Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure»

Виконав: студент 2-го курсу, групи 2КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Доманський Б.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. КН
Барабан С.В.
(прізвище та ініціали)
«07» 12 2023 р.

Опонент: к.т.н., доц. каф. САІТ
Варчук І.В.
(прізвище та ініціали)
«09» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«08. 12» 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри КН
д.т.н., проф. Яровий А.А.

(підпис)
"29" 08 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Доманському Богдану В'ячеславовичу

1. Тема роботи: «Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure».
Керівник роботи: Барабан Сергій Володимирович, к.т.н., доцент.
затверджені наказом вищого навчального закладу «18» 09 2023 року № 247
2. Строк подання студентом роботи 13.11.2023
3. Вихідні дані до роботи: платформа програмного засобу – платформа .Net; конфігурації необхідні для розгортання – не менше 5шт; параметри для налаштувань конфігурації – не менше 5 шт; Мова програмування – об'єктно-орієнтована.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз технологій розгортання інфраструктури додатків на платформі .Net; розробка інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure; програмна реалізація інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure; економічна частина, висновки, список використаних джерел, додатки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму розгортання інфраструктури на основі хмарного провайдера Azure; структура інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure; приклад роботи

програмного засобу розгортання інфраструктури на основі хмарного провайдера Azure

6. Консультанти розділів проекту (роботи)

Консультанти розділів роботи в таблиці 1.

Таблиця 1 - Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Барабан С.В., к.т.н., доцент каф. КН	29.09.23	10.11.23
4	Ратушняк О. Г., к.т.н, доц. каф. ЕПВМ	5.11.23	20.11.23

7. Дата видачі завдання 29.08.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз технологій розгортання інфраструктури додатків на платформі .Net Постановка задач дослідження	07.09.23-10.10.23	
2	Розробка інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure	24.09.23-20.10.23	
3	Програмна реалізація інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure	20.10.23-7.11.23	
4	Підготовка економічної частини	05.11.23-20.11.23	
5	Оформлення пояснювальної записки, графічного матеріалу та презентації	25.11.23-10.11.23	

Студент _____ (підпис) Доманський Б.В. (прізвище та ініціали)

Керівник роботи _____ (підпис) Барабан С.В. (прізвище та ініціали)

АНОТАЦІЯ

УДК 621.374.411

Доманський Б.В. Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure. Магістерська кваліфікаційна робота зі спеціальності 122 – комп'ютерні науки, освітня програма - Системи штучного інтелекту. Вінниця: ВНТУ, 2023. 105с.

На укр. мові. Бібліогр.: 20 назв; рис.: 12; табл. 14.

Дана магістерська кваліфікаційна робота присвячена розробці інформаційної технології по розгортанню інфраструктури на основі хмарного провайдера Azure.

В ході роботи проведено аналіз предметної області розгортання інфраструктури додатків на платформі .Net. Обґрунтовано вибір технологій, проведено проектування архітектури та розроблено та реалізовано інформаційну технологію по розгортанню інфраструктури на основі хмарного провайдера Azure у вигляі програмного забезпечення за допомогою рекурсивного акроніму yaml та комплексного рішення Azure DevOps resource manager templates.

В розділі економічної частини проведено оцінювання комерційного потенціалу розробки інформаційної технології по розгортанню інфраструктури на основі хмарного провайдера Azure, спрогнозовано витрати на виконання наукової роботи та впровадження результатів, які склали 166414,47 грн, розраховано період окупності – 0,4 року.

Ключові слова: інформаційна технологія, Azure, хмарні технології, платформа .Net.

ABSTRACT

Domanskyi B.V. Information technology with infrastructure deployment based on the Azure cloud provider. Master's thesis on specialty 122 - computer science, educational program - Artificial intelligence systems. Vinnytsia: VNTU, 2023. 105p.

In Ukrainian speech Bibliography: 20 titles; Fig.: 11; table 14.

This master's thesis is devoted to the development of information technology for the deployment of infrastructure based on the Azure cloud provider.

In the course of the work, an analysis of the subject area of application infrastructure deployment on the .Net platform was carried out. The choice of technologies was substantiated, architecture was designed, and information technology was developed and implemented for deploying infrastructure based on the Azure cloud provider in the form of software using the recursive acronym yaml and the comprehensive solution Azure DevOps resource manager templates.

In the section of the economic part, an assessment of the commercial potential of the development of information technology for the deployment of infrastructure based on the Azure cloud provider was carried out, the costs of performing scientific work and implementing the results were forecast, which amounted to UAH 166,414.47, and the payback period was calculated - 0.4 years.

Keywords: information technology, Azure, cloud technologies, .Net platform.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Управління процесами розгортання та подальшої експлуатації інфраструктури є важливим завданням для будь-якої IT-компанії. Цей процес є ключовим елементом для успішної організації як всіх внутрішніх процесів компанії, так і для реалізації її майбутніх стратегій.

З роками зростає кількість компаній, які стикаються із завданням побудови власної, автономної інфраструктури. Правильне виконання цього завдання забезпечить ефективну роботу як для штатних працівників, так і для всіх, хто залучений до підтримки продуктів компанії протягом їх життєвого циклу. Іноді ця необхідність стає очевидною лише після того, як виникають проблеми. Після неприємного інциденту стає зрозумілим, що кожен елемент інфраструктури, навіть якщо його роль вважалася не настільки критичною, може вплинути на робочі процеси кожного працівника.

IT-інфраструктура - це сукупність сервісів та систем компанії, включаючи програмне забезпечення, обчислювальні програми та інші складові, необхідні для вирішення бізнес-завдань підприємства. Іншими словами, це включає пристрої з доступом в Інтернет, бази даних, програмне забезпечення, корпоративну пошту та інше. Усі ці компоненти взаємодіють через Інтернет, створюючи ефективне середовище для роботи всієї компанії та кожного її підрозділу. Надійна робота IT-інфраструктури забезпечує зв'язок між різними відділами, передачу файлів та інформації, а також правильну функціональність всіх сервісів компанії.

Тому розробка інформаційної технології, яка вирішує завдання розгортання інфраструктури на базі хмарного провайдера Azure, є надзвичайно доцільною

Зв'язок роботи з науковими програмами, планами, темами. Магістерська робота виконана відповідно до напрямку наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою дослідження магістерської кваліфікаційної роботи є підвищення якості процесу розгортання інфраструктури додатків за рахунок використання хмарного середовища Azure.

Для досягнення поставленої мети слід розв'язати такі завдання:

- проаналізувати існуючі програмні реалізації розв'язання задачі розгортання інфраструктури додатків на основі хмарного провайдера Azure;
- розробити структуру інформаційної технології розгортання інфраструктури додатків на основі хмарного провайдера Azure;
- навести стадії інформаційної технології та на їх основі розробити алгоритм роботи програмного засобу;
- виконати програмну реалізацію запропонованої інформаційної технології розгортання інфраструктури додатків на основі хмарного провайдера Azure;
- провести тестування програмного продукту та виконати аналіз отриманих результатів;
- здійснити економічні розрахунки доцільності розробки нової інформаційної технології.

Об'єкт дослідження – процес розгортання інфраструктури додатків на основі хмарного провайдера Azure.

Предмет дослідження – інформаційна технологія розгортання інфраструктури додатків на основі хмарного провайдера Azure.

Методи дослідження. У роботі використані такі методи наукових досліджень: методи та моделі подання знань, методи математичного моделювання, методи автоматизованого тестування програмних засобів, методи та техніки тест-дизайну, методи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів полягає в наступному:

Набула подальшого розвитку інформаційна технологія розгортання інфраструктури додатків на платформі .Net, яка відрізняється від існуючих використанням інформаційної моделі хмарного середовища, що дозволяє підвищити якість процесу розгортання інфраструктури.

Практичне значення одержаних результатів:

1. Розроблено програмний засіб для розгортання інфраструктури додатків на платформі .Net з використанням хмарного провайдера Azure.

Особистий внесок магістранта. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно. У опублікованій роботі автору належить результат: розгортання інфраструктури додатків .NET на основі хмарного провайдера Azure [1].

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Апробація результатів роботи. Результати роботи були апробовані на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (м. Вінниця, Україна, 2023 р.) [1].

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано тезу доповіді на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» [1].

1 АНАЛІЗ ТЕХНОЛОГІЙ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ДОДАТКІВ НА ПЛАТФОРМІ .NET

1.1 Особливості розгортання програмної інфраструктури

За оцінками компанії IBM біля 70% IT-бюджету витрачається на підтримку існуючої інфраструктури і тільки 30% на її розширення. І чим далі, тим гостріше бізнес потребує раціоналізації використання IT-ресурсів і автоматизації їх управління. Отож, будь-яка автоматизації в області IT, а особливо процесів розробки ПЗ є явищем необхідним.

IT-інфраструктура – це сукупність сервісів та різного роду систем компанії, осередків програмного забезпечення підприємства, обчислювальних програм, які забезпечують все необхідне задля вирішення бізнес-завдань підприємства.

Іншими словами IT-інфрастурктура включає в себе пристрої з виходом в Інтернет, бази даних, програмне забезпечення, корпоративну пошту і багато іншого. Всі ці елементи об’єднуються через інтернет і працюють у зв’язці, що дозволяє ефективно виконувати різні завдання і створити бізнес-середовище для скоординованої роботи всієї компанії і кожного її підрозділу. Коректна робота IT-інфраструктури забезпечує зв’язок між відділами або департаментами, які можуть бути фізично віддалені один від одного, передачу і отримання файлів та інформації, а також правильну роботу всіх сервісів компанії.[1]

Грамотно спроектована і побудована IT-інфраструктура створює для бізнесу ряд переваг, серед яких: підвищення прибутковості, оптимізація витрат, поліпшення продуктивності і результативності бізнес-процесів і технологічних процесів.

Попри грамотне проектування внутрішньої взаємодії бізнес елементів, інфраструктура передбачає врахування зовнішніх чинників, чи факторів які можуть вплинути на кінцевий результат роботи та буде, чи не основною, характеристикою що впливатиме на безпосередню відказостійкість основних процесів.

Погано розпланована інфраструктура може містити велику кількість обмежень, які в свою чергу створюватимуть залежності, яким повинні будуть відповідати усі як існуючі, так і майбутні процеси компанії. В гіршому випадку, можливість появи

небажанних результатів, некорректного функціювання та взаємодія певних частин інфраструктури, можливо навіть повна відмова працездатності, буде тільки збільшуватись.

На данний момент, існує ряд рішень які можуть бути використаними з цілью розміщення інфраструктури. Наприклад:

- Традиційні (On-Premise) фізичні сервери;
- Colocation;
- Hosting;
- IAAS (infrastructure as a service);
- PAAS (platform as a service);
- SAAS (software as a service).

Серед вище перелічених варіантів, розміщення інфраструктури, можливо виділити рішення що досі є актуальними але мають тенденцію відступати на другорядний план, за наявності більш простіших в опануванні та підтримці рішень, які постійно розвиваються за підтримки певної общини.

Цими варіантами є: традиційне розміщення інфраструктури на фізичних серверах, colocation та hosting. Кожний з цих варіантів передбачає необхідність висококваліфікованого спеціаліста, навички якого, будуть вирішувати подільшу працездатність проекту. Причиною вибору одного з вищеописаних варіантів розміщення інфраструктури, може виступати лише певна специфіка проекту, яка створює неординарні вимоги до платформи, при чому, ці вимоги можуть бути задовільненими лише при виборі конкретної платформи.

Вибір платформи для розміщення інфраструктури повинний бути доцільним. Саме тому, в будь-якому іншому випадку ліпшим рішенням буде обрати більш простішу платформу, наприклад IAAS, PAAS, SAAS, цим самим не створюючи перед собою додаткових вимог, яких потрібно буде дотримуватись для підтримки життєвого циклу продукту, та подальшої розробки.

Згідно статистики, що була зібрана ресурсом Statista, щорічно спостерігається тенденція до зросту кількості компаній, які розміщують інфраструктуру на певній платформі хмарних обчислень. На початок 2021 року 67% компаній використовують

хмарні платформи, що добре спостерігається на фоні обігу коштів, які було витрачено лише на ресурси хмарних обчислень [2]. Ця тенденція продовжує свій зріст та з кожним роком все більша кількість компаній відмовляється від розміщення інфраструктури в певному датацентрі, при цьому надаючи перевагу хмарним обчисленням.

