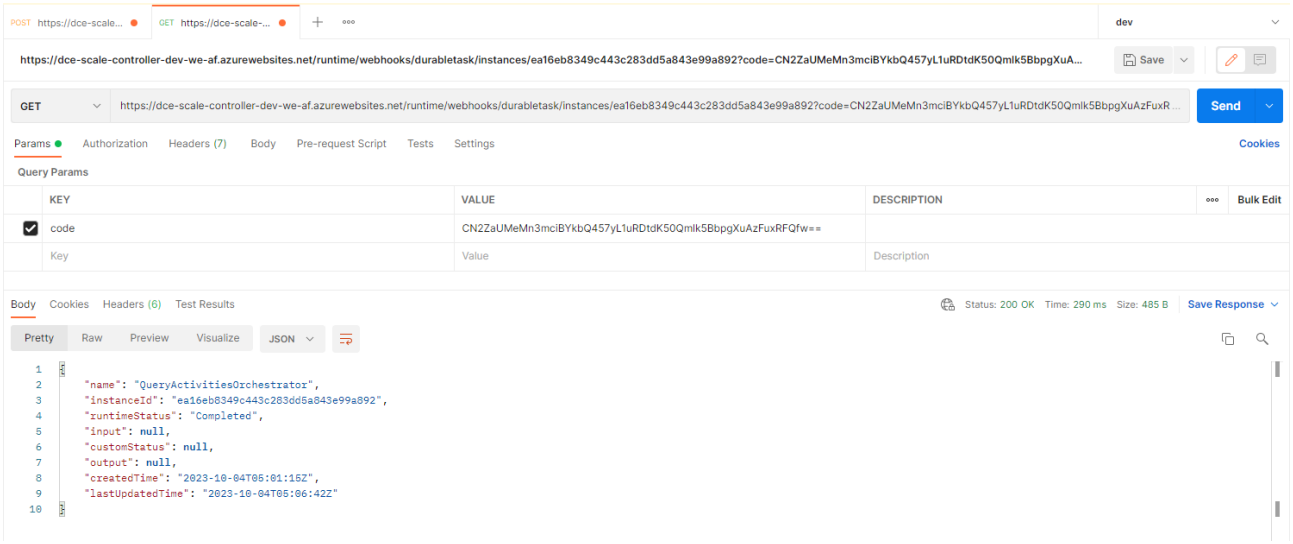


Рисунок 4.12 — Статус виконання аналізу

Після завершення процесу аналізу, результати будуть збережені у Cosmos DB базі даних, яка матиме назву DCE, у колекції UserActivitiesResults. Результати аналізу відображенні на рисунку 4.13

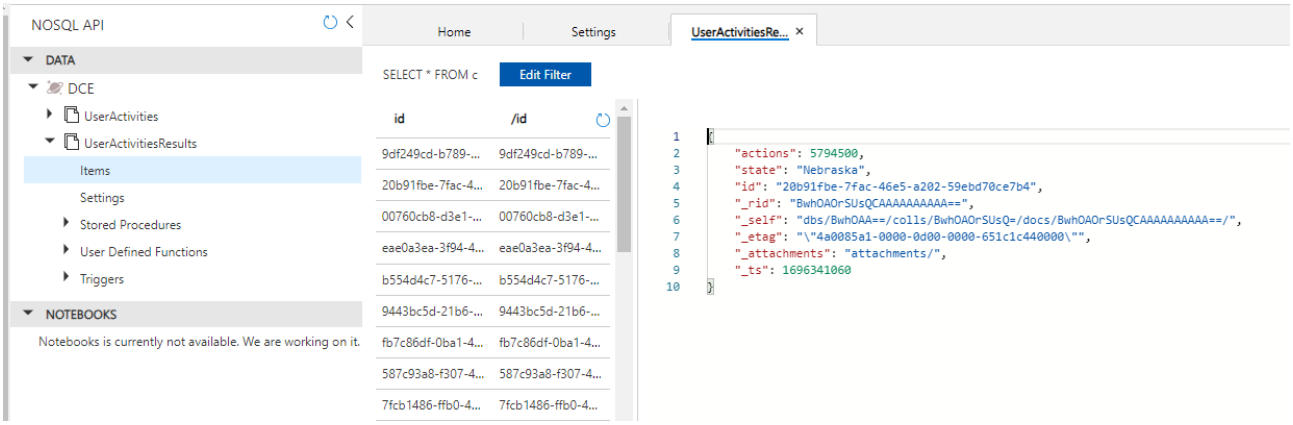


Рисунок 4.13 — Результати аналізу активності користувача

4.4 Верифікація моделі масштабування

Для перевірки роботи масштабування каркасної функції необхідно запустити її у динамічному обчислювальному середовищі із певним рівнем масштабування. Функція автоматично буде масштабуватись відповідно до

навантаження. Підтвердження даного процесу можна отримати у Application Insights у режимі моніторингу у реальному часу як це зображено на рисунку 4.14. Відповідно до рисунку, функція автоматично масштабувалась до двох екземплярів. Дані екземпляри опрацьовують запити паралельно. Коли кількість запитів зменшиться, функція зменшить кількість екземплярів до одного. Якщо запити до функції припиняться, функція вимкне останній екземпляр і перейде в режим очікування

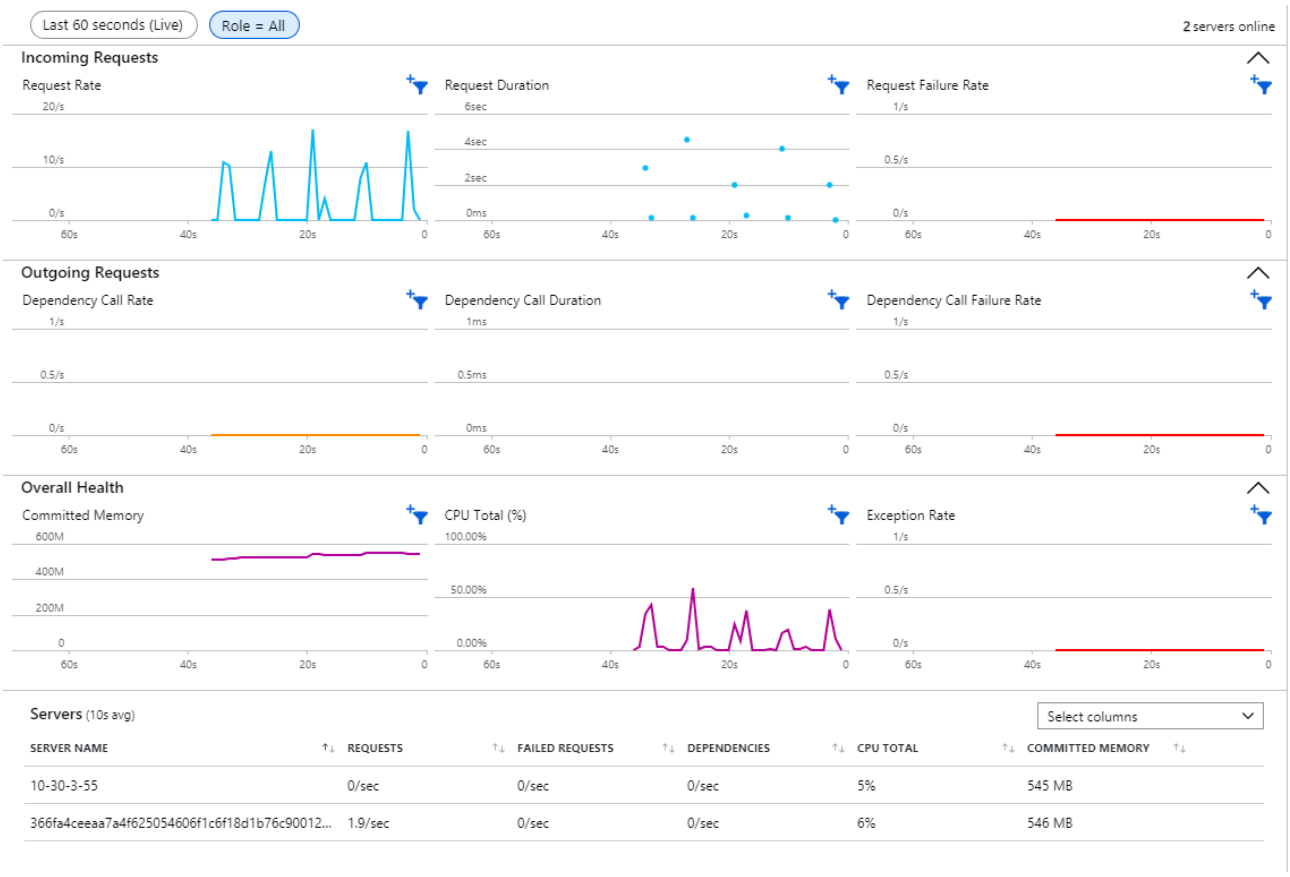


Рисунок 4.14 — Масштабування каркасної функції

4.5 Верифікація моделі оплати

Для того, щоб провести аналіз моделі оплати, необхідно провести аналіз коштів, необхідний для ресурс групи, як це показано на рисунку 4.15. До впровадження змін, вартість середовища, без обчислень становила 1.50$. Це яскраво видно за період із 25 вересня по 30 вересня. Із врахуванням внесених

змін, поточна вартість середовища становить 0.05$ (період з 30 вересня по 2 жовтня).