1.2 Аналіз існуючих технологій розгортання програмної інфраструктури

У сучасному корпоративному IT існує багато факторів, які компанія повинна врахувати, щоб вирішити, чи відповідає інфраструктура необхідним стандартам. Незалежно від того, розміщує компанія свої програми в хмарі або вирішує тримати їх у датацентрі, безпека даних завжди буде першочерговою. Але для підприємств, що працюють у високорегульованих галузях, для них рішення щодо розміщення своїх додатків вже може бути прийнято заранння. А знаючи, що ваші дані та IT-інфраструктура знаходяться на власних серверах може надати відчуття спокою.

Підприємства від малих та середніх підприємств до великих фінансових установ намагаються надати легкі, безпечні та доступні платіжні рішення для своїх клієнтів та досягнення в послуги хмарної розробкипрокладають шлях вперед. Швидкість виходу на ринок та масштабованість мають вирішальне значення для успішності запуску FinTech, тому перед проведенням хмарного порівняння з локальним порівнянням необхідно провести обдумане дослідження. Обрати найліпший варіант можливо лише коли відомі усі позитивні та негативні сторони кожної із сторін.

Cloud (хмарні) обчислення в найпростішій формі - це процес використання віддалених серверів через Інтернет для розміщення таких ресурсів, як електронна пошта, мережа, ресурс для зберігання файлів, та програмне забезпечення. Управління, обробка та шифрування даних залишається під контролем стороннього оператора і надає можливість керувати звітами та аналітикою даних у режимі реального часу [2].

Безмежні з точки зору обчислювальної потужності та пропускна здатність зберігання даних, хмарні обчислення дозволяють фінансовим стартапам проникати у фінансову галузь та конкурувати на тих же умовах, що і великі фінансові установи. Зазвичай компанії використовуються за необхідності, компанії платять лише за те, що вони використовують, і можуть збільшувати або зменшувати відповідно до потреб клієнтів та ринку.

Хмарні сервіси, такі як платіжні шлюзи, захищені онлайн-транзакції та цифрові гаманці, є одними з найпоширеніших прикладів того, що пропонується, і можуть бути використані для забезпечення платежів для різних підприємств [3].

Ключовою характеристикою хмарних обчислень є те, що ви платите за те, що використовуєте. Постачальник хмарних послуг також піклується про підтримку своєї мережевої архітектури, надаючи вам свободу зосередитися на своєму додатку [4].

Більшість хмарних провайдерів пропонують набагато кращу інфраструктуру та послуги, ніж організації, створені індивідуально. Оренда стійочного простору в центрі обробки даних коштує лише незначну частину вартості встановлення та обслуговування внутрішньої інфраструктури такого масштабу. Крім того, можна значно заощадити технічний персонал, оновлення та ліцензії.

On-premise - це традиційний підхід, при якому все необхідне програмне забезпечення та інфраструктура для даного додатку перебувають у власному розпорядженні. У більшому масштабі це може означати, що бізнес розміщує власний центр обробки даних на місці [2].

Запуск додатків на місці включає придбання та обслуговування власних серверів та інфраструктури. Окрім фізичного простору, це рішення вимагає спеціалізованого IT-персоналу, кваліфікованого для обслуговування та моніторингу серверів та їхньої безпеки [4].

Це класична модель доступу до обчислювальних ресурсів, що припускає, що сервери, на яких встановлено корпоративне ПО, знаходяться безпосередньо у власності організації. Як і в випадку хмарних рішень, переваги тут слідують з недоліків, і навпаки. Вам доведеться стежити за розгортанням і підтриманням серверної інфраструктури, зате вона буде повністю відповідати вашим вимогам.

Colocation - це послуга, яка представляє собою розміщення власних серверів на спеціально обладнаному під ці цілі майданчику, а саме - в дата-центрі (центрі обробки даних). По суті - це фізичний хостинг, тобто ви самостійно встановлюєте сервер в обраному ЦОД, вибираєте програмне і апаратне забезпечення.

У дата-центрах створені спеціальні умови для утримання серверів, які складно дотримуватися в звичайному офіс.

У таблиці 1.1 наведено переваги та недоліки кожної зі способів розміщення інфрастуктури.

Таблиця 1.1 – Порівняння хмарних обчислень та виділених серверів

	On-premise	Cloud
Вартість	Вища	Нижча
Технічні рівень знань	Надзвичайно високий	Низький
Маштабованість	Обмежений апаратною частиною чи провайдером.	Вертикальна та горизонтальна масштабованість
Безпека та відповідність до вимог	Безпека локального хостингу повністю залежить від персоналу, який його підтримує	Cloud провайдер забезпечує безпечне середовище. Дешевші варіанти на ринку забезпечують менший рівень безпеки, ніж локальна інфраструктура
Контроль	Повний контроль та нескінченні можливості налаштування	Рівень гіпервізора між інфраструктурою і апаратними засобами. Немає прямого доступу до обладнання

Веб-хостинг — це послуга по розміщення веб-сайтів на спеціалізованих серверах і забезпечення доступу до них через мережу Інтернет. Передбачається налаштований сервер з усім необхідним ПО, з використанням якого працюють та розміщено інші веб-сайти чи програмні рішення [4].

Не існує чіткого переможця між On-premise рішеннями та рішеннями хмарних обчислень, які охоплюють усі бізнес-цілі.

Як на виділених серверах, так і в хмарі по-різному вирішують питання продуктивності, вартості, безпеки, відповідності, резервного копіювання та аварійного відновлення.

Вищеописані причини казують нам на перевагу хмарних обчислень над виділеними серверами в тому випадку, якщо немає чітких причин використовувати специфічний додатковий фукнціонал на виділеному сервері [5].

1.3 Аналіз існуючих рішень для розгортання інфраструктури додатків

Модель розгортання хмарових обчислень включає приватні, загальнодоступні (публічні) та гібридні хмари, кожна з яких має свої унікальні характеристики та використовується для різних цілей.

Приватні хмари представляють собою внутрішню хмарну інфраструктуру та послуги, що функціонують в межах корпоративної мережі. Організація може самостійно керувати такою хмарою або делегувати цю відповідальність зовнішньому підряднику. Ідеальний сценарій приватного хмарнного середовища - це хмара, розгорнута на території самої організації, повністю обслуговувана та контрольована її співробітниками. Приватні хмари мають привілеї, подібні до загальнодоступних, але з ключовою особливістю: підприємство відповідає за встановлення та управління хмарною інфраструктурою. Створення та підтримка внутрішнього хмарного середовища може бути витратною та складною задачею, і витрати на експлуатацію можуть перевищити вартість використання загальнодоступних хмар.

Приватні хмари володіють численними перевагами перед загальнодоступними. Зокрема, детальний контроль над ресурсами дозволяє компаніям налаштовувати конфігурації з врахуванням своїх унікальних потреб. Крім того, приватні хмари ідеально підходять для виконання завдань, довіра до яких з точки зору безпеки не допускає їх розміщення у загальнодоступних хмарах.

Загальнодоступні (публічні) хмари представляють собою хмарні послуги, які надаються постачальником і розташовані поза корпоративною мережею.

Ці хмарні послуги доступні для широкого загалу користувачів та компаній, і їх використання не обмежене внутрішньою мережею організації. Публічні хмари можуть забезпечувати різноманітні послуги, такі як обчислення, зберігання даних, машинне навчання та інші, через віддалені сервери, що ведуться постачальником хмарних послуг.

Важливою перевагою загальнодоступних хмар є їх гнучкість і масштабованість. Користувачі можуть використовувати ресурси за потребою, не турбуючись про обслуговування або розширення інфраструктури. Це дозволяє компаніям швидко реагувати на зміни обсягу роботи та підтримувати ефективну продуктивність. Загальнодоступні хмари також забезпечують високий рівень доступності, оскільки їхні ресурси розташовані в різних географічних областях.

Однак користувачі загальнодоступних хмар не мають такого самого рівня контролю над інфраструктурою, як у приватних хмарах. Безпека та конфіденційність даних можуть бути питаннями, особливо для тих секторів, які мають високі вимоги до збереження конфіденційності.

Враховуючи особливості обох типів хмар, підприємства часто вирішують використовувати гібридний підхід, комбінуючи приватні та загальнодоступні хмари відповідно до своїх конкретних потреб та обмежень.

Користувачі хмарних послуг не мають контролю над управлінням та обслуговуванням даних в хмарі, оскільки відповідальність цього лежить на власнику хмари. Постачальник хмарних послуг бере на себе завдання з установки, управління, надання та обслуговування програмного забезпечення, інфраструктури застосунків або фізичної інфраструктури. Клієнти сплачують лише за використані ресурси. Будь-яка компанія чи індивідуальний користувач може стати абонентом таких послуг, які надають швидкий та доступний за ціною спосіб розгортання веб-сайтів або бізнес-систем з великими можливостями масштабування. Приклади включають онлайн-сервіси Amazon EC2, Amazon Simple Storage Service (S3), Google Apps/Docs, Salesforce.com, Microsoft Office Web, Azure Storage Account.

Важливо відзначити, що послуги публічних хмар надаються переважно у вигляді стандартних конфігурацій, обмежуючи вибір користувача порівняно з системами, де ресурсами управляє сам користувач. Також, оскільки користувачі мають обмежений контроль над інфраструктурою, процеси, які вимагають суворих заходів безпеки та відповідності нормативним вимогам, не завжди підходять для реалізації в загальнодоступних хмарах.

Гібридні хмари представляють собою поєднання загальнодоступних і приватних хмар, часто розробляються підприємством, розподіляючи обов'язки з управління між підприємством і постачальником загальнодоступних хмар. Цей тип хмари надає послуги як загальнодоступні, так і приватні, і його використання є корисним в разі сезонної активності організації. У такому випадку, коли внутрішня IT-інфраструктура не в змозі впоратися з завданнями, частина потужностей переноситься до публічного хмари (наприклад, для обробки великих обсягів статистичної інформації, яка не є критичною для підприємства), а також для надання доступу користувачам до ресурсів підприємства (приватної хмари) через публічну хмару. Гібридні хмари дозволяють ефективно обслуговувати і критично важливі процеси, що вимагають високого рівня безпеки, а також менш важливі відділи [3].

Головним недоліком цього типу хмари є складність ефективного створення та управління подібними рішеннями. Для досягнення цієї мети необхідно отримувати послуги з різних джерел та організовувати їх в єдиний функціональний блок, уявляючи себе як єдиний джерело. Взаємодія між приватними і загальнодоступними компонентами може ускладнити рішення ще більше. Оскільки ця архітектурна концепція в області хмарних обчислень відносно нова, для цієї моделі надходять нові практичні рекомендації та інструменти, і її широке впровадження може затриматися, поки вона не буде більш детально досліджена.

За словами Тома Біттмана, віце-президента та провідного аналітика компанії "Gartner", серед трьох вищезазначених моделей розгортання хмар наразі найбільш актуальною для бізнесу є приватна хмара. Біттман виділяє п'ять ключових аспектів, які допомагають краще зрозуміти структуру приватних хмар [5].

Залежно від типу хмарних послуг, ними можуть володіти та управляти як постачальник, так і користувач, або обидва одночасно. Також можуть розрізнятися права доступу до ресурсів (табл. 1.2).

Хмарові обчислення — це не лише віртуалізація. Хоча віртуалізація серверів та інфраструктури є важливою основою для приватних хмарних обчислень, сама за собою віртуалізація та управління віртуалізованим середовищем не забезпечують хмарової ідентифікації. Віртуалізація дозволяє ефективно організовувати, об'єднувати в пул та динамічно надавати ресурси інфраструктури, такі як сервери, робочі станції, простори для зберігання, мережеве обладнання, програмне забезпечення для сполучення та інше. Проте для того, щоб технологічне середовище було визнане як хмарне, необхідні інші компоненти, такі як віртуальні машини, операційні системи або контейнери для сполучення, високодоступні операційні системи, програмне забезпечення для геодистриб'юції обчислень, засоби абстрагування ресурсів зберігання та засоби масштабування і кластеризації.

Таблиця 1.2 - Обслуговування та управління різними видами хмарних ресурсів

Вид хмари	Ким обслуговується інфраструктура	Хто є власником інфраструктури	Де знаходиться інфраструктура	У кого є доступ
Публічне	Зовнішнім провайдером	Зовнішній провайдер	У зовнішнього провайдера	У будь-якого
Приватне/Суспільне	Користувачем або зовнішнім провайдером	Користувач або зовнішній провайдер	У зовнішнього провайдера або у користувача	У ваторизованого користувача
Гібридне	Користувачем і зовнішнім провайдером	Користувач і зовнішній провайдер	У зовнішнього провайдера і у користувача	У авторизованих і у будь-яких зовнішніх користувачів

У відміну від загальнодоступних чи гібридних, термін "приватна хмара" вказує на використання ресурсів однією організацією або на те, що хмарні ресурси цієї організації повністю ізольовані від інших. Хмара не завжди призначена для економії грошей, і найпоширенішою помилкою є вважання, що вона обов'язково призведе до економії. Хоча приватна хмара дозволяє ефективно перерозподіляти ресурси для задоволення корпоративних потреб і може знизити капітальні витрати на обладнання, вона також потребує інвестицій в автоматизацію, і сама економія може не повністю компенсувати ці витрати.

Таким чином, основним стимулом для переходу до хмарової моделі має бути не лише економія, але й швидкість введення на ринок, можливість швидко адаптуватися та динамічно масштабуватися відповідно до попиту. Це дозволяє підвищити швидкість впровадження нових сервісів. Важливо відзначити, що приватна хмара не завжди розгортається у власнику, але вона передбачає конфіденційність, а не конкретне місце розташування, власність чи самостійне управління. Багато постачальників пропонують нелокальні приватні хмари, виділяючи ресурси для одного замовника, але ізолюючи їх від інших за допомогою технологій, таких як віртуальна приватна мережа (VPN). Приватна хмара, подібно до публічної, не обмежується лише інфраструктурними сервісами, і серверна віртуалізація виступає важливою складовою для приватних хмарних обчислень, які можуть включати в себе не лише послуги IaaS, але і PaaS, наприклад, для розробки та тестування нового програмного забезпечення.

На сьогоднішній день найшвидше зростаючим сегментом хмарних обчислень є Інфраструктура як послуга (IaaS). Цей сегмент надає низькорівневі ресурси Центру обробки даних (ЦОД) у зручній формі, залишаючи без змін основні принципи функціонування.

Для розробки нових додатків, спеціально призначених для хмарового середовища і надають абсолютно нові послуги, розробникам зручніше використовувати Платформу як послуга (PaaS). Приватна хмара може втратити характер приватності, оскільки, з одного боку, вона надає переваги, такі як швидкість

перебудови, масштабованість і ефективність, а також звільняє від деяких загроз безпеки, які характерні для загальнодоступних хмар [6].

З іншого боку, з часом рівень обслуговування, безпека і контроль відповідності вимогам в загальнодоступних хмарних сервісах непередбачувано підніматимуться. Тому деякі приватні хмари, можливо, повністю перетворяться в загальнодоступні. Більшість служб приватного хмари, ймовірно, буде еволюціонувати в гібридні хмарні сервіси, розширюючи свої можливості за рахунок використання загальнодоступних хмарних послуг та інших зовнішніх ресурсів.

1.4 Аналіз методів розгортання інфраструктури додатків

Хмарні обчислення надають обчислювальні ресурси на вимогу, які відокремлені від фізичного обладнання та необхідної базової конфігурації. Автономні програмні системи забезпечують ці обчислювальні ресурси в хмарі для досягнення автоматизації, яку пропонують хмарні обчислення. Завдяки такій автоматизації можна програмно контролювати та маніпулювати наявними ресурсами, взаємодіючи з хмарними провайдерами. Таким чином, зміни інфраструктури (наприклад, масштабування ресурсів) можуть бути впроваджені швидше та надійніше, і вони функціонують, в основному, без ручної взаємодії, але все одно з можливістю контролювати весь процес і повертати зміни, якщо щось не йде за планом [7].

Інфраструктура як код (IaC) - це підхід до автоматизації розгортання та змін інфраструктури шляхом визначення бажаних станів ресурсів та їх взаємозв’язків у коді. Код написаний спеціалізованими, зручними для читання мовами інструментів IaC. Фактичні ресурси в хмарі створюються (або змінюються) під час виконання коду. Потім цей інструмент пропонує інструменту взаємодіяти з хмарним провайдером або системою розгортання від вашого імені, щоб застосувати необхідні зміни, не використовуючи веб-інтерфейс хмарного провайдера. Код може бути модифікований при необхідності - після виконання коду інструмент IaC подбає про те, щоб знайти відмінності між бажаною інфраструктурою в коді та фактичною інфраструктурою в хмарі, вживаючи заходів для того, щоб фактичний стан дорівнював бажаному.

За принципами опису інфраструктури можна виділити такі способи: імперативний та декларативни [7].

Імперативний підхід - це парадигма, заснована на складанні алгоритму дій (інструкцій / команд), які змінюють стан (інформацію / дані / пам'ять) програми. Першими мовами програмування, заснованими на такому підході, були машинні коди і асемблери. Простими словами це процесс опису команд/ шляху яким нам необхідно слідувати для отримання бажаного результату інфраструктури. Наприклад скрипт, що описує ряд команд для автоматичного встановлення будь якого програмного забезпечення, вважається імперативним способом оскільки описує сам шлях отримаання кінцевого результату [5].

Декларативне програмування - це парадигма, при якій описується бажаний результат, без складання детального алгоритму його отримання. Як приклад, можна привести HTML і SQL. При створенні HTML ми за допомогою тегів описуємо, яку хочемо отримати сторінку в браузері [5].

На данний момент, імперативний підхід використовується лише в крайніх випадках, коли неможливо описати бажаний стан з використанням декларативним методу. В такому випадку імперативний підхід може описати бажаний стан але тим не меш буде інтегрований з декларативним підходом. В деяких програмних засобах цей процес носить назву модуль.

1.5 Аналіз існуючих програмних засобів розгортання інфраструктури додатків

Впровадження методів Agile та DevOps сприяло скороченню циклів програмування до тижнів і встановленню стабільного інтервалу розгортання. Сучасна практика безперервної інтеграції інфраструктури дозволяє впроваджувати оновлення ще ефективніше, навіть протягом декількох днів чи годин. Це досягається через регулярне відправлення коду до спільного сховища, щоб розробники могли швидко виявляти дефекти за допомогою автоматизованих тестів і виправляти їх негайно.

Усі існуючі програмні засоби відрізняються не тільки різними підходами (імперативний, декларативний), а ще і за різними методами доставлення кінцевої необхідної конфігурації. Це метод “push” і “pull”. Основна різниця хто ініціює зміни в кінцевій точці/сервері. В pull режимі кінцевий хост сам ініціює отримання конфігурації. В push режимі конфігурація надсилається з основного(master) сервера [5].

Спочатку важливо відзначити, що розробники інструментів даного типу завжди прагнуть створити продукт, який об'єднує можливості конфігурації безперервної інтеграції, доставки, розгортання та тестування в одному місці.

Отже, всі інструменти, які розглядаються у даному розділі, можна розглядати як готовий набір рішень для налаштування всього пайплайну проекту. Засновуючись на власному досвіді використання різноманітних інструментів такого роду і широкому обсягу теоретичної інформації, я прийняв рішення використовувати наступні критерії для порівняльного аналізу (табл. 1.3):

- можливості хостингу: Різні програмні рішення відрізняються своєю системою управління інфраструктурою. Хмарні інструменти розміщуються на стороні постачальника, вимагають мінімальної конфігурації та можуть бути налаштовані за потребою. Існують також власні рішення, які вимагають від вашої внутрішньої команди DevOps відповідальності за розгортання та підтримку. Локальні послуги можуть забезпечити більшу гнучкість, але хостингові рішення пропонують більшу масштабованість та звільняють від труднощів налаштування;

- зручність використання: Деякі інструменти можуть значно спростити процес збирання завдяки чіткому та зрозумілому GUI та UX. Добре розроблений інтерфейс може економити час на етапі впровадження;

- інтеграція та підтримка програмного забезпечення: Оцінка можливості інтеграції інструменту CI з зовнішніми сервісами та іншими інструментами, такими як системи управління проектами (наприклад, Jira), інцидент-менеджмент (наприклад, PagerDuty та Bugzilla), інструменти статичного аналізу та інші. Інструмент для процесу інтеграції повинен бути гнучким для підтримки різних типів

інструментів побудови та програмного забезпечення для управління версіями або VCS (наприклад, Subversion, Perforce, Git) тощо [6];

– підтримка контейнеризації: Наявність плагіну або конфігурації для розгортання контейнерних інструментів, таких як Kubernetes і Docker, спрощує інтеграцію інструмента розгортання з цільовим середовищем програми;

– бібліотека коду багаторазового використання: Бажано мати інструмент з різноманітним загальнодоступним сховищем плагінів, що має відкритий вихідний код і надає можливість користувачам доповнювати його власними напрацюваннями.

Таблиця 1.3 – Опис програмних засобів розгортання інфраструктури

Назва ПЗ	Створено організацією	Метод	Підхід	Написано на
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby
Pulumi	Pulumi	Push	Declarative	Typescript, Python, Go
Otter	Inedo	Push	Declarative and imperative	-
SaltStack	SaltStack	Push and Pull	Declarative and imperative	Python
Puppet	Puppet (2005)	Pull	Declarative	Ruby
CFEngine	CFEngine	Pull	Declarative	-
DSC	Microsoft	Push/Pull	Declarative/Imperative	PowerShell
Terraform	HashiCorp (2014)	Push	Declarative	Go
ARM	Microsoft	Push	Declarative and imperative	-
Ansible / Ansible Tower	RedHat (2012)	Push	Declarative and imperative	Python

На основі аналізу ринку та результатів незалежного опитування користувачів було вибрано кілька найбільш популярних інструментів, які будуть подальше

детально проаналізовані та порівняні з використанням вищезазначених критеріїв. Серед них: Jenkins, TeamCity, Azure DevOps, Travis CI, Circle CI, GitLab CI, GoCD.

Проаналізуємо кожний інструмент для розгортання хмарної інфраструктури.

Jenkins – це проект з відкритим кодом, написаний на Java, і сумісний з операційними системами, такими як Windows, macOS, і іншими, що подібні до Unix. Являючи безкоштовним і підтримуваним спільнотою, Jenkins може стати вашим основним інструментом для постійної інтеграції. Хоча переважно використовується для локального розгортання, Jenkins також може ефективно працювати на хмарних серверах. Його інтеграція з Docker та Kubernetes розширює можливості використання контейнерів і забезпечує більш часті випуски. Jenkins можна встановити через нативні системні пакети, Docker або запустити автономно на будь-якій машині з встановленою Java [8].

Вимоги до системи:

- сервер з встановленою операційною системою Windows, macOS або Unix;
- мінімум 256 МБ оперативної пам’яті, хоча рекомендується використання більше 512 МБ;
- мінімум 1 ГБ дискового простору;
- встановлена Java 8.

Основні переваги:

- безмежні інтеграції: Jenkins може ефективно інтегруватися з практично будь-якою зовнішньою програмою, використовуваною для розробки програм. Це дозволяє використовувати контейнерні технології, такі як Docker і Kubernetes, у нестандартних сценаріях. Рецензенти G2 Crowd відзначають: "Не існує кращого інструменту для інтеграції ваших сховищ та баз коду з інфраструктурою розгортання";

- велика бібліотека плагінів: Jenkins має доступну багату бібліотеку плагінів, яка охоплює широкий спектр інструментів, таких як Git, Gradle, Subversion, Slack, Jira, Redmine, Selenium, Pipeline. Плагіни Jenkins включають п'ять областей: платформи, інтерфейс користувача, адміністрування, управління вихідним кодом і, найчастіше, управління збіркою. Jenkins вирізняється всебічною інтеграцією плагінів, що відсутня у багатьох інших інструментах CI. Спільнота Jenkins активно

заохочує користувачів до розширення функціоналу, надаючи навчальні ресурси;

- активна спільнота: Спільнота Jenkins проводить екскурсії для ознайомлення з основами та надає вдосконалені навчальні посібники для ефективного використання інструменту. Крім того, щорічно проводиться конференція "DevOps World | Світ Jenkins";

- розподіл складових і випробувальних навантажень на декількох машинах: Jenkins використовує архітектуру Master-Slave, де майстер є головним сервером, що відстежує роботу рабів – віддалених машин, що використовуються для розповсюдження процесів збірки програмного забезпечення та тестування навантажень.