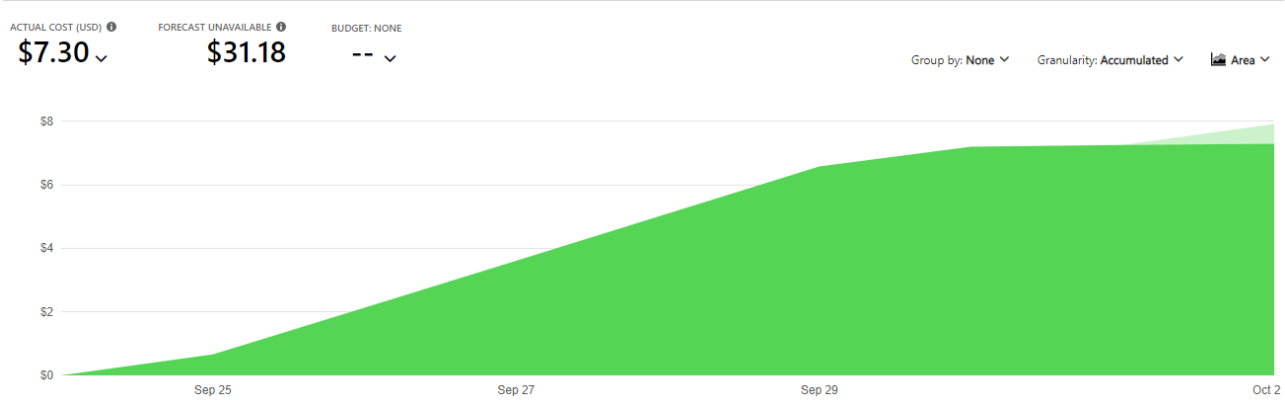


Рисунок 4.15 — Вартість динамічного обчислювального середовища

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково–технічної діяльності.

Магістерська кваліфікаційна робота за темою “Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ” передбачає розробку системи масштабування обчислювальних ресурсів в залежності від порогових значень.

Ціна аналогу становить 125600 грн/місяць.

Проведемо оцінювання комерційного потенціалу даної розробки. Для проведення технологічного аудиту було залучено 3–х незалежних експертів: Захарченко Сергій Михайлович – керівник магістерської кваліфікаційної роботи, професор, викладач кафедри комп’ютерних наук; Кадук Олександр Володимирович, Крупельницький Леонід Віталієвич. В таблиці 4.1 наведено критерії оцінювання комерційного потенціалу розробки та їх оцінки в балах

Таблиця 4.1 — Критерії оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5–ти бальною шкалою)					
Кри–терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 4.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 4.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово–промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10–ти років	Термін реалізації ідеї від 3–х до 5–ти років. Термін окупності інвестицій більше 5–ти років	Термін реалізації ідеї менше 3–х років. Термін окупності інвестицій від 3–х до5–ти років	Термін реалізації ідеї менше 3–х років. Термін окупності інвестицій менше 3–х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові–домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь–які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки потрібно звести в таблицю за зразком таблиці 4.2.

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1–Захарченко С.М.	2–Кадук О.В.	3–Крупельницький Л.В.
	Бали, виставлені експертами:		
1	3	3	4
2	4	3	4
3	3	3	3
4	3	2	3
5	2	2	3
6	4	4	4
7	2	3	3
8	4	3	3
9	3	4	3
10	4	3	4
11	3	3	3
12	4	4	2
Сума балів	$СБ_1$=39	$СБ_2$=37	$СБ_1$=39
Середньоарифметична сума балів СБ	$\overline{СБ}\frac{\sum_1^3 СБ_i}{3} = \frac{115}{3} = 38.33$		

За даними таблиці 4.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 4.3.

Рівень комерційного потенціалу розробки, становить 38,33 балів, що відповідає рівню «вище середнього».

Сфера розробки технологій масштабування є досить розвинутою, оскільки в сучасному світі зростає затребуваність хмарних рішень і технологій управління

ресурсами. Таким чином масштабування хмарних обчислювальних ресурсів за допомогою порогових значень є актуальним

Таблиця 4.3 — Рівні комерційного потенціалу розробки.

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 –10	Низький
11–20	Нижче середнього
21–30	Середній
31–40	Вище середнього
41–48	Високий

5.2 Прогнозування витрат на виконання наукової роботи та впровадження результатів

Проведемо прогнозування витрат на виконання науково–дослідної, дослідно–конструкторської та конструкторсько–технологічної роботи для розробки програмного забезпечення, яке складається з таких етапів:

1) розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

2) розрахунок загальних витрат на виконання даної роботи;

3) прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

5.2.1 Розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи

5.2.1.1 Виконаємо розрахунок витрат приймаючи до уваги те, що для розробки інформаційної технології було залучено одного розробника програмного забезпечення. Основна заробітна плата кожного із розробників (дослідників) $З_о$, якщо вони працюють в наукових установах бюджетної сфери:

$$З_о = \frac{М}{T_р} \cdot t \text{ [грн]}, \quad (4.1)$$

де М — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн;

$T_р$ — число робочих днів в місяці; приблизно $T_р$ = (21…23) дні;

t — число робочих днів роботи розробника (дослідника), розробка програмного забезпечення триває 80днів.

Зроблені розрахунки внесені до таблиці 4.5:

Таблиця 4.4 — Основна заробітна плата розробників.

Найменування посадивиконавця	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на оплату праці,грн.	Примітка
Програміст	40000	1818,18	80	145454,4	
Науковець	20000	909,09	80	72727,2	
Всього				$\sum З_о$	218181,6

5.2.1.2 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 розробників та робітників розраховується за формулою:

$$З_Д = 0.10 \cdot 218181{,}6 = 21818{,}16(\text{грн}).$$

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

$$Н_{зп} = (З_0 + З_Д) \cdot \frac{\beta}{100} \text{ [грн]}, \quad (4.2)$$

де $З_0$ — основна заробітна плата розробника, грн.;

$З_Д$ — додаткова заробітна плата розробника, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування , 22%.

$$Н_{ЗП} = (218181{,}6 + 21818{,}16) \cdot 0{,}22 = 52799{,}95(\text{грн})$$

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

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

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

$$А = \frac{Ц \cdot Т}{12 \cdot Т_В}[\text{грн}], \quad (4.3)$$

де Ц — загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн;

Т — фактична тривалість використання, міс;

$Т_В$— термін, використання обладнання, приміщень тощо, місяці, роки.

Зроблені розрахунки наведено в таблиці 4.4

Під час розробки програмного продукту використовувались лише безкоштовні програмні засоби.

5.2.1.5 Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \ [грн], \quad (4.4)$$

де В — вартість 1 кВт–год. електроенергії, 6.4 грн/кВт;

П — установлена потужність обладнання, кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_п$ — коефіцієнт використання потужності;

Таблиця 4.5 — Амортизаційні відрахування

Найменування	Балансова вартість, грн	Термін використання, роки	Фактична тривалість використання, міс.	Величина амортизаційних відрахувань, грн
Офісне приміщення	500000	20	4	8333.33
Ноутбук	21000	2	4	3500
Всього				11833.33

Потужність використовуваного комп'ютера становить П=0.6 кВт.

Фактична кількість годин роботи обладнання – 640 год (80 робочих днів по 8 годин на день).

$$В_е = 6.4 \cdot 0{,}6 \cdot 640 \cdot 0{,}6 = 1428.48 \text{ (грн)}.$$

5.2.1.6 Сума всіх попередніх статей витрат дає витрати на виконання даної частини розділу роботи В.

$$В = 218181{,}6 + 21818{,}16 + 52799{,}95 + 11833{,}33 + 1428{,}48 = 306061{,}52$$

5.2.2 Розрахунок загальних витрат на виконання даної роботи.

$$В_{заг} = \frac{В}{\alpha}[грн], \quad (4.5)$$

$$В_{заг} = \frac{306061{,}52}{2} = 153030{,}76[грн],$$

5.2.3 Прогнозування загальних витрат на виконання та впровадження результатів виконаної роботи.

5.2.3.1 Прогнозування витрат ЗВ на виконання та впровадження виконаної роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} [грн], \quad (4.7)$$

де β — коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Так, як розробка знаходиться:

— на стадії розробки дослідного зразка, то $\beta \approx 0{,}5$;

— на стадії технічного проектування , то $\beta \approx 0{,}2$;

— на стадії розробки конструкторської документації то $\beta \approx 0{,}3$;

— на стадії розробки технології, то $\beta \approx 0{,}4$;

— на стадії науково–дослідних робіт, то $\beta \approx 0{,}1$;

— на стадії промислового зразка, $\beta \approx 0{,}7$;

— на стадії впровадження, то $\beta \approx 0{,}9$.

$$ЗВ = \frac{153030{,}76}{0{,}7} = 218615{,}37 \text{ (грн)}.$$

Отже, прогноз загальних витрат на виконання та впровадження результатів становить 218615,37 грн.

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. В умовах ринку узагальнюючим позитивним результатом, що

його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей.

Зростання чистого прибутку забезпечить підприємству надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік.

Позитивні результати від впровадження розробки очікуються на другий рік впровадження.

Проведемо детальніше прогнозування позитивних результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_я = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_n \text{ [грн]}, \quad (4.8)$$

де $\Delta П_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ — основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки покращується якість, що дозволяє підвищити ціну його реалізації на 200 грн, а кількість одиниць реалізованої послуги збільшиться: протягом першого року на

0 од., протягом другого року на 5000 од., протягом третього року ще на 3500 од.

Орієнтовно: реалізація послуг до впровадження результатів наукової розробки складала 15 шт., а її ціна — 150рн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1 = 0 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2 = (15+150)\ (5000) = 825000 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3 = (15+150) \cdot (5000+3500) = 1402500 \text{ (грн)}.$$

Отже, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

5.4 Визначення економічної доцільності фінансування наукової розробки

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає:

1) розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку;

2) розрахунок очікуваного збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки;

3) розрахунок приведеної вартості всіх чистих прибутків ПП;

4) розрахунок відносної (щорічної) ефективності вкладених в наукову розробку інвестицій;

5) розрахунок терміну окупності вкладених у реалізацію наукового проекту інвестицій.

5.4.1 Розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку

5.4.1.1 Розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку. Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, тобто ЗВ = PV = 218615,37(грн).

5.4.2 Розрахунок очікуваного збільшення прибутку

5.4.2.1 Розрахунок очікуваного збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження проведено вище.

5.4.3 Розрахунок приведеної вартості всіх чистих прибутків ПП

5.4.3.1 Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.10)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т — період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні — 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки „0”.

$$ПП = \frac{218615,37}{(1+0,1)^1} + \frac{0}{(1+0,1)^2} + \frac{825000}{(1+0,1)^3} + \frac{1402500}{(1+0,1)^4} = 1776502.32 \text{ (грн).}$$

$$Е_{абс} = 1776502.32 - 218615,37 = 1557886.95 \text{ (грн).}$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

5.4.4 Розрахунок відносної (щорічної) ефективності вкладених в наукову розробку інвестицій

5.4.4.1 Розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$E_в = \sqrt[T_ж]{1+\frac{E_{абс}}{PV}} - 1, \qquad (4.11)$$

де $Е_{абс}$ — абсолютна ефективність вкладених інвестицій, грн;

PV — теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ — життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[4]{1 + \frac{1557886.95}{218615,37}} - 1 = \sqrt[4]{8,126} - 1 = 0,6884 \text{ або } 68,84\%$$

Порівняємо $Е_в$ з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

Спрогнозуємо величину $\tau_{мін.}$ У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f, \qquad (4.12)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; d = 0,14;

f — показник, що характеризує ризикованість вкладень; величина f = 0,3.

$$\tau = 0,14 + 0,3 = 0,44$$

Припустимо, що за даних умов прибуток буде збільшуватись, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

5.4.5 Розрахунок терміну окупності вкладених у реалізацію наукового проекту інвестицій

5.4.5.1 Розраховують термін окупності вкладених у реалізацію наукового проекту інвестицій $Т_{ок}$ за формулою:

$$T_{ок} = \frac{1}{E_в} \text{ [грн].} \qquad (4.13)$$

$$\mathrm{T}_{ok} = \frac{1}{0,6884} = 1.45 \quad \text{(роки).}$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($T_{ок} < 3$ років), то фінансування нової розробки є доцільним.

5.5 Результати економічного аналізу

В даному розділі було здійснено оцінювання комерційного потенціалу розробки технології реалізації конвеєрної збірки динамічних обчислювальних середовищ.

Проведено технологічний аудит з залученням трьох експертів. Аналіз експертних даних показав, що рівень комерційного потенціалу розробки вище середнього. Дослідження комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

Згідно із розрахунками всіх статей витрат на виконання науково–дослідної, дослідно–конструкторської та конструкторсько–технологічної роботи загальна вартість витрат на розробку і впровадження складає 218615,37 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 1557886.95 грн свідчить про отримання прибутку інвестором від впровадження програмного продукту у діяльність підприємства.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 68,84%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 44%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.
Термін окупності складає 1,45 років, що також свідчить про доцільність фінансування.

Усе це, узяте разом, забезпечує прийняття рішення про доцільність виготовлення нового продукту.

ВИСНОВКИ

У даній роботі проведено аналіз основних постачальників хмарних послуг, методів конвеєрної збірки динамічних обчислювальних середовищ та моделей розміщення коду, а також вибір компонентів. На основі чого було обрано хмарне середовище Microsoft Azure та вибір відповідних Pipelines, було обґрунтоване використання безсерверних рішень з використанням Azure Durable Functions.

Вдосконаленно метод масштабування обчислювальних ресурсів за рахунок уведення порогових значень, що дозволило оптимізувати їх використання.

Здійснено практичну реалізацію запропонованого методу у вигляді рішення, що складається з наступних компонентів: конвеєр збірки динамічного середовища, IAC рішення для створення динамічного обчислювального середовища, конвеєр збірки каркасної функції, конвеєр розгортання каркасної функції та каркасна функція.

Проведено верифікацію працездатності отриманого рішення, що підтвердило функціональність, здатність до масштабування, відповідно до кількості вхідних запитів. Також було підтверджено платіжну модель відповідно до часу використання.

Для даного рішення було проведено економічний аналіз, який підтвердив економічну доцільність даного рішення та його конкурентоспроможність.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is Azure—Microsoft Cloud Services | Microsoft Azure. https://azure.microsoft.com/en–us/resources/cloud–computing–dictionary/what–is–azure/ .

2. Microsoft Azure – Wikipedia. https://en.wikipedia.org/wiki/Microsoft_Azure.

3. Microsoft Azure Tutorial – javatpoint. https://www.javatpoint.com/microsoft–azure.

4. What is AWS – aws.amazon.com. https://aws.amazon.com/what–is–aws/.

5. Overview of Amazon Web Services – docs.aws.amazon.com. https://docs.aws.amazon.com/whitepapers/latest/aws–overview/introduction.html.

6. What is AWS Architecture? Components Explained with Diagram – Intellipaat. https://intellipaat.com/blog/what–is–aws–architecture/.

7. Google Cloud Platform Tutorial – GeeksforGeeks. https://bing.com/search?q=Google+GCP+overview.

8. Products and Services | Google Cloud. https://cloud.google.com/products/.

9. Google Cloud overview | Overview. https://cloud.google.com/docs/overview/

10. About | DigitalOcean. https://www.digitalocean.com/about.

11. DigitalOcean Platform Overview :: DigitalOcean Documentation. https://docs.digitalocean.com/products/platform/.

12. DigitalOcean – Investor Relations. https://investors.digitalocean.com/overview/default.aspx.

13. YAML schema reference | Microsoft Learn. https://learn.microsoft.com/en–us/azure/devops/pipelines/yaml–schema/?view=azure–pipelines.

14. Expressions – Azure Pipelines | Microsoft Learn. https://learn.microsoft.com/en–us/azure/devops/pipelines/process/expressions?view=azure–devops.

15. Azure Pipelines | How to Create and Use Pipelines in Azure? – EDUCBA. https://www.educba.com/azure–pipelines/

16. Tips for using the Azure CLI | Microsoft Learn. https://learn.microsoft.com/en–us/cli/azure/use–cli–effectively.

17. Azure CLI samples for Azure Functions | Microsoft Learn. https://learn.microsoft.com/en–us/azure/azure–functions/functions–cli–samples.