Головні недоліки Jenkins:

- неповна документація: Часто відсутній достатній опис, особливо в частині створення трубопроводів, що ускладнює роботу інженерів і вимагає додаткових зусиль для вирішення завдань.
- недосконалий інтерфейс користувача: Інтерфейс вважається застарілим, не відповідає сучасним принципам дизайну, і відсутність пробілів призводить до заплутаності та переповнення виглядів. Багато функцій та значків прогресу мають низьку якість (пікселізацію) і не автоматично оновлюються після завершення завдань;
- ручне відстеження та оновлення: Вимагає вручну відстежувати сервер Jenkins та його рабів, щоб зрозуміти взаємозалежності між плагінами та регулярно їх оновлювати;
- найбільше підходить для великих проектів: Хоча Jenkins ефективний для великих проектів з великою кількістю налаштувань через плагіни, процес конфігурації може зайняти значний час. Для швидкого початку роботи рекомендується розглядати альтернативи.

TeamCity - інструмент безперервної інтеграції для створення та розгортання різних типів проектів, який працює в середовищі Java та інтегрується з Visual Studio та IDE. Зручно встановлювати як на серверах Windows, так і на Linux, підтримує .NET та проекти з відкритим стеком.

У версії TeamCity 2019.1 впроваджено новий інтегральний інтерфейс та вбудовану підтримку GitLab. Також підтримуються запити на потягування сервера GitLab та Bitbucket. Випуск включає аутентифікацію на основі токенів, виявлення, звітування про тести Go та запити на AWS Spot Fleet.

Команди достатньо часто обирають TeamCity для реалізації великої кількості завдань, таких як аутентифікація, розгортання та тестування функцій у нестандартних сценаріях, а також для підтримки Docker. Цей інструмент крос-платформений, сумісний з останніми версіями операційних систем Windows, Linux та macOS, і працює на платформах Solaris, FreeBSD, IBM z/OS та HP-UX. Після встановлення TeamCity готовий до використання без додаткових налаштувань, і він вражає своїм набором унікальних функцій, таких як детальні звіти про історію, миттєвий зворотній зв'язок щодо тестових помилок та можливість повторного використання налаштувань, що уникне дублювання коду [8].

Щодо цінових моделей, TeamCity пропонує безкоштовну версію з повним функціоналом, але з обмеженнями - 100 конфігурацій побудови та три агенти побудови. Додавання одного додаткового агента та 10 конфігурацій побудови обійдеся в 299 доларів США. Крім того, TeamCity пропонує 60-денну пробну хмару для тих, хто не бажає встановлювати його локально. Для корпоративного використання доступне платне видання з ціною, що залежить від кількості використовуваних агентів. Компанія надає 50% знижки стартапам і безкоштовні ліцензії на проекти з відкритим кодом.

Основні переваги TeamCity:

- підтримка .NET: TeamCity інтегрується з .NET інструментами краще, ніж будь-який інший інструмент CI, забезпечуючи важливі інструменти, такі як аналіз покриття коду, різноманітні фреймворки тестування .NET та аналіз статичного коду;

- широка підтримка VCS: TeamCity дозволяє створювати проекти лише за URL-адресою сховища VCS, підтримуючи всі популярні VCS, такі як AccuRev, ClearCase, CVS, Git, Gnu bazaar, Mercurial, Perforce, Borland StarTeam, Subversion, Team Foundation Server, SourceGear Vault, Visual SourceSafe та IBM Rational ClearCase. На рисунку 1.1 наведено схематичне зображення зовнішніх інтеграцій TeamCity [15].

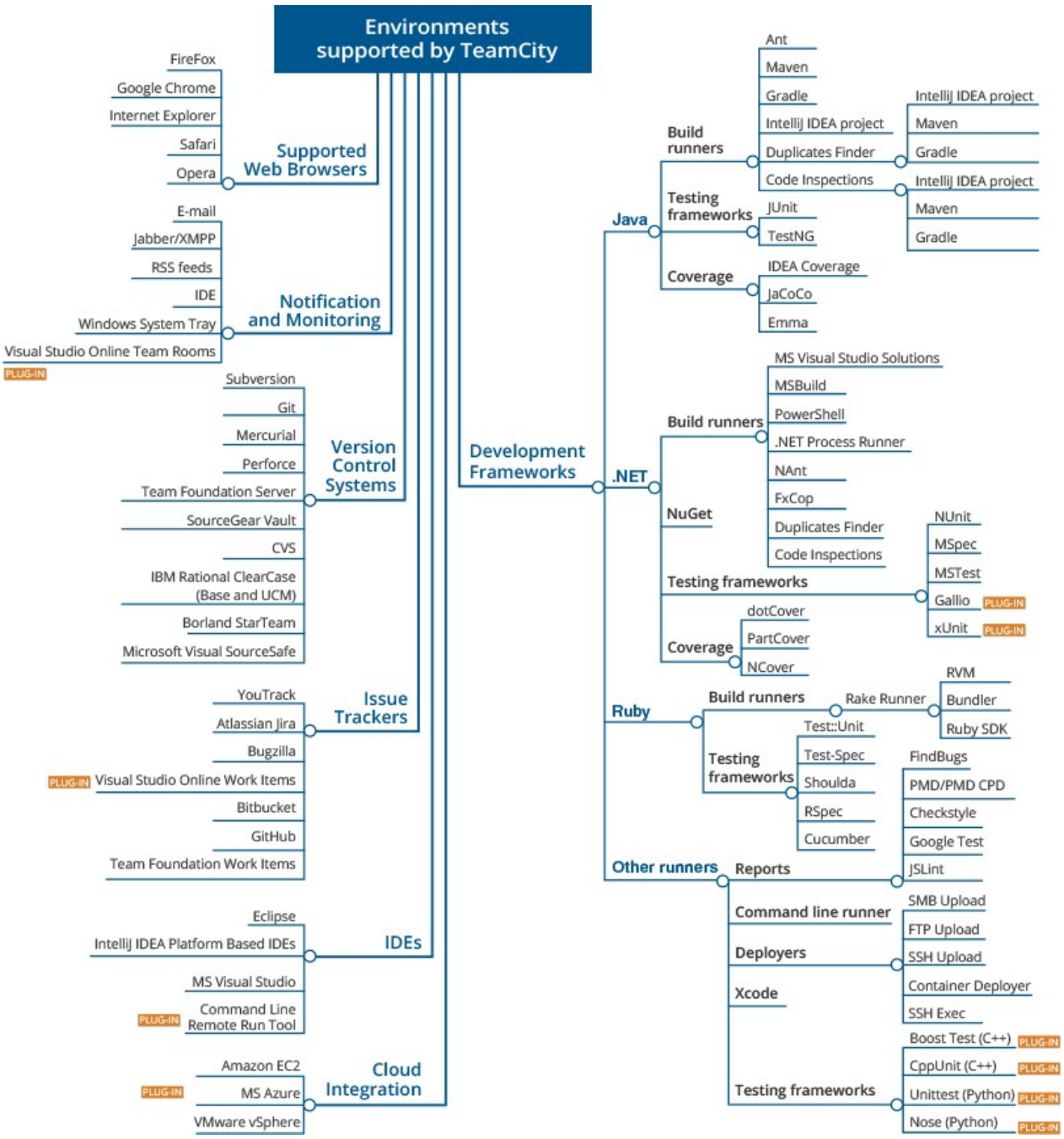

Рисунок 1.1 – Схема основних зовнішніх інтеграцій інструменту TeamCity

- компетентна документація: Інструкції для користувача, надані TeamCity, вражають своєю компетентністю та ретельністю. Загальну функціональність можна легко освоїти, переглядаючи це докладне та обширне керівництво;

- простота налаштування: TeamCity пропонує легку настройку та готовий до використання одразу після встановлення. Його інтерфейс дозволяє швидко і зручно налаштовувати параметри для початку роботи з проектом;

- вбудовані функції: TeamCity має значний функціонал, який полегшує процес

створення проекту. Серед цих функцій - детальні звіти про будівлі, інформація про помилки та будь-які зміни, контроль джерел та інструменти для будівництва ланцюга. Однією з ключових особливостей TeamCity є "Опублікувати артефакти", що дозволяє розгортати код або навіть будувати безпосередньо в будь-якому середовищі. Інструмент показує хід процесу складання на кожному етапі, а також кількість тестів, які залишилися до завершення будівлі. Крім того, ця функція дозволяє повторно виконувати будь-які неуспішні тести відразу після виконання нічної будівлі, щоб уникнути витрат часу на це наступного ранку.

Недоліки TeamCity:

- складна крива навчання: Хоча TeamCity славиться своїм естетичним інтерфейсом, для новачків він може виглядати трохи складним і вимагати серйозного зусилля для освоєння, зокрема через широкий спектр параметрів конфігурації;

- ручний процес оновлення: Перехід між основними версіями TeamCity вимагає довготривалого та ручного процесу, що може викликати труднощі для адміністраторів серверів.

Bamboo: Найкраща інтеграція з продуктами Atlassian.

Окрім сприяння інтеграції, Bamboo володіє функціональністю розгортання та управління випусками. Відмінно інтегрується з іншими продуктами Atlassian, такими як Bitbucket, Jira і Confluence, уникнувши при цьому потреби у великій кількості плагінів, як у Jenkins. Робоча платформа Bamboo охоплює Windows, Linux, Solaris та macOS. Для користувачів Linux, офіційна документація рекомендує створення спеціального користувача для попередження можливого зловживання.

Bamboo пропонує 30-денну пробну версію та різні опції користувацького рівня, включаючи необмежену кількість локальних агентів і 10 завдань. Ціна на платформу збільшується з 10 до 126,500 доларів США в залежності від кількості віддалених агентів. Усі ціни включають 12-місячний термін обслуговування, який можна подовжити за додаткову плату. Програмне забезпечення Atlassian є безкоштовним для проектів з відкритим кодом, які відповідають їх критеріям.

Головні переваги:

- Bitbucket Pipelines: Після відміни хмарового варіанту Bamboo у 2016 році Atlassian випустив Bitbucket Pipelines – хмарну альтернативу, вбудовану в Bitbucket, для управління сховищами Git. Bitbucket Pipelines повністю інтегровані з Bamboo,

забезпечуючи автоматичну побудову, тестування та розгортання коду. Використовуючи Docker, Bitbucket Pipelines надає швидкі та ефективні побудови. Одночасно, сервер Bamboo доступний для попередньої установки та може бути розміщений у хмарному контейнері або віртуальній машині;

- кілька методів повідомлення: Bamboo Wallboard демонструє результати побудови на моніторі та відправляє звіти поштою або в чат розробників (наприклад, HipChat, Google Talk);

- потужна і проста інтеграція: Bamboo підтримує більшість ключових технологій, включаючи CodeDeploy, Docker, Maven, Git, SVN, Mercurial, Ant, AWS, та Amazon S3 Buckets. Функція "дозволи на середовище" дозволяє розгортати в різних середовищах;

- документація та супровід: Bamboo має докладну та інформативну документацію, а Atlassian гарантує кваліфіковану підтримку. Однак розмір спільноти менший порівняно з Jenkins.

Головні недоліки:

- погана підтримка плагінів: У порівнянні з Jenkins та TeamCity, Bamboo має обмежену кількість підтримуваних плагінів. На момент огляду, лише 208 додатків вказано у сховищі Atlassian;

- складний перший досвід роботи: Деякі користувачі скаржаться на складність перших налаштувань для розгортання, і для повного розуміння можливостей системи потрібен час.

Bamboo ідеально підходить для тих, хто шукає інтеграцію з продуктами Atlassian у своєму стеку.

Travis CI: Зріле рішення для неперервної інтеграції з простою інтеграцією GitHub.

Travis CI – це сервіс неперервної інтеграції, який використовується для створення та тестування проектів. Система автоматично реагує на нові коміти, які надходять у сховище GitHub, та виконує побудову проекту та відповідні тести.

Інструмент підтримує різноманітні конфігурації побудов та мов, таких як Node, PHP, Python, Java, Perl тощо. Перші 100 збірок є безкоштовними, а після цього доступні чотири тарифних плани для хобі-проектів (69 доларів США на місяць) і різних розмірів команд (від 129 до 489 доларів США на місяць), в залежності від

кількості одночасних завдань.

Головні переваги Travis CI:

- просте налаштування та конфігурація: Travis CI не вимагає складних встановлень і може бути легко налаштований лише за допомогою реєстрації та додавання проекту. Файл конфігурації у форматі YAML розміщується в кореневій директорії проекту, а користувацький інтерфейс є дружелюбним і зручним для моніторингу збірок;

- пряме з'єднання з GitHub: Travis CI відмінно взаємодіє з системами керування версіями, зокрема з GitHub. Для проектів з відкритим кодом GitHub, інструмент є безкоштовним та автоматично реагує на кожен коміт;

- резервне копіювання останньої збірки: При кожній новій збірці Travis CI створює клон сховища GitHub у новому віртуальному середовищі, забезпечуючи наявність резервної копії.