18. Deploying Azure Functions with the Azure CLI – Mark Heath. https://markheath.net/post/deploying–azure–functions–with–azure–cli.

19. Web API design best practices – Azure Architecture Center. https://learn.microsoft.com/en–us/azure/architecture/best–practices/api–design.

20. Guidance for developing Azure Functions | Microsoft Learn. https://learn.microsoft.com/en–us/azure/azure–functions/functions–reference.

21. Build Serverless APIs with Azure Functions – Training. https://learn.microsoft.com/en–us/training/modules/build–api–azure–functions/.

22. Інфраструктура як код — Вікіпедія. https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%8F%D0%BA_%D0%BA%D0%BE%D0%B4.

23. Введення в інфраструктуру як код – HashDork. https://hashdork.com/uk/%D0%B7%D0%BD%D0%B0%D0%B9%D0%BE%D0%BC%D1%81%D1%82%D0%B2%D0%BE–%D0%B7–%D1%96%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BE%D1%8E–%D1%8F%D0%BA–%D0%BA%D0%BE%D0%B4%D0%BE%D0%BC/.

24. Інфраструктура як код — Вікіпедія. https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%8F%D0%BA_%D0%BA%D0%BE%D0%B4.

25. Terraform by HashiCorp. https://www.terraform.io/.

26. What is Terraform | Terraform | HashiCorp Developer. https://www.terraform.io/intro

27. Types and Values – Configuration Language | Terraform – HashiCorp Developer. https://developer.hashicorp.com/terraform/language/expressions/types

28. ARM template documentation | Microsoft Learn. https://learn.microsoft.com/en–us/azure/azure–resource–manager/templates/.

29. Azure Resource Manager | Microsoft Learn. https://bing.com/search?q=Azure+ARM+Templates+%d0%be%d1%81%d0%bd%d0%be%d0%b2%d0%bd%d1%96+%d1%84%d1%83%d0%bd%d0%ba%d1%86%d1%96%d1%97.

30. Tutorial – Create and deploy template – Azure Resource Manager. https://learn.microsoft.com/en–us/azure/azure–resource–manager/templates/template–tutorial–create–first–template.

31. Plantillas de Azure Resource Manager | Microsoft Azure. https://azure.microsoft.com/es–es/products/arm–templates/.

32. Bicep language for deploying Azure resources – Azure Resource Manager .... https://learn.microsoft.com/en–us/azure/azure–resource–manager/bicep/overview.

33. 10 BEST Bicep Exercises (Get BIGGER Arms) – YouTube. https://www.youtube.com/watch?v=bXwYwtjbFOk.

34. Bicep documentation | Microsoft Learn. https://learn.microsoft.com/en–us/azure/azure–resource–manager/bicep/.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д..

"29" вересня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

"Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ"

08–57.МКР.004.00.000.ТЗ

Науковий керівник: проф. каф.ОТ

_____ Захарченко С.М.

Студент групи 1КІ–22м

_____ Гуменюк О.В.

ВНТУ 2023

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Важливим є актуальність дослідження у напрямку магістерської роботи, яка обумовлена тим, що сьогодні, для продовження роботи над науковими відкриттями та розробками, необхідно аналізувати величезні масиви даних. Під час даного процесу виникає дві проблеми: час, необхідний для виконання аналізу та бюджет наукового дослідження. Для того, щоб пришвидшити процес аналізу необхідно мати дуже потужні обчислювальні середовища, які мають велику вартість. І навпаки, для того щоб зменшити вартість аналізу отриманих результатів, можливо проводити аналіз з використанням менших потужностей. Відповідно, необхідно знайти баланс між цими двома характеристиками.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — аналіз і розробка покращення методів конвеєрної збірки обчислювальних середовищ, за рахунок динамічного додавання або згортання обчислювальних ресурсів залежно від навантаження.

2.2 Призначення розробки — система управління динамічними обчислювальними ресурсами призначення для оптимізації проведення обчислень, за рахунок динамічного додавання або зменшення кількості обчислювальних ресурсів в залежності від потреби програмного забезпечення

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу існуючих методів та принципів конвеєрної збірки динамічних обчислювальних середовищ;

3.2 Розробка та дослідження методів масштабування ресурсів в динамічних обчислювальних середовищах

3.3 Розробка програмного рішення технології конвеєрної збірки динамічних обчислювальних середовищ;

3.4 Проведення верифікації та аналізу отриманих результатів

3.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору;

4 Вимоги до виконання МКР

Головна вимога — використовувати горизонтальне і вертикальне масштабування із використанням порогових значень

5 Етапи МКР та очікувані результати

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

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз існуючих методів конвеєрної збірки динамічних обчислювальних середовищ	18.09.2023	25.09.2023	Аналітичний огляд джерел, задачі досліджень, розділ 1
2	Дослідження методів побудови динамічних обчислювальних середовищ	28.09.2023	5.10.2023	Розділ 2
3	Розробка технології конвеєрної збірки динамічних середовищ в хмарному середовищі Microsoft Azure	5.10.2023	12.10.2023	Розділ 3

Продовження таблиці А.1

4	Підготовка економічної частини	12.10.2023	19.10.2023	Розділ 4
5	Апробація та впровадження результатів дослідження	18.10.2023	26.10.2023	Тези доповідей
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	27.10.2023	2.11.2023	ПЗ, графічний матеріал і презентація
7	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	2.11.2023	10.11.2023	Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання 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»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