Недоліки використання Travis CI:

- відсутність вбудованої неперервної доставки та розгортання: У відміну від інших інструментів CI в списку, Travis CI не підтримує процесів постійної доставки;

- хостинг лише для GitHub: Так як Travis працює тільки з проектами, розміщеними у GitHub, команди, які використовують GitLab або інші альтернативи, повинні звертатися до інших інструментів CI;

- в підсумку, Travis CI є ідеальним вибором для проектів з відкритим кодом, які потребують тестування в різних середовищах, та відзначається швидким початком інтеграції.

CircleCI: Простий та корисний інструмент для CI при розробці проектів на ранній стадії.

CircleCI - це інструмент для неперервної інтеграції та постійної доставки (CI/CD), спрямований на прискорення розробки та випуску програмного забезпечення. Цей інструмент забезпечує автоматизацію усього циклу розробки, від написання коду та його тестування до розгортання.

CircleCI легко інтегрується з GitHub, GitHub Enterprise та Bitbucket, щоб автоматично реагувати на новий код та розпочинати процеси збірки. Система може бути розгорнута в хмарному варіанті або використовувати власні сервери та

брандмауер для приватної інфраструктури.

CircleCI пропонує безкоштовний пробний період для macOS, дозволяючи збирати як на Linux, так і на macOS. Безкоштовний Linux-пакет включає один контейнер, а можливість розширення (за 50 доларів США на місяць) дозволяє обирати рівень паралелізації від одного до 16 одночасних завдань. Плани для macOS мають різні вартості від 39 до 249 доларів США на місяць, залежно від рівня паралелізації та підтримки електронної пошти.

Якщо у вашого підприємства є особливі потреби, CircleCI пропонує індивідуальні тарифні плани на основі особистого використання. Для проектів з відкритим кодом доступно чотири безкоштовні Linux-контейнери та план macOS Seed з одночасністю 1x.

Переваги CircleCI:

- простий інтерфейс користувача: CircleCI відзначається зручним веб-додатком на одній сторінці, який легко використовувати для управління збірками та робочими процесами;

- якісна підтримка клієнтів: Швидка підтримка є важливим аспектом CircleCI, і користувачі відзначають високу швидкість відповіді на їх запити;

- широкі можливості тестування: CircleCI може запускати різні типи тестів, включаючи веб, мобільні та контейнерні середовища;

- розумне вирішення встановлення вимог та залежностей: Замість ручного встановлення середовищ, CircleCI може отримувати дані з різних проектів, використовуючи параметри відмітки.

Недоліки CircleCI:

- надмірна автоматизація: CircleCI може змінювати середовище без попередження, що може становити проблему, оскільки вона не чекає на інструкції від користувача, як це робить, наприклад, Jenkins;

- відсутність кешування зображень Docker: Відсутність можливості кешувати зображення Docker може ускладнити деякі аспекти роботи;

- відсутність тестування в ОС Windows: CircleCI не надає можливості для будування та тестування в середовищі Windows.

Загалом, CircleCI є ідеальним інструментом для швидкого розгортання

проектів на ранній стадії розробки, якщо ви шукаєте ефективну і просту систему CI.

GitLab CI: GitLab представляє собою комплексний інструмент для керування різними аспектами життєвого циклу розробки програмного забезпечення. Основним продуктом є веб-менеджер репозиторіїв Git, який включає в себе функції відстеження проблем, аналітики та Wiki з розгорнутою документацією.

GitLab CI надає можливість запускати збірки, виконувати тести та розгортати код з кожним комітом або натисканням кнопки. Користувачі можуть створювати завдання на віртуальній машині, контейнері Docker або на іншому сервері. Деякі з ключових можливостей GitLab CI включають:

- керування кодом та даними проектів: За допомогою інструментів розгалуження, користувачі можуть переглядати, створювати та керувати кодом та даними проектів;

- єдина розподілена система управління версіями: GitLab дозволяє розробникам швидко ітерувати та доставляти кінцеві продукти з єдиної системи управління версіями

- автоматизація CI-процесів: GitLab CI допомагає командам повністю сприймати Continuous Integration, автоматизуючи збірку, інтеграцію та перевірку вихідного коду;

- забезпечення безпеки: Інструмент забезпечує сканування контейнерів, статичне тестування безпеки додатків (SAST), динамічне тестування безпеки додатків (DAST) та сканування залежностей для доставки захищених програм разом із дотриманням ліцензій.

Автоматизація випусків і доставка додатків: GitLab CI допомагає автоматизувати та прискорити випуски та доставку додатків.

Відкритий вихідний код: GitLab CI є проектом з відкритим вихідним кодом.

GoCD: GoCD є інструментом з відкритим вихідним кодом для створення та випуску програмного забезпечення, спрямованим на підтримку сучасної інфраструктури для CI/CD.

Розроблений компанією ThoughtWorks, GoCD доступний для платформ Windows, Mac та Linux, і використовує систему трубопроводів (pipelines) для організації робочого процесу. Трубопроводи допомагають утримувати весь робочий

процес, розділяючи завдання на менші частини з командами, відповідальними за кожен етап, що полегшує комунікацію та підвищує ефективність.

Однією з переваг GoCD є його можливість обробки складних сценаріїв та обширний список плагінів. Деякі користувачі можуть користуватися його безкоштовною версією за винятком комерційних ліцензій [3].

1.6 Постановка задачі

Аналізуючи переваги та недоліки аналогів, можна зробити висновок, що інформаційна технологія розгортання інфраструктури має досить великий попит та є актуальною. Ті інструменти що наразі існують не дають можливості розгорнути інфраструктуру з усіма необхідними функціальними частинами та налаштуваннями.

Отже створений програмний засіб повинний мати наступні можливості:

- кількість вхідних конфігурацій не менше – 20;
- платформа програмного засобу .Net;
- підтримка розгортання 3 і більше середовищ(environments);
- можливість інтегрування інфраструктури у приватну мережу;
- підхід до розгортання IAC;
- опис інфраструктури - декларативний;
- можливість перестворення будь-якого елементу інфраструктури, без впливу на інші.

Процес розгортання не повинний передбачати жодних ручних налаштувать, та усі дії повинні бути автоматизованими.

1.7 Висновок до розділу 1

В даному розділі проведено аналіз предметної області, розглянуто сучасні аналоги та основні способи розміщення інфраструктури. Проведено їх порівняння та виявлено основні недоліки засобів розгортання інфраструктури. Сформульовано вимоги до створюваної інформаційної технології по розгортанню інфраструктури для додатків на платформі .Net.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ДОДАТКІВ НА ОСНОВІ ХМАРНОГО ПРОФАЙДЕРА AZURE

2.1 Особливості роботи додатків на платформі .Net

Під терміном "Microsoft .NET" розуміється інтегрована система розробки, розгортання і виконання складних, розподілених програмних систем. У цьому наборі базових класів включено функціонал для роботи з рядками, введенням-виведенням даних, багатопоточністю і іншими завданнями. Класи для роботи з даними надають можливість використання SQL-запитів, ADO.Net і обробки XML даних.

Загальномовне середовище виконання (CLR) активізує виконуваний код, проводить для нього перевірку безпеки, розміщує його в пам'яті та запускає. Воно також відповідає за збірку сміття. Після компіляції програми на одній з алгоритмічних мов платформи MS.Net, отриманий програмний код представляється загальною проміжною мовою (CIL).

.NET Framework, випущений компанією Microsoft у 2002 році, є програмною платформою, яка базується на CLR і призначена для використання різними мовами програмування. Ці можливості доступні для будь-яких мов, які використовують середовище CLR.

Основною метою .NET Framework було забезпечити свободу розробникам, дозволяючи їм створювати додатки різних типів, придатні для виконання на різних пристроях та в різних середовищах. Крім того, платформа орієнтується на системи, які працюють під управлінням операційних систем Microsoft Windows.

Програма, розроблена для .NET Framework будь-якою мовою програмування, яку підтримує платформа, спочатку компілюється в єдиний байт-код Common Intermediate Language (CIL), раніше відомий як Microsoft Intermediate Language (MSIL). У контексті .NET, цей байт-код формує збірку (assembly), яка представляє собою єдиний і виконуваний файл програми.

Після створення збірки код може виконуватися двома способами. По-перше,

він може бути виконаний віртуальною машиною Common Language Runtime (CLR), яка є ключовою частиною .NET Framework. Використання віртуальної машини є переважним, оскільки це звільняє розробників від необхідності враховувати особливості конкретного обладнання, дозволяючи коду працювати на будь-якому пристрої, який підтримує CLR. В цьому випадку вбудований JIT-компілятор (Just-In-Time) перетворює байт-код на льоту в машинний код для конкретного процесора.

Другий спосіб - трансляція коду в виконуваний код для конкретного цільового процесора за допомогою утиліти NGen.exe. Це використовується для оптимізації продуктивності, забезпечуючи виконання програми у вигляді нативного коду, призначеного конкретному апаратному середовищу.

CLR, крім того, відповідає за базові принципи безпеки, управління пам'яттю та систему винятків, забезпечуючи надійність та ефективність виконання програм.

Архітектура .NET Framework описана в специфікації Common Language Infrastructure (CLI), розробленій Microsoft, затвердженій ISO і ECMA. Ця специфікація визначає типи даних .NET, формат метаданих для програмної структури, систему виконання байт-коду та інші важливі компоненти платформи.

.NET є багаторівневим, модульним і ієрархічним. Мови програмування є верхнім та найбільш абстрактним рівнем, предоставляючи високорівневий інтерфейс для розробників. CLR, найнижчий рівень, забезпечує виконання програми та управління системними ресурсами. Ця ієрархія важлива для розуміння взаємозв'язків та ефективного використання різних компонентів .NET Framework.

.NET Framework складається з модулів, які кожен відповідає за певну область функціоналу. Це дозволяє розробникам працювати на високому рівні абстракції, використовуючи лише ті служби та інструменти, які їм необхідні для конкретного завдання. Загальна архітектура .NET Framework відображена на рисунку 2.1, і ця структура сприяє ефективності та гнучкості розробки програмного забезпечення на платформі .NET.

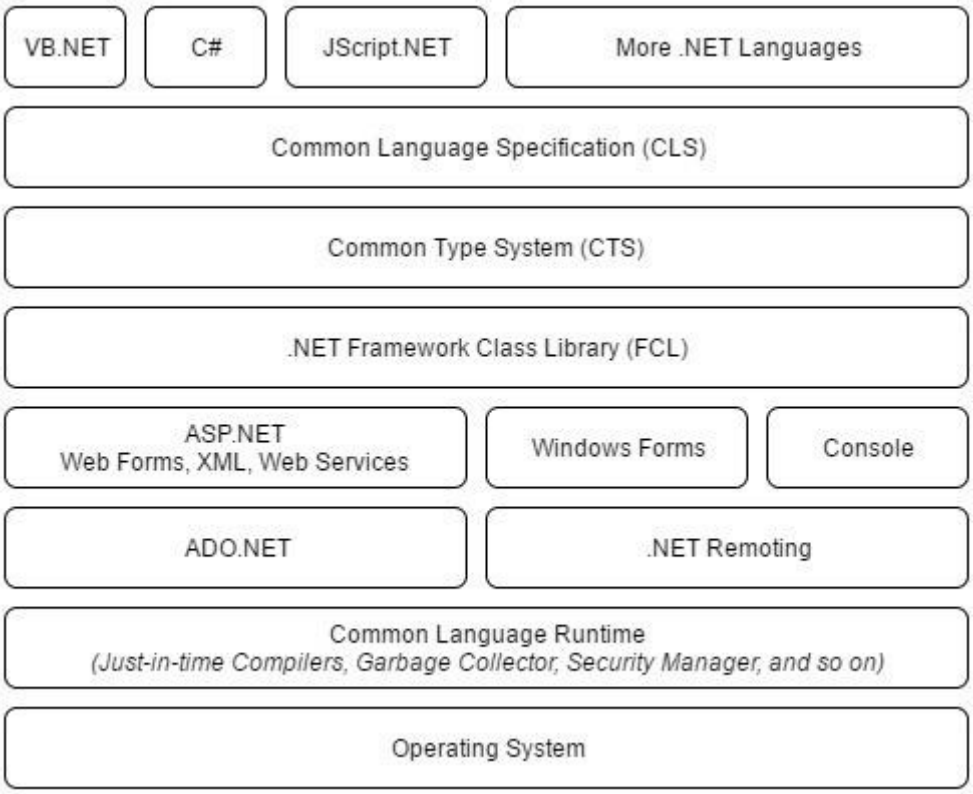

Рисунок 2.1 - Архітектура платформи .NET

Об'єктні класи .NET, які доступні для всіх підтримуваних мов програмування, включаються в бібліотеку Framework Class Library (FCL). Ця бібліотека є важливою частиною .NET і надає широкий спектр функціоналу для розробки програм. В FCL входять різноманітні класи, серед яких можна виділити Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation та інші (рис. 2.2). Ядро FCL відоме як Base Class Library (BCL).

Windows Forms в FCL дозволяє розробникам створювати графічний інтерфейс користувача для десктопних застосунків. ADO.NET забезпечує доступ до даних та можливість їх обробки в програмах. ASP.NET служить для розробки веб-застосунків та веб-сайтів. Language Integrated Query (LINQ) надає можливість виконувати запити до даних безпосередньо в коді програми. Windows Presentation Foundation (WPF) та Windows Communication Foundation (WCF) в FCL допомагають розробникам створювати сучасні інтерфейси та реалізовувати механізми комунікації між

програмами відповідно.

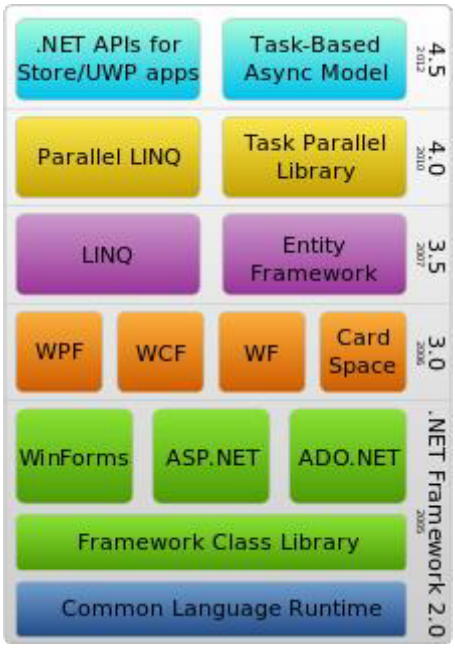

Рисунок 2.2 - Стек технологій .NET Framework [10]

Base Class Library (BCL), як ядро FCL, містить основні класи та компоненти, які використовуються для розробки будь-якого .NET-застосунку. Це включає класи для роботи зі стрічками, зчитування/запису файлів, мережевої комунікації, роботи з потоками, роботи з колекціями даних та інше. BCL надає розробникам потужні та готові до використання інструменти для реалізації різноманітних завдань у своїх додатках.

Отже, FCL, як невід'ємна складова .NET Framework, не лише розширює функціональні можливості мов програмування, але й забезпечує розробникам готовий інструментарій для реалізації різноманітних функцій та завдань у різних типах програм.

Таким чином, для розгортання будь якого додатку розробленого на платформі .Net нам необхідно буде застосувати ряд програмних рішень для повного розгортання інфраструктури, яка в подальшому буде лише самою платформою для розміщення додатку.

Кінцеве місце розташування додатку повинно підтримувати саме цю мову програмування, та надавати можливість оновлення версії без критичних витрат часу

та сил висококваліфікованих спеціалістів.

Для чіткого отримання результату, необхідно щоб середовище компілювання коду вміло чітко працювати з усіма існуючими залежностями цієї платформи, обробляти їх ще на етапі компілювання та отримати список підключенних додаком бібліотек.

Коректний результат компіляці можливо буде отримати після виконання таких декількох основних етапів які включають в себе роботу з усіма залежностями додатку, завантаження відповідних пакетів, що були описані в згаданих раніше залежностях, з використанням пакетного менеджера Nuget, обробка та компілювання початкового коду, формування артифакту(результат компіляції) додатку у відповідному форматі, який буде використаний для подальшого розміщення артифактів в середині кінцевої інфраструктури. По завершенню, будуть надані необхідні дані для створення обов'язкових конфігурацій що вимагаються додатком.

2.2 Обгрунтування вибору програмних інструментів для розгортання інфраструктури додатків на платформі .Net

Відповідно до інформації що була надана в розділі 1, про використання кожного з обраних інструментів розгортання інфраструктури, ми можемо обрати найбільш доречний у нашому випадку.

Безумовно кожний з цих інструментів може бути більш доречний в тих чи інших реаліях компанії, вимог що є присутніми до самого інструменту та об'ємів самого ентерпрайсу. Не дивлячись на це, нам необхіний найбільш універсальний інструмент робота з яким може бути в першу чергу максимально прогнозованою та оптимальною. Наш інструмент повинний бути гарно задокументований, для можливості зсилання саме на цю документацію у випадку якщо така необхідність присутня.

Інструмент повинний бути доступний для інтеграції з найпопулярнішими рішеннями розміщення інфраструктури (BareMetal,HyperV), та клауд провайдерами (Azure, AWS, Ali).

Наведемо порівняльну характеристику найбільш популярних інструментів розгортання інфраструктури, та їх найбільш ключових на мою думку особливостей.

В останні роки можливість використання хмарних інфраструктур дедалі міцніше зміцнюється у свідомості власників корпоративних бізнес-рішень (IT-рішень). Навіть якщо підприємство не робить великих ставок на обчислення в публічних хмарах, використання хмарних рішень у тій чи іншій формі є невід'ємною частиною стратегії та планів розвитку інфраструктури більшості підприємств через доступність, економічну ефективність та достатній рівень надійності. Для підприємств, які мають намір використовувати ці можливості, стають актуальними такі питання: «Яку хмарну платформу вибрати?», «Яка хмарна платформа буде найбільш економічно ефективною для завдань підприємства?» та «Як спланувати міграцію сервісів у хмару?».

Порівняємо основні інфраструктурні сервіси найпопулярніших хмарних провайдерів, таких як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). На додаток до основної інфраструктури, кожен постачальник хмарних послуг надає свої унікальні пропрієтарні пропозиції в NoSQL, Big Data, Analytics, ML та інших подібних областях, які можуть впливати на вибір хмари для підприємства залежно від його конкретних бізнес-вимог та технічних вимог.

Amazon Web Services (AWS). AWS є найбільш популярним провайдером загальнодоступних хмар із найширшим спектром продуктів, опцій для обчислень та зберігання даних, а також послуг, які клієнт може передати під керування. AWS Marketplace також є найбільшим торговим майданчиком для програм та пристроїв сторонніх виробників. Товари та послуги на такому ринку і AWS часто оновлюються, щоб постійно додавати функції продуктів для своїх клієнтів. Завдяки відгукам клієнтів нові сервіси забезпечують тісну інтеграцію з основними Web сервісами та робота клієнтів стає ще зручнішою. AWS приділяють велику увагу рекомендаціям щодо безпеки та архітектури. Корпоративні інфраструктури, такі як Well-Architected Framework та Cloud Adoption Framework були розроблені на основі досвіду роботи з великими корпоративними клієнтами. Крім основних послуг, вони також випускають нетрадиційні послуги, такі як SnowMobile (пристрій передачі даних з автомобіля),

RoboMaker (фреймворк для робототехніки) і «наземна станція як послуга» (для керованого завантаження даних із супутника). Це підтримує інтерес клієнтів та може відкрити цілі галузі. Частка ринку AWS у 51% є доказом цих тверджень. Тим не менш, AWS не найдешевша хмара на ринку.

Останнім часом AWS, схоже, переорієнтували свою увагу на гібридну хмару та оголосили про такі пропозиції, як Outposts (у партнерстві з VMware), які дозволять замовникам використовувати добре відомі сервіси AWS та API-інтерфейси у своїх приватних центрах обробки даних.

AWS - чудовий вибір як для стартапів, так і для підприємств. AWS надає широкий спектр послуг, які можуть використовувати клієнти - від невеликих веб-сервісів до масштабних міграцій центрів обробки даних у хмару. Щоб допомогти клієнтам усіх форм і розмірів розпочати роботу на платформі, AWS випустила нібито нішеві сервіси, такі як RoboMaker, з одного боку, і одночасно створила такі сервіси, як LightSail (віртуальний приватний сервер), щоб допомогти навіть найменшим робочим навантаженням з одним сервером .

В даний час AWS також пропонує найвищі на ринку варіанти здійснення обчислень та засоби зберігання даних. Широкий діапазон типів віртуальних машин (136 типів віртуальних машин у 26 сімействах віртуальних машин) дозволяє замовникам запускати практично все - від невеликих web-систем до найбільших робочих навантажень HPC (High Performance Computing) та SAP.

Для машинного навчання та штучного інтелекту AWS також надає найвищі конфігурації типів віртуальних машин із підтримкою графічного процесора GPU. Для навантажень, що вимагають оренди фізичного сервера відповідно до нормативних вимог, AWS також надає Bare-Metal-as-a-Service (апаратний сервер в оренду). Для віртуалізованих навантажень, у разі потреби, AWS надає такі функції, як «групи розміщення», щоб забезпечити виконання обчислень та зберігання на вказаному апаратному устаткуванні.

AWS сподівається, що різні розміри віртуальних машин будуть відповідати вашим вимогам.Тому він не підтримує створення розмірів користувача віртуальних машин (vCPU, RAM). На відміну від інших хмарних провайдерів (CSP), AWS надає

лише певний набір сімейств віртуальних машин із графічними процесорами. AWS не дозволяє підключати графічні процесори до будь-яких або всіх типів віртуальних машин у своєму портфелі.

Хмарне сховище даних поставляється AWS з різними варіантами, такими як динамічна зміна розміру, різні типи дисків (традиційний та SSD). На відміну від інших CSP, AWS не обмежує кількість операцій вводу-виводу за секунду (IOPS) за розміром розділу. Ви можете отримати більше IOPS за додаткову плату, навіть для невеликих дисків.

В області наданих як сервіс реляційних баз даних, що обслуговуються, AWS підтримує керовані бази даних для MySQL, PostgreSQL, MariaDB, Oracle і MS SQL в рамках своєї пропозиції RDS. Крім того, AWS мають власні бази даних, сумісні з MySQL і PostgreSQL, які можуть похвалитися продуктивністю, подібною Oracle, але доступні за невелику плату. AWS вкладають у це значні кошти, а також анонсували версії баз даних Multi-Master та Serverless.

AWS обслуговує роботу уряду США в окремих регіонах GovCloud у США. Клієнти, яким необхідно надавати послуги клієнтам у Китаї, можуть розраховувати на китайський регіон AWS, який надається у партнерстві зі сторонніми постачальниками.

Загалом AWS надає широкий спектр послуг та функцій, які підходять для значної кількості підприємств.

Microsoft Azure. Azure прийшов пізніше з помітним відставанням від AWS у наданні публічних хмарних сервісів і спочатку зосередився на пропозиціях SaaS і PaaS (програмне забезпечення як послуга та платформа як послуга), оскільки її сильні сторони лежать як у корпоративному, так і споживчому програмному забезпеченні. Спочатку Microsoft орієнтувалася на служби PaaS для Azure і були сфокусовані на існуючій базі розробників, які використовують технології Microsoft. Згодом Microsoft розширила свою увагу і на послуги Linux, і на наданні IaaS (інфраструктура як послуга). Це також відображено в ребрендингу Windows Azure на Microsoft Azure та на Microsoft Loves Linux. З часом Microsoft також зробила Azure більш зручною та вбудовала підтримку API для різних сервісів. Однак, незважаючи на широкий спектр

послуг, Microsoft суттєво відстає від AWS у корпоративному впровадженні. Великі підприємства, в яких вже існують відносини з Microsoft, залишаються значною частиною бази користувача, хоча в Azure спостерігається стійке зростання річного доходу.

Azure - це зріла хмарна платформа з широким набором функцій, яка може бути кращою платформою для клієнтів, які вже якимось чином використовують продукти Microsoft. Хоча Azure підтримує низку сервісів на основі продуктів з відкритим вихідним кодом, портфоліо Microsoft у хмарі - це те, що виділяє його серед конкурентів.