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

Витрати на обслуговування динамічного обчислювального середовища

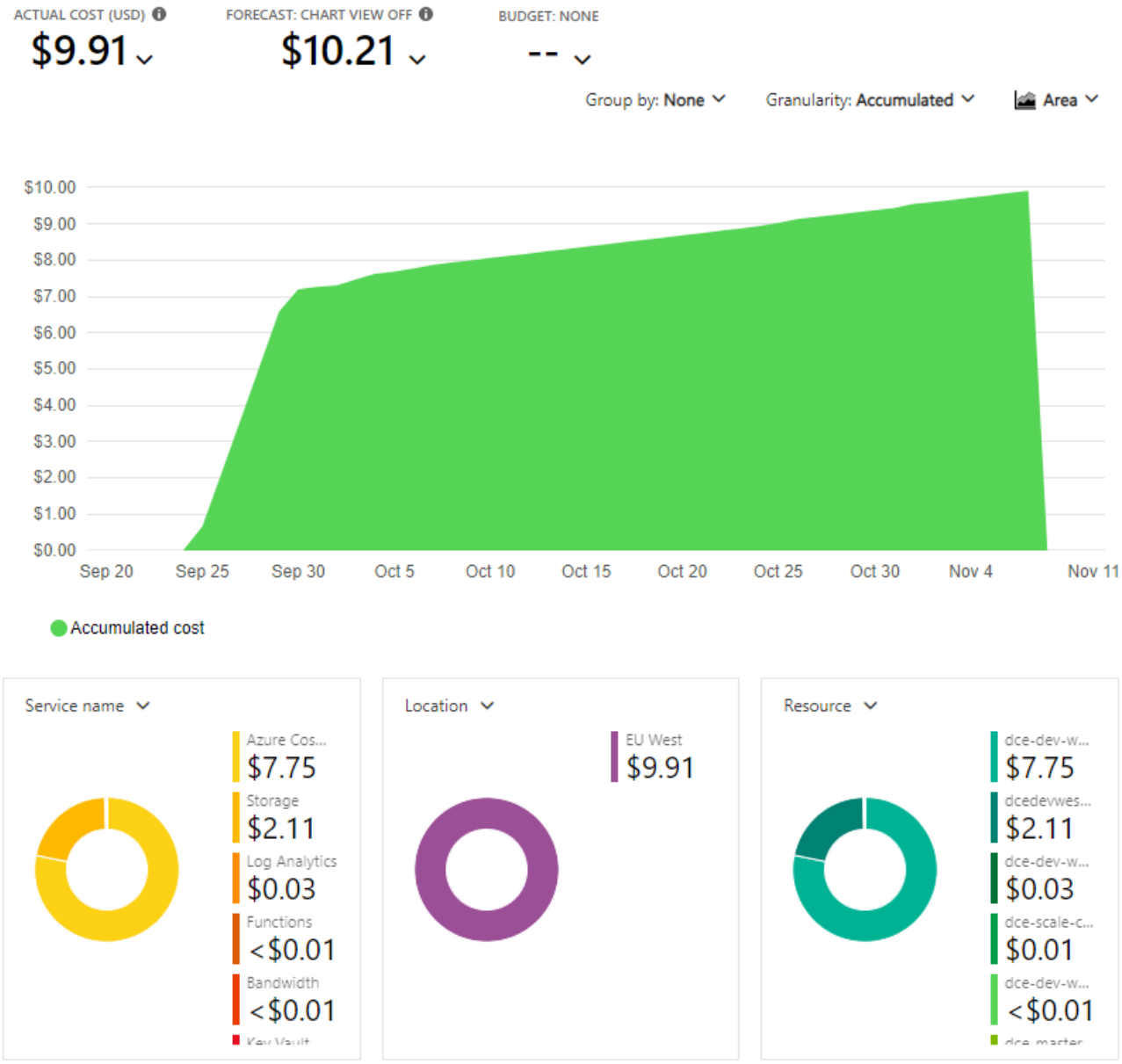


Рисунок Б.1 — Витрати на обслуговування динамічного обчислювального середовища

ДОДАТОК В

Лістинг програми

1. Main.bicep

```
param productName string = 'dce'
param masterKeyVaultSub string = '55711f69–20a9–4e30–b2c1–
43e819a0f3b6'
param environmentType string
param region string
param location string
param masterKeyVaultName string
param masterKeyVaultRg string

var tags = {
  productName: productName
  envType: environmentType
  costCenter: productName
}

// Key Vault
module keyVault 'modules/key–vault/key–vault.bicep' = {
  name: '${productName}–key–vault'
  params: {
    location: location
    name: '${productName}–${environmentType}–${region}–kv'
    tags: tags
  }
}

resource masterKeyVault 'Microsoft.KeyVault/vaults@2019–09–01' existing =
{
  name: masterKeyVaultName
  scope: resourceGroup(masterKeyVaultSub, masterKeyVaultRg)
}

module keyVaultSecrets 'modules/key–vault/key–vault–secrets.bicep' = {
  name: '${productName}–key–vault–secrets'
  params: {
    keyVaultName: keyVault.outputs.name
    someSecret: masterKeyVault.getSecret('SomeSecret–${environmentType}')
    cosmosDbConnectionString: cosmosDBAccount.outputs.connectionString
```

```
  }
  dependsOn: [keyVault, cosmosDBAccount]
}

// Application Insights
module appInsights 'modules/app–insights.bicep' = {
  name: '${productName}–app–insights'
  params: {
    name: '${productName}–${environmentType}–${region}–ai'
    location: location
    workspaceName: '${productName}–${environmentType}–${region}–lga'
    tags: tags
  }
}

// Storage account
module storageAccount 'modules/storage–account.bicep' = {
  name: '${productName}–storage–account'
  params: {
    name: '${productName}${environmentType}${region}sa'
    location: location
    tags: tags
    publicNetworkAccess: 'Enabled'
  }
}

// Cosmos DB Account
module cosmosDBAccount 'modules/cosmos–db/cosmos–db–account.bicep' =
{
  name: '${productName}–cosmos–db–account'
  params: {
    name: '${productName}–${environmentType}–${region}–cosmosdb'
    location: location
    tags: tags
  }
}

// Cosmos Databases
module cosmosDatabase 'modules/cosmos–db/cosmos–database.bicep' = {
  name: '${productName}–cosmos–database'
  params: {
    accountName: cosmosDBAccount.outputs.name
```

```
    databaseName: toUpper(productName)
    location: location
    tags: tags
  }
  dependsOn: [cosmosDBAccount]
}

// Cosmos Containers
module cosmosDatabaseContainers 'modules/cosmos–db/cosmos–database–
containers.bicep' = {
  name: '${productName}–cosmos–database–containers'
  params: {
    accountName: cosmosDBAccount.outputs.name
    databaseName: cosmosDatabase.outputs.name
    collectionConfig: [
      {
        name: 'UserActivities'
        partitionKey: '/user/userId'
        indexIncludedPaths: [
          {
            path: '/*'
          }
        ]
        indexExcludedPaths: [

        ]
      }
    ]
  }
  dependsOn: [cosmosDBAccount, cosmosDatabase]
}

// App Service plans
module dynamicServicePlan 'modules/app–service–plan.bicep' = {
  name: '${productName}–dynamic–service–plan'
  params: {
    name: '${productName}–${environmentType}–${region}–dyn–asp'
    location: location
    skuName: 'Y1'
    tier: 'Dynamic'
    tags: tags
  }
```

```
}

var functionAppSettings = [
  {
    name: 'FUNCTIONS_WORKER_RUNTIME'
    value: 'dotnet–isolated'
  }
  {
    name: 'FUNCTIONS_EXTENSION_VERSION'
    value: '~4'
  }
  {
    name: 'APPINSIGHTS_INSTRUMENTATIONKEY'
    value: appInsights.outputs.instrumentationKey
  }
  {
    name: 'WEBSITE_RUN_FROM_PACKAGE'
    value: 1
  }
  {
    name: 'AzureWebJobsStorage'
    value: storageAccount.outputs.connectionString
  }
]

// Function App
module generatorFunction 'modules/dynamic–function–app.bicep' = {
  name: '${productName}–scale–controller–af'
  params: {
    name: '${productName}–scale–controller–${environmentType}–${region}–
af'
    location: location
    tags: tags
    environmentType: environmentType
    appServicePlanId: dynamicServicePlan.outputs.id
    storageAccountName: storageAccount.outputs.name
    keyVaultName: keyVault.outputs.name
    appSettings: concat(functionAppSettings, [
      {
        name: 'WEBSITE_CONTENTSHARE'
        value: 'dce–scale–controller'
      }
```

```
        {
          name: 'WEBSITE_CONTENTAZUREFILECONNECTIONSTRING'
          value: storageAccount.outputs.connectionString
        }
        {
          name: 'StorageURL'
          value: storageAccount.outputs.blobUrl
        }
        {
          name: 'CosmosDbProcessMaxItems'
          value: 1000
        }
        {
          name: 'CosmosDB:ConnectionString'
          value:
'@Microsoft.KeyVault(VaultName=${keyVault.outputs.name};SecretName=C
osmosDBConnectionString)'
        }
        {
          name: 'CosmosDB:Database:Name'
          value: 'DCE'
        }
        {
          name: 'CosmosDB:Database:Collections:0'
          value: 'UserActivities'
        }
        {
          name: 'SchedulePasTimerTrigger'
          value: '0 0 0 1 * *'
        }
        {
          name: 'AzureAD:TenantId'
          value: subscription().tenantId
        }
      ])
    }
    dependsOn: [storageAccount, appInsights]
  }
```

2. Iac–template.yml

```
parameters:
```

```
  envType: ''

jobs:
– deployment: Deploy
  displayName: 'Deploy IaC'
  condition: succeeded()
  timeoutInMinutes: 360
  environment: '${{ parameters.envType }}'
  variables:
    – name: BICEP_FILE
      value: '$(System.DefaultWorkingDirectory)/main.bicep'
    – template: '../variables/${{ parameters.envType }}–variables.yml'
  pool:
    vmImage: windows–latest
  strategy:
    runOnce:
      deploy:
        steps:
          – checkout: self


          – task: PowerShell@2
            displayName: Lint Bicep code
            inputs:
              targetType: 'inline'
              script: 'az bicep build —file ${{ variables.BICEP_FILE }}'
```

```
    – task: AzureCLI@2
      displayName: Validation of Azure DCE Infrastructure
      inputs:
        azureSubscription: '${{ variables.AZURE_SRVC_CONNECTION }}'
        scriptType: ps
        scriptLocation: inlineScript
        inlineScript: >
          az deployment group validate
          —resource–group ${{ variables.RESOURCE_GROUP }}
          —template–file ${{ variables.BICEP_FILE }}
          —parameters  environmentType=${{ parameters.envType }} `
                      region=${{ variables.REGION }} `
                      location=${{ variables.LOCATION }} `
                      masterKeyVaultName=${{
variables.MASTER_KEY_VAULT_NAME }} `
                      masterKeyVaultRg=${{ variables.MASTER_KEY_VAULT_RG
}}

    – task: AzureCLI@2
      displayName: Preview changes
      inputs:
        azureSubscription: '${{ variables.AZURE_SRVC_CONNECTION }}'
        scriptType: ps
        scriptLocation: inlineScript
        inlineScript:  >
          az deployment group what–if
```

```
            —resource–group ${{ variables.RESOURCE_GROUP }}
            —template–file ${{ variables.BICEP_FILE }}
            —parameters  environmentType=${{ parameters.envType }} `
                        region=${{ variables.REGION }} `
                        location=${{ variables.LOCATION }} `
                        masterKeyVaultName=${{
variables.MASTER_KEY_VAULT_NAME }} `
                        masterKeyVaultRg=${{ variables.MASTER_KEY_VAULT_RG
}}


        – task: AzureCLI@2
          displayName: Deploy
          condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
          inputs:
            azureSubscription: '${{ variables.AZURE_SRVC_CONNECTION }}'
            scriptType: ps
            scriptLocation: inlineScript
            inlineScript:  >
              az deployment group create
            —resource–group ${{ variables.RESOURCE_GROUP }}
            —template–file ${{ variables.BICEP_FILE }}
            —parameters  environmentType=${{ parameters.envType }} `
                        region=${{ variables.REGION }} `
                        location=${{ variables.LOCATION }} `
                        masterKeyVaultName=${{
variables.MASTER_KEY_VAULT_NAME }} `
```

```
                masterKeyVaultRg=${{ variables.MASTER_KEY_VAULT_RG
}}
```