Azure надає більше 151 типів віртуальних машин і понад 26 сімейств віртуальних машин, які підтримують усі – від невеликих web-систем до навантажень HPC, Oracle та SAP.Azure має як Windows, і кілька версій Linux (RHEL, CentOS, SUSE, Ubuntu). У Azure є окреме сімейство екземплярів для роботи з машинним навчанням та штучним інтелектом (ML/AI).

Azure також був першим хмарним гравцем, що розпізнав тенденцію до створення гібридної хмари, та запропонував одне з перших гібридних хмарних рішень та хмарного центру обробки даних (стек Azure). Клієнти, яким потрібний інтерфейс Azure, але які хочуть запускати служби у власних центрах обробки даних, можуть використовувати Azure Stack. Інші хмарні гравці тільки наздоганяють Azure у цьому сенсі. Azure також надав підтримку гібридних пристроїв зберігання даних, таких як StorSimple, яка була унікальною у публічному просторі хмар.

Коли йдеться про бази даних SQL і NoSQL, в Azure є досить хороший набір сервісів. Azure забезпечує MSSQL Server та SQL Datawarehouse як сервіс. Azure також надає бази даних MySQL, PostgreSQL та MariaDB. Azure Table є керованим сховищем значень ключів, тоді як CosmosDB надає багатомодельну глобально розподілену базу даних NoSQL з декількома моделями узгодженості. Він надає API, сумісний з MongoDB, Cassandra, Gremlin (Graph) та сховищем таблиць Azure. Якщо вам потрібно запустити кілька моделей керованих даних, у тому числі моделі даних документа, графа, ключа-значення, таблиці та сімейства стовпців в одній хмарі, то Cosmos може

бути слушним варіантом. Кеш Azure для Redis доповнює пропозиції керованим кешем.

На додаток до моделі виставлення рахунків PAYG (Pay as You Grow) з кредитними картками та режимами виставлення рахунків, клієнти з існуючими корпоративними обліковими записами можуть попередньо придбати підписки Azure в рамках свого щорічного продовження. Це корисно для клієнтів, які хочуть мати нагоду заздалегідь планувати щорічні витрати на хмару. Це запобігає невизначеності та додатковим затвердженням бюджету на середину року, які зазвичай пов'язані з моделями PAYG. При цьому підприємства мають з деякою точністю визначати розмір своїх прогнозованих робочих навантажень, щоб наприкінці року не було витрачено передплачені кредити.

Мобільність ліцензій у хмарі для продуктів Microsoft також є відносно простою для клієнтів з кількома продуктами Microsoft, що працюють на місці.

Хмарна платформа Google Cloud Platform (GCP), незважаючи на запізнілий вихід і найменшу частку ринку серед постачальників загальнодоступних хмарних обчислень у порівнянні з конкурентами (поточна частка ринку близько 4%), демонструє стійке відсоткове зростання. GCP може похвалитися кількома функціями, які ставлять його попереду своїх конкурентів у певних галузях. Ще один аспект у тому, що GCP користується популярністю не лише у клієнтів, які вже є частиною екосистеми, але також і користувачів хмарних технологій, які прагнуть розширити свій ландшафт на Google у рамках стратегії використання кількох хмарних платформ.Google також почав із PaaS-сервісів, але неухильно розширює свій портфель продуктів.

Поряд з інноваційними функціями Google може похвалитися найнижчою ціною на інфраструктуру в порівнянні з іншими хмарними провайдерами. Звичайно, загальні витрати для будь-якого підприємства залежать від послуг, що використовуються, і діючих заходів з управління витратами.

Google має глобальну мережу доступу до ресурсів з низькою затримкою. Навіть з погляду клієнта мережа VPC (Virtual Private Cloud) охоплює всі регіони. Інші CSP (постачальники хмарних послуг) обмежують VPC мережі в окремому регіоні. Це

дозволяє клієнтам GCP створювати додатки, які обслуговують клієнтів по всьому світу без створення складних міжрегіональних інфраструктурних механізмів і механізмів реплікації даних.

Як докладно описано вище, кожна хмара має особливості та переваги, які відповідають конкретним потребам клієнтів. У той час як усі хмарні провайдери продовжуватимуть надавати певні загальні послуги (такі як керована база даних MySQL), кожен CSP продовжуватиме створювати унікальні, диференційовані послуги (наприклад, Aurora, Cosmos, Spanner), які спеціально призначені для вирішення специфічних потреб клієнтів. . CSP сподіваються, що це збільшить прихильність клієнтів до їх послуг та створить бар'єр для переходу до конкурента.

З погляду клієнта, ці послуги також стануть рушійною силою для прийняття мультихмарної стратегії. Наприклад, клієнт, можливо, захоче використовувати GCP для однієї програми, якій потрібні функції Spanner, тоді як він використовує AWS для своїх служб AI та Azure для певних завдань Windows.

Навіть для таких перспективних послуг, як комп'ютерний зір та розпізнавання мови, потреби клієнтів можуть підштовхнути їх до змішування та зіставлення сервісів на хмарних платформах, щоб задовольнити вимоги їх додатків. Клієнт, швидше за все, буде використовувати одну хмару як свою основну платформу, а служби інших використовуватимуться для конкретних додатків.

З точки зору виставлення рахунків, Google надає автоматичні знижки, такі як знижки на постійне використання, які знижують ціну на вимогу, якщо віртуальна машина працює певнішу кількість годин на місяць. Якщо вам потрібен найефективніший хмарний провайдер на ринку сьогодні, то GCP є гарним вибором.

Незважаючи на те, що GCP може не мати достатньої глибини можливостей деяких інших постачальників хмарних послуг, у його портфелі є кілька унікальних продуктів, і він є привабливим варіантом, будучи лідером цін на ринку.

Для зручності сприйняття набір параметрів зведено у таблицю. У ній стовбці - позначають інструмент розгортання, а стрічки – параметри та особливості взаємодії з відповідним інструментом.

На мою думку, найбільш прийнятнішими у нашому випадку інструментами

розгортання інфраструктури є Jenkins та Azure DevOps. Щоб визначитись з фаворитом – необхідно більш детально розібрати кожний пунктів що описані у таблиці 2.1.

Таблиця 2.1 – Порівняльна таблиця інструментів розгортання інфраструктури

	Jenkins	TeamCity	Azure Devops	Travis CI	Circle CI	GitLab	Go CD
Ціна	Безкоштовно	299-1999 у.о.	0-200 у.о.	69-489 у.о.	50-3150 у.о.	0-99 у.о.	Безкоштовно
Операційна система	Windows, Unix-like, macOS	Windows, Linux, macOS, Solaris, FreeBSD	Windows, Linux, MacOS, Solaris	Linux, MacOS	Linux, IOS, Andori d	Only Unix-like OS	Windows, Linux, MacOS
Хостинг	On Premise cloud	On premise	Provided by Azure Cloud/On Premise	On Premise cloud	Provided by Cloud	On Premise cloud	On Premise cloud
Підтримка контейнеризації	+	+	+	+	+	+	+
Плагіни	5	4	5	4	3	3	4
Документація та підтримка	Середня	Середньо	Добра	Погана	Добра	Добра	Середня
Складність використання	Легко	Середньо	Середньо	Легко	Легко	Легко	Легко
Ціль використання	Універсальна система	Для комерційних проектів	Універсальна система	Для невеликих проектів та стартапів	Для великих бюджетних в та швидких проектів	Для невеликих проектів з визначеною метою	Універсал ьна система

Розпочавши з Jenkins важко не відмітити те що він безкоштовний. На фоні інших інструментів це являється сильною стороною, але якщо вдатись у деталі, можна зрозуміти, що це не зовсім так. У пункті хостингу ми можемо бачити, що більшість інструментів розміщуються чи на On-Premise серверах чи в якомусь клауді. В даному випадку це означає що для іхнього функціювання та розміщення повинна бути використана якась з доступних платформ, на якій вже і відбудеться подальша робота з даним ісінтрументом. Саме тому, більш доцільніше, було б при урахуванні вартості самого інструменту, враховувати і необхідні на його функціювання витрати.

В той же момент, Azure DevOps являється більш висококласним інструментом, оскільки він може самостійно працювати в якості сервісу розгортання інфраструктури без необхідності будь-якої додаткової платформи.

2.3 Розробка UML діаграми розгортання інфраструктури на основі хмарного провайдера Azure

Для більш кращої візуалізації процесу, розгортання інфраструктури, доцільно використати діаграму видів діяльності, оскільки сам процес розгортання інфраструктури являється логічно розділений за місцем виконання задач. Мається на увазі, що якісь завдання для розгортання будуть виконуватись самим інженером, hosted агентом, чи самим Cloud провайдером що було обрано (рис. 2.3).

Hosted agent – це віртуальна чи фізична машина що бере участь у процесі розгортання інфраструктури, за рахунок якої процес розгортання і отримує основний об’єм обраховуючих здідностей. Цей агент являється проміжною ланкою між користувачем, та кінцевим місцем розгортання інфраструктури. В нашому випадку це Cloud провайдер Azure.

Процес розгортання інфраструктури було розподілено на три частини, що на рисунку відмежовані пунктиром. Розподілення виконувалось відносно завдань та середовища в якому вони виконуються.

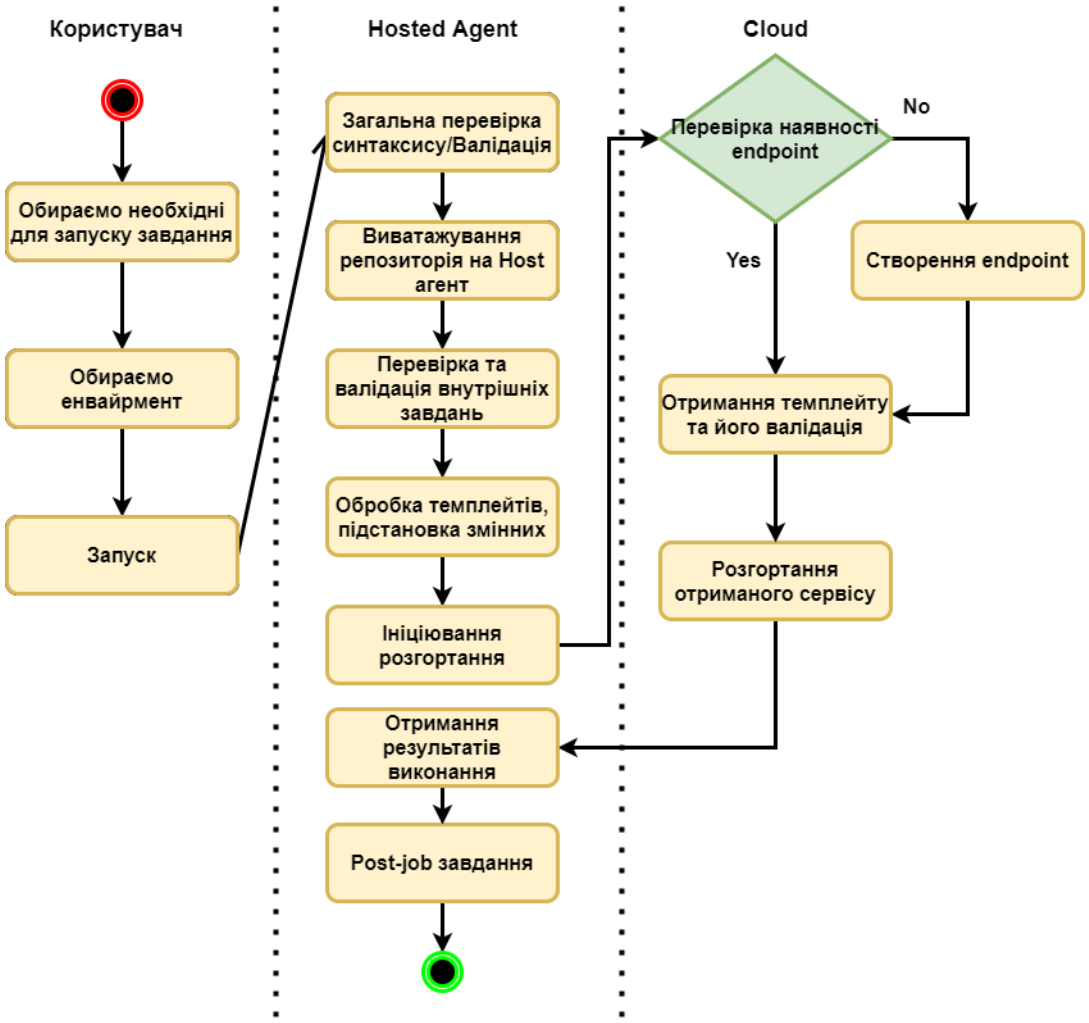

Рисунок 2.3 – UML діаграма видів діяльності

2.4 Висновок до розділу 2

В даному розділі було проаналізовано особливості роботи додатків на платформі .Net. Обгрунтовано вибір програмних інструментів для розгортання інфраструктури додатків, на основі порівняльної характеристики з програмами аналогами: Jenkins, TeamCity, Travis CI, Circle CI, Gitlab, Go. В результаті було обрано програмний інструмент Azure DevOps. Детально проаналізовано переваги та недоліки використання сервісів від хмарних провайдерів Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP). Розроблено UML-діаграму розгортання інфраструктури додатків на основі хмарного провайдера Azure.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПО РОЗГОРТАННЮ ІНФРАСТРУКТУРИ ДОДАТКІВ НА ОСНОВІ ХМАРНОГО ПРОВАЙДЕРА AZURE

3.1 Вибір мови програмування та середовища розробки

Для програмної реалізації даної внформаційної технології необхідно оцінювати подальшу можливість описання все більшої кількості елементів інфраструктури, що в свою чергу прямопропорційно приведе до збільшення часу виконання завдань. Саме тому нам необхідно враховувати усі можливі моменти що в перспективі приведуть до збільшення часу виконання з точки зору самого способу розгортання інфраструктури.