azure–bicep–pipelines.yml

```
name: $(Build.SourceBranchName)$(Rev:.r)

stages:
  – stage: 'Dev'
    displayName: 'Deploy to dev'
    jobs:
      – template: 'pipelines/templates/iac–template.yml'
        parameters:
          envType: 'dev'

  – stage: 'Tst'
    displayName: 'Deploy to tst'
    condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
    dependsOn: 'Dev'
    jobs:
      – template: 'pipelines/templates/iac–template.yml'
        parameters:
          envType: 'tst'

  – stage: 'Int'
    displayName: 'Deploy to int'
    condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
    dependsOn: 'Tst'
    jobs:
      – template: 'pipelines/templates/iac–template.yml'
```

```
      parameters:
        envType: 'int'

  – stage: 'Pre'
    displayName: 'Deploy to pre'
    condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
    dependsOn: 'Int'
    jobs:
      – template: 'pipelines/templates/iac–template.yml'
        parameters:
          envType: 'pre'

  – stage: 'Prd'
    displayName: 'Deploy to prd'
    condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
    dependsOn: 'Pre'
    jobs:
      – template: 'pipelines/templates/iac–template.yml'
        parameters:
          envType: 'prd'
```

3. DocumentDB Repository

```
using Infrastructure.ContinuationToken;
using Infrastructure.Models;
using Microsoft.Azure.Cosmos;
using Microsoft.Azure.Cosmos.Linq;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using System.Linq.Expressions;
```

```
namespace Infrastructure.Repositories
{
    public abstract class DocumentDbRepository<T> :
IDocumentDbRepository<T> where T : DbEntity
    {
        private object _locker = new object();

        private readonly IConfiguration _configuration;
        private readonly string _databaseId;
        private readonly string _collectionId;
        private readonly string _connectionString;
        private readonly ILogger _logger;
        private readonly Container _container;

        public CosmosClient CosmosDBClient { get; }

        private Container _batchContainer;

        private Container BatchContainer
        {
            get
            {
                lock (_locker)
                {
                    if (_batchContainer == null)
                    {
                        var options = new CosmosClientOptions() {
AllowBulkExecution = true, EnableContentResponseOnWrite = false,
```

```
MaxRetryAttemptsOnRateLimitedRequests = 30,
MaxRetryWaitTimeOnRateLimitedRequests = new TimeSpan(0, 0, 5) };

                                        var client = new
CosmosClient(_connectionString, options);

                                        _batchContainer =
client.GetDatabase(_databaseId)?.GetContainer(_collectionId) ?? throw new
ArgumentNullException("Database");

                                    }

                                    return _batchContainer;

                                }


                            }

                        }


                        protected DocumentDbRepository(IServiceProvider serviceProvider,
string databaseId, string collectionId)

                        {

                            _logger =
serviceProvider.GetRequiredService<ILogger<DocumentDbRepository<T>>>();

                            _configuration =
serviceProvider.GetRequiredService<IConfiguration>();

                            _databaseId = databaseId;

                            _collectionId = collectionId;


                            _connectionString =
_configuration["CosmosDB:ConnectionString"] ?? throw new
ArgumentNullException("ConnectionString");


                            CosmosDBClient = new CosmosClient(_connectionString, new
CosmosClientOptions()

                            {

                                MaxRetryAttemptsOnRateLimitedRequests = 30,
```

```
                MaxRetryWaitTimeOnRateLimitedRequests =
TimeSpan.FromSeconds(10)
            });


            _container =
CosmosDBClient.GetDatabase(databaseId)?.GetContainer(_collectionId) ?? throw
new ArgumentNullException("Database");
        }


        public async Task<T> GetItemAsync(Expression<Func<T, bool>>
predicate, string partitionKey = null)
        {
            predicate ??= t => true;
            T result = null;
            var item = _container.GetItemLinqQueryable<T>(requestOptions:
new QueryRequestOptions()
            {
                MaxItemCount = 1,
                PartitionKey = partitionKey == null ? null : new
PartitionKey(partitionKey)
            }).Where(predicate);


            FeedIterator<T> resultSetIterator = item.ToFeedIterator();


            if (resultSetIterator.HasMoreResults)
            {
                var response = await resultSetIterator.ReadNextAsync();


                _logger.LogDebug("GetItemAsync has latency {latency}
ms, where predicate:{predicate}, R/U:{requestCharge}",
response.Diagnostics.GetClientElapsedTime().TotalMilliseconds,
predicate.Body.ToString(), response.RequestCharge);
```

```
                result = response.FirstOrDefault();
            }
            return result;
        }

        public async Task<T> GetItemAsync(string id, string partitionKey =
null)
        {
            try
            {
                var key = partitionKey == null ? PartitionKey.None : new
PartitionKey(partitionKey);

                ItemResponse<T> document = await
_container.ReadItemAsync<T>(id, key);

                _logger.LogDebug("GetItemAsync has latency {latency}
ms, where id:{id}, R/U:{requestCharge}",
document.Diagnostics.GetClientElapsedTime().TotalMilliseconds, id,
document.RequestCharge);

                return (T)(dynamic)document.Resource;
            }
            catch (CosmosException e)
            {
                if (e.StatusCode == System.Net.HttpStatusCode.NotFound)
                {
                    return null;
                }
                else
                {
```

```
                throw;
            }
        }
    }

    public async Task<IEnumerable<T>>
GetItemsAsync(Expression<Func<T, bool>> predicate = null, string partitionKey =
null)
    {
        predicate ??= t => true;

        PartitionKey? key = partitionKey == null ? null : new
PartitionKey(partitionKey);

        QueryRequestOptions options = new() { PartitionKey = key };
        var res = _container.GetItemLinqQueryable<T>(requestOptions:
options).Where(predicate);

        FeedIterator<T> resultSetIterator = res.ToFeedIterator();
        List<T> results = new List<T>();

        while (resultSetIterator.HasMoreResults)
        {
            var response = await resultSetIterator.ReadNextAsync();

            results.AddRange(response);

            _logger.LogDebug("GetItemsAsync has latency {latency}
ms, where predicate:{predicate}, R/U:{requestCharge}",
response.Diagnostics.GetClientElapsedTime().TotalMilliseconds,
predicate.Body.ToString(), response.RequestCharge);
```

```
            }

            return results;
        }

        public async Task<(List<string>, IEnumerable<T>)>
GetItemsByPageAsync(int? pageSize, string continuationToken,
Expression<Func<T, bool>> predicate = null)
        {
            predicate ??= t => true;

            QueryRequestOptions options = new() { MaxItemCount =
pageSize ?? –1, MaxConcurrency = –1 };
            string token =
CompositeContinuationToken.BuildTokenString(continuationToken);

            var query = _container
                .GetItemLinqQueryable<T>(continuationToken: token,
requestOptions: options)
                .Where(predicate);

            FeedIterator<T> resultSetIterator = query.ToFeedIterator();
            List<T> results = new();
            List<string> continuationTokens = null;

            if (resultSetIterator.HasMoreResults)
            {
                var feedResponse = await
resultSetIterator.ReadNextAsync();
                var totalCount = string.IsNullOrEmpty(continuationToken)
? await query.CountAsync() : 0;
```

```
                    _logger.LogDebug("GetItemsByPageAsync has latency
{latency} ms, where pageSize:{pageSize}, continuationToken:{continuationToken},
predicate:{predicate}, R/U:{requestCharge}",
feedResponse.Diagnostics.GetClientElapsedTime().TotalMilliseconds, pageSize,
continuationToken, predicate.Body.ToString(), feedResponse.RequestCharge);


                    results.AddRange(feedResponse);


                    if (string.IsNullOrEmpty(continuationToken) &&

    !string.IsNullOrEmpty(feedResponse.ContinuationToken) &&
                        totalCount != pageSize)
                    {
                        continuationTokens = new List<string>
                        {
                            null,

    CompositeContinuationToken.ParseToken(feedResponse.ContinuationToken)
                        };
                        continuationTokens.AddRange(await
GetTokensAsync(resultSetIterator));
                    }
                }


                return (continuationTokens, results);
            }


            private async Task<List<string>> GetTokensAsync(FeedIterator<T>
resultSetIterator)
            {
                var maxPages =
_configuration.GetValue<int?>("Audit:MaxPages", null);
```

```
                List<string> results = new();
                while (resultSetIterator.HasMoreResults &&
(!maxPages.HasValue || maxPages > results.Count + 2))
                {
                        var feedResponse = await
resultSetIterator.ReadNextAsync();
                        _logger.LogDebug("GetTokensAsync has latency {latency}
ms, R/U:{requestCharge}",
feedResponse.Diagnostics.GetClientElapsedTime().TotalMilliseconds,
feedResponse.RequestCharge);

                        if
(!string.IsNullOrEmpty(feedResponse.ContinuationToken))
                        {

        results.Add(CompositeContinuationToken.ParseToken(feedResponse.Continua
tionToken));
                        }
                }
                return results;
        }

        public async Task<T> CreateItemAsync(T item, string partitionKey =
null)
        {
                var key = partitionKey == null ? PartitionKey.None : new
PartitionKey(partitionKey);

                item.Id = string.IsNullOrEmpty(item.Id) ?
Guid.NewGuid().ToString() : item.Id;
```

```
                ItemResponse<T> doc = await _container.CreateItemAsync(item,
key);

                _logger.LogDebug("CreateItemAsync has latency {latency} ms,
where id:{id}, R/U:{requestCharge}",
doc.Diagnostics.GetClientElapsedTime().TotalMilliseconds, item.Id,
doc.RequestCharge);

                return (T)(dynamic)doc.Resource;
            }
            public async Task<List<T>> CreateItemsAsyncBatch(List<T> items,
Expression<Func<T, string>> partitionKeySelector = null)
            {
                var partitionKeyFunc = partitionKeySelector?.Compile();

                var tasks = items.Select(t =>
                {
                    var partitionKey = partitionKeyFunc == null ?
PartitionKey.None : new PartitionKey(partitionKeyFunc.Invoke(t));

                    t.Id = string.IsNullOrEmpty(t.Id) ?
Guid.NewGuid().ToString() : t.Id;
                    return BatchContainer.CreateItemAsync(t, partitionKey);
                });

                await Task.WhenAll(tasks);

                return items;
            }
            public async Task<List<T>> UpdateItemsAsyncBatch(List<T> items,
Expression<Func<T, string>> partitionKeySelector = null)
```

```
        {
            var partitionKeyFunc = partitionKeySelector?.Compile();

            var tasks = items.Select(t =>
            {
                try
                {
                    var partitionKey = partitionKeyFunc == null ? PartitionKey.None : new PartitionKey(partitionKeyFunc.Invoke(t));
                    return BatchContainer.ReplaceItemAsync(t, t.Id, partitionKey);
                }
                catch (Exception)
                {
                    return null;
                }
            });
            await Task.WhenAll(tasks);
            return items;
        }

        public async Task<T> UpdateItemAsync(T item, string partitionKey = null)
        {
            try
            {
                var key = partitionKey == null ? PartitionKey.None : new PartitionKey(partitionKey);
```

```
                ItemResponse<T> doc = await
_container.ReplaceItemAsync(item, item.Id, key);


                _logger.LogDebug("UpdateItemAsync has latency
{latency} ms, where id:{id}, R/U:{requestCharge}",
doc.Diagnostics.GetClientElapsedTime().TotalMilliseconds, item.Id,
doc.RequestCharge);
                return (T)(dynamic)doc.Resource;
            }
            catch (Exception)
            {
                return null;
            }
        }

        public async Task<T> DeleteItemAsync(string id, string partitionKey =
null)
        {
            var key = partitionKey == null ? PartitionKey.None : new
PartitionKey(partitionKey);


            var doc = await _container.DeleteItemAsync<T>(id, key);


            _logger.LogDebug("DeleteItemAsync has latency {latency} ms,
where id:{id}, R/U:{requestCharge}",
doc.Diagnostics.GetClientElapsedTime().TotalMilliseconds, id, doc.RequestCharge);
            return (T)(dynamic)doc.Resource;
        }

        public async Task<T> UpsertItemAsync(T item, string partitionKey =
null)
        {
```

```
                var key = partitionKey == null ? PartitionKey.None : new
PartitionKey(partitionKey);

                var doc = await _container.UpsertItemAsync(item, key);

                _logger.LogDebug("UpsertItemAsync has latency {latency} ms,
where id:{id}, R/U:{requestCharge}",
doc.Diagnostics.GetClientElapsedTime().TotalMilliseconds, item.Id,
doc.RequestCharge);

                return (T)(dynamic)doc.Resource;
            }
        }
}
```

ДОДАТОК Г

Алгоритм масштабування ресурсів процесора

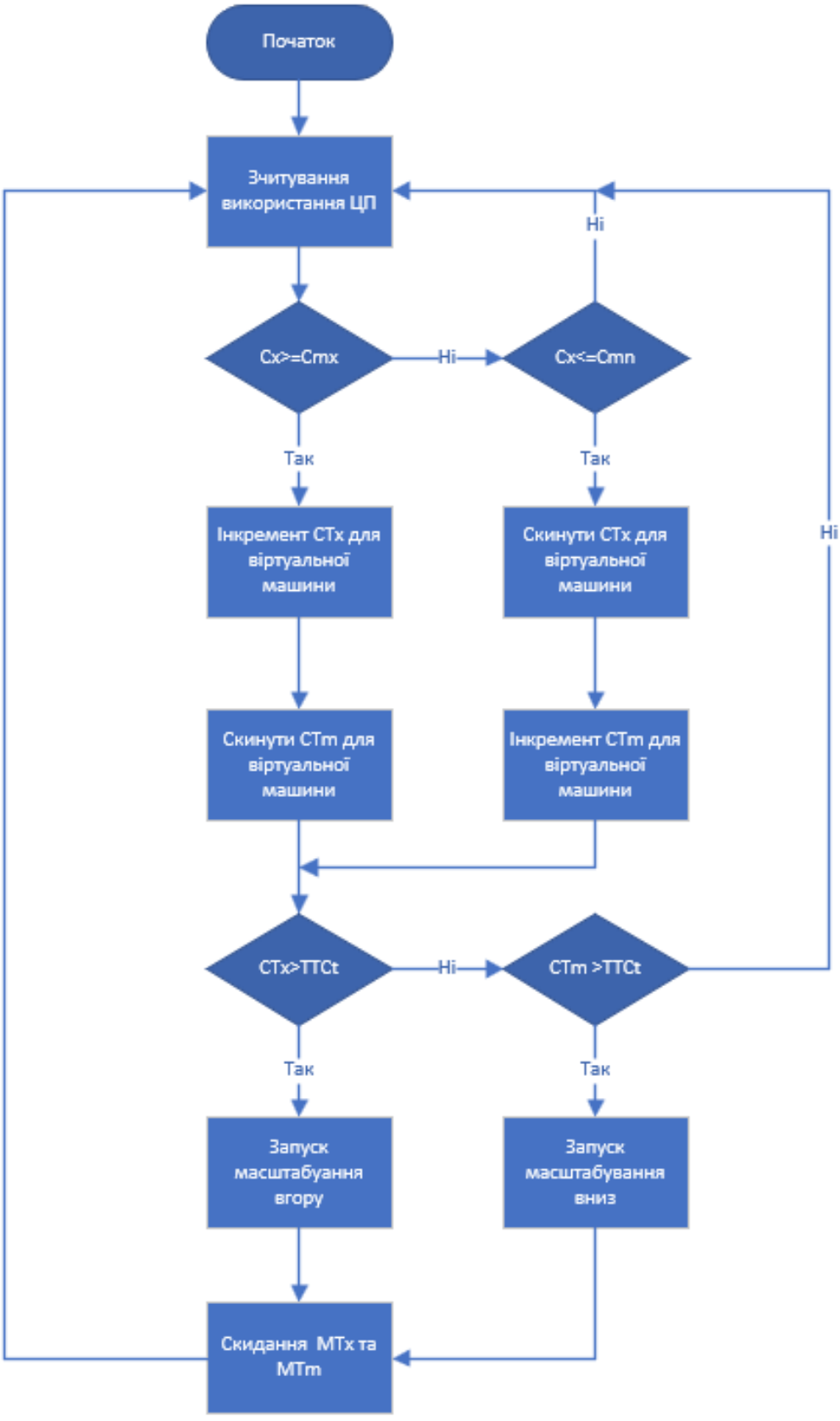


Рисунок Г.1 — Алгоритм масштабування ресурсів процесора

ДОДАТОК Д

Алгоритм масштабування ресурсів оперативної пам’яті

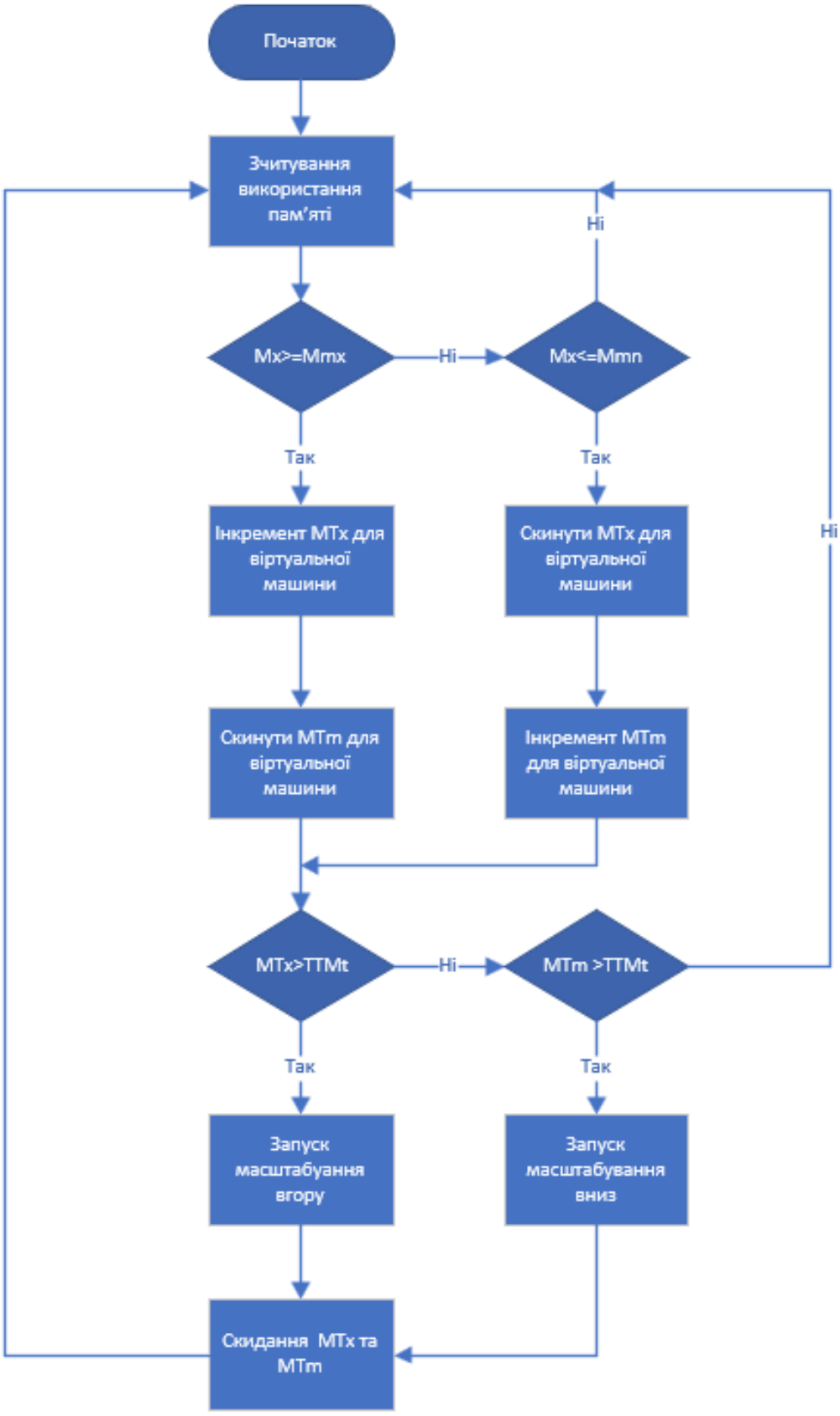


Рисунок Д.1 — Алгоритм масштабування ресурсів оперативної пам’яті

ДОДАТОК Е

Алгоритм вертикального масштабування на основі порогових значень

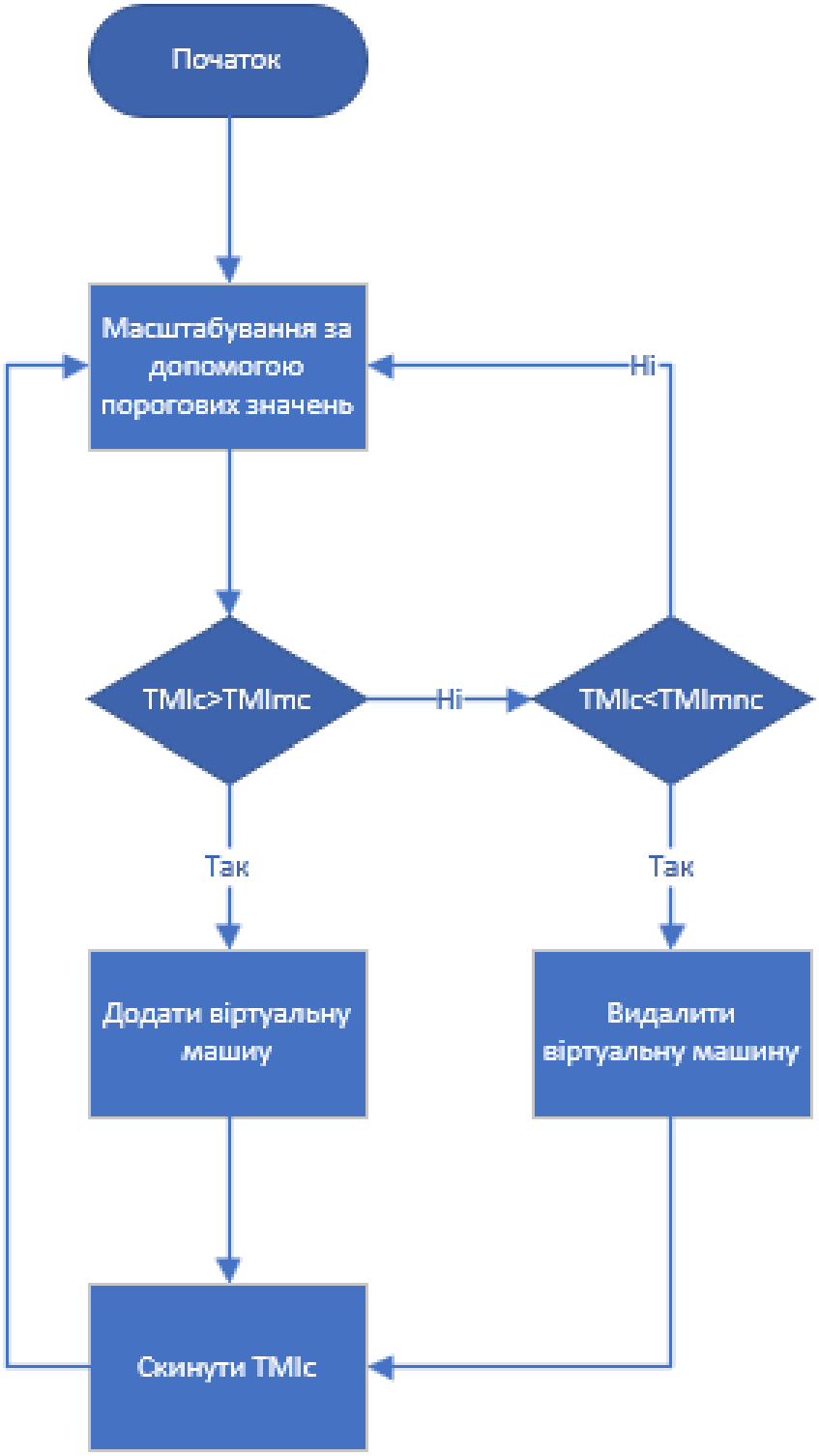


Рисунок Е.1 — Алгоритм вертикального масштабування на основі порогових значень

ДОДАТОК Ж

Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ.**

Тип роботи: кваліфікаційна робота
Підрозділ: кафедра обчислювальної техніки, ФІТКІ, 1–КІ–22м
Науковий керівник: проф. Захарченко С.М.

Unicheck	
Оригінальність	**96.8%**
Схожість	3.2%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Технології реалізації конвеєрної збірки динамічних обчислювальних середовищ».

Автор _____ Гуменюк О.В.

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку Захарченко С.М.

_____ Захарченко С.М.

(підпис) (прізвище, ініціали)

Експерт _____ Захарченко С.М.

(за потреби) (підпис) (прізвище, ініціали, посада)