Обраний інструмент – Azure DevOps, передбачає можливість опису кінцевого результату декларативними методами з можливостю інтеграції різних інструментів. Найбільш популярні вважаються Terraform, Ansible, ARM.

Terraform є інструментом для безпечного та ефективного створення, зміни та модернізації інфраструктури. Цей інструмент здатний керувати існуючими провідними постачальниками хмарних обчислень, а також індивідуальними приватними рішеннями. Файли конфігурації описують компоненти Terraform, необхідні для розгортання одного застосунку або навіть цілого центру обробки даних.

Terraform генерує план виконання, який ретельно описує, що буде зроблено для досягнення бажаного стану. Після цього він виконує цей план, створюючи визначену конфігурацією інфраструктуру. При зміні конфігурації Terraform може автоматично визначити, що зазначено змінити, і створює відповідні плани виконання, які можна застосувати.

Інфраструктура, якою Terraform керує, охоплює компоненти низького рівня, такі як обчислювальні машини, сховища та мережі, а також включає компоненти високого рівня, такі як DNS-записи, функції SaaS і інше. Для опису конфігурацій Terraform використовує мову HCL (HashiCorp Configuration Language) [18].

Основний процес функціювання даного програмного рішення заключається в використанні та підключенні чітко заданих модулів, що розроблені компанією Hashicorp. Це в свою чергу має як свої переваги, так і недоліки.

Не є секретом, що інфраструктура подібного проекту, навіть тієї ж самої компанії матиме свої певні особливості у будь-якому випадку і це зробить її унікальною. При цьому самі рішення будуть надходити зі сторони спеціаліста що займається описом та розгортанням. Terraform вирішує саме такі питання, оскільки єдине що може змінитись при розгортанні з його використанням – порядок змінних, спосіб їх передавання, та самі їхні значення. Але у випадку якщо якийсь момент інфраструктури не був описанний розробником – це виливається у велику проблему, де решту роботи доводиться виконувати, чи альтернативними засобами, чи задопомогою імперативног методу.

Тому цей інструмент є доцільним у випадку “примітивної” інфраструктури, яка не передбачає рідкісних сервісів, та специфічної взаємодії. Хоча, з іншого боку, це може бути гарною причиною дотримуватись певних загальноприйнятих стандартів розгортання інфраструктури та мати неймовірний потенціал в майбутньому. На данний момент це проявляється в процессі розвитку такого стандарту як Terraform Cloud Adoption Framework.

Нажаль це рішення не є достатньо завершене для повноцінного впровадження.

Ansible - це інструмент автоматизації, спрямований на надання простого, але потужного засобу для управління розробочими середовищами. Зазвичай використовується IT-фахівцями для розгортання додатків, оновлень на робочих станціях та серверах, а також для надання хмарних послуг, управління конфігурацією, оркестрації внутрішніх служб та практично будь-якої задачі, яку виконує системний адміністратор щотижня або навіть щоденно. Ansible не потребує встановлення агента (крім наявності пакету Python, який зазвичай є встановленим за замовчуванням у більшості актуальних дистрибутивів ОС) і не вимагає додаткової інфраструктури безпеки, що полегшує його розгортання [15].

Оскільки Ansible спрямований на автоматизацію, кожен крок виконується на основі інструкцій. Завдяки декларативній мові, оформленій в простому форматі

YAML, легко управляти версіями компонентів і забезпечувати ідемпотентність системи. Практичний результат полягає в значному внеску у рух IaaC (Infrastructure as Code): ідея, що управління серверною та клієнтською інфраструктурою може бути розглянуто так само, як і розробка програмного забезпечення. Це включає зберігання версій перевірених скриптів у системах контролю версій, що дозволяє ефективно керувати організацією незалежно від змін персоналу.

Хоча Ansible може бути в передовому ряді автоматизації, адміністрування систем і методології DevOps, він також дуже корисний для повсякденних потреб користувачів. Ansible дозволяє налаштовувати не один комп'ютер, а потенційно цілу групу комп'ютерів одночасно, і для його використання не потрібні вміння програмування. Інструкції, написані для Ansible, легко читаються як новачками, так і досвідченими розробниками.

В Ansible існують дві категорії комп'ютерів: вузол управління та керовані вузли. Вузол управління - це комп'ютер, на якому встановлений та працює Ansible. Повинен бути принаймні один вузол управління, хоча також може існувати резервний вузол управління для забезпечення доступу до інфраструктури у разі критичних ситуацій. Керований вузол - це будь-який пристрій, яким керує вузол управління.

Ansible працює, підключаючись до вузлів (клієнтів, серверів чи будь-чого, що ви налаштовуєте) в мережі, і надсилає до цього вузла невелику програму, яку називають модулем Ansible. Ansible виконує ці модулі через SSH та видаляє їх після завершення. Єдина вимога для цієї взаємодії - щоб ваш вузол управління Ansible мав доступ до керованих вузлів. SSH-ключі - найпоширеніший спосіб надання доступу, але підтримуються інші форми аутентифікації.

Термін "модулі" може звучати складно, але Ansible бере на себе більшу частину цієї складності, а не користувач. Модуль Ansible описується як модель бажаного стану системи, тобто кожен модуль визначає, що має бути застосовано до будь-якого керованого вузла. Наприклад, якщо системний адміністратор вирішив, що на всіх робочих станціях організації має бути встановлена версія LibreOffice версії X.Z, то перш за все слід перевірити, чи кожен вузол має LibreOffice X.Z встановлено. Якщо

Ansible виявить керований вузол із встановленою версією LibreOffice X.Y, він визначить тип дистрибутиву та версію операційної системи і, після цього, запустить необхідну процедуру для оновлення LibreOffice до версії X.Z. Таким чином, кожну робочу станцію в організації можна оновлювати протягом ночі за допомогою програмного забезпечення, яке підтримується IT-відділом.

Однак управління інфраструктурою - це не лише перевірка версій програмного забезпечення. Коли мова йде про використання Ansible, зазвичай мають на увазі використання модулів Ansible, оскільки це частини, що виконують конкретні завдання. Якщо вам потрібно щось автоматизувати на кількох комп'ютерах, варто розглянути існуючі модулі Ansible, щоб знайти той, який вирішує завдання, що вам потрібно виконати, і потім встановити його в Ansible для подальшої настройки та виклику цього модуля. Також є можливість написати власні спеціалізовані модулі для виконання специфічних завдань. Крім того, завжди є можливість запропонувати самостійно розроблений модуль для включення до офіційного переліку модулів, щоб інші користувачі могли вільно його використовувати.

Хоча модулі використовуються для виконання завдань, їх використання здійснюється через так звані "книги ігор" (playbooks) Ansible. Playbook - це файл конфігурації, написаний у форматі YAML, який містить інструкції щодо того, що потрібно зробити, щоб привести керований вузол у потрібний стан. Книги ігор повинні бути простими, читабельними для людей та незалежними сценаріями. Вони є універсальними для різних систем, що означає, що їх можна запускати в системі в будь-який час без негативного впливу. Якщо playbook запускається в системі, яка вже належним чином налаштована і перебуває у бажаному стані, то ця система все одно має бути належним чином налаштована після виконання сценарію.

Сценарій (playbook) може бути дуже простим, наприклад, встановлення привілейованого користувача (root) HTTP-сервера Apache на будь-якому вузлі групи веб-серверів IT-відділу. Але в той же час він може бути дуже складним з умовами та змінними. Однак, оскільки більшість роботи виконується модулями Ansible, playbooks зазвичай залишаються короткими, читабельними та зрозумілими, хоча вони можуть охоплювати цілі групи керованих вузлів.

ARM – розшифровується як Azure Resource Manager. Це служба управління всіма створюваними в Azure ресурсами. У першій версії Microsoft Azure ARM був відсутній, і при створенні служб для додатка розробники стикалися з проблемою: всі служби створювалися у вигляді одиночних елементів, без яких-небудь перехресних посилань або залежностей. Управляти додатком, що складається з безлічі ресурсів, було непросто [11].

Resource Manager Azure забезпечує рівень управління для створення, оновлення та видалення ресурсів в обліковому записі. Його функції управління, такі як управління доступом, блокування та додавання тегів, дозволяють захищати і впорядковувати ресурси після розгортання.

Коли користувач надсилає запит з будь-якого засобу Azure, API або пакету SDK, він спрямовує його до Resource Manager. Resource Manager виконує аутентифікацію та авторизацію запиту, а потім передає його службі Azure, яка виконує запитану дію. Оскільки всі запити обробляються через один API, результати та можливості будуть узгоджені в різних засобах.

Найголовнішою причиною актуальності ARM являється його гнучкість, можливість описання будь якого сервісу самостійно, виконуючи ті маніпуляції з ресурсом які необхідні. Цей ресурс є основою для багатьох інструментів взаємодії Azure та описаний у json форматі.

Чіткий процесс, що був налаштований компанією розробником, забезбечує високу швидкість, структурованість як вхідних так і вихідних даних. Можливо змінювати шаблон розгортання залежно від заданих умов та отриманих вхідних даних, що дає змогу описати універсальний шаблон для певного вида ресурсів. При цьому конкурентні продукти позбавлені такої можливості, щей на настільки високому рівні як це представлено в ARM.

У вигляді таблиці порівняємо час розгортання, у секундаїх, найбільш популярних ресурсів, таких як: Web App, Service Plan, Storage Account, Key Vault, за допомогою усіх трьох наведенних вище інструментів.

Як можна побачити з таблиці 3.1, у переважній більшості випадків ARM є більш швидшим інструментом розгортання ніж наведені аналоги.

Таблиця 3.1 – Порівняльна таблиця інструментів розгортання інфраструктури

	Ansible	ARM	Terraform
Web App	79 с.	67 с.	74 с.
Service Plan	46 с.	43 с.	44 с.
Storage Account	52 с.	38 с.	46 с.
Key Vault	74 с.	64 с.	60 с.

3.2 Розробка структури інформаційної технології по розгортанню інфраструктури на основі хмарного провайдера Azure

Під час управління процесом розгортання інфраструткури потрібно враховувати що кінцевими користувачами модуля будуть інженери чи спеціалісти що мають мінімальний набір знань відносно загальних принципів процесу розгортання інфраструктури та взаємодії з хмарним провайдером Azure.

Загальний процес розгортання інфраструктури включає в себе велику кількість додаткових дій, які виконуються вже на стороні провайдера, без можливості втручання. Проте, усі інші дії є результатом описанних в темлпейтах задач чи результатом їхньої роботи, відповідно до заданих даних.

Цей сценарій включає в себе вибір кінцевого environment (середовище) яке буде розгорнуте в результаті роботи. До модуля розгортання інфраструктури будуть передані різні вхідні дані, відносно попередньо обраного середовища. Після чого, модуль розгортання інфраструктури працює з отриманими змінними та парсить їх у параметри json формату. В результаті, отримані параметри будуть використані для заповнення json темплейтів відповідного сервісу.

Структура інформаційної технології по розгортанню інфраструктури на основі хмарного провайдера Azure показано на рисунку 3.2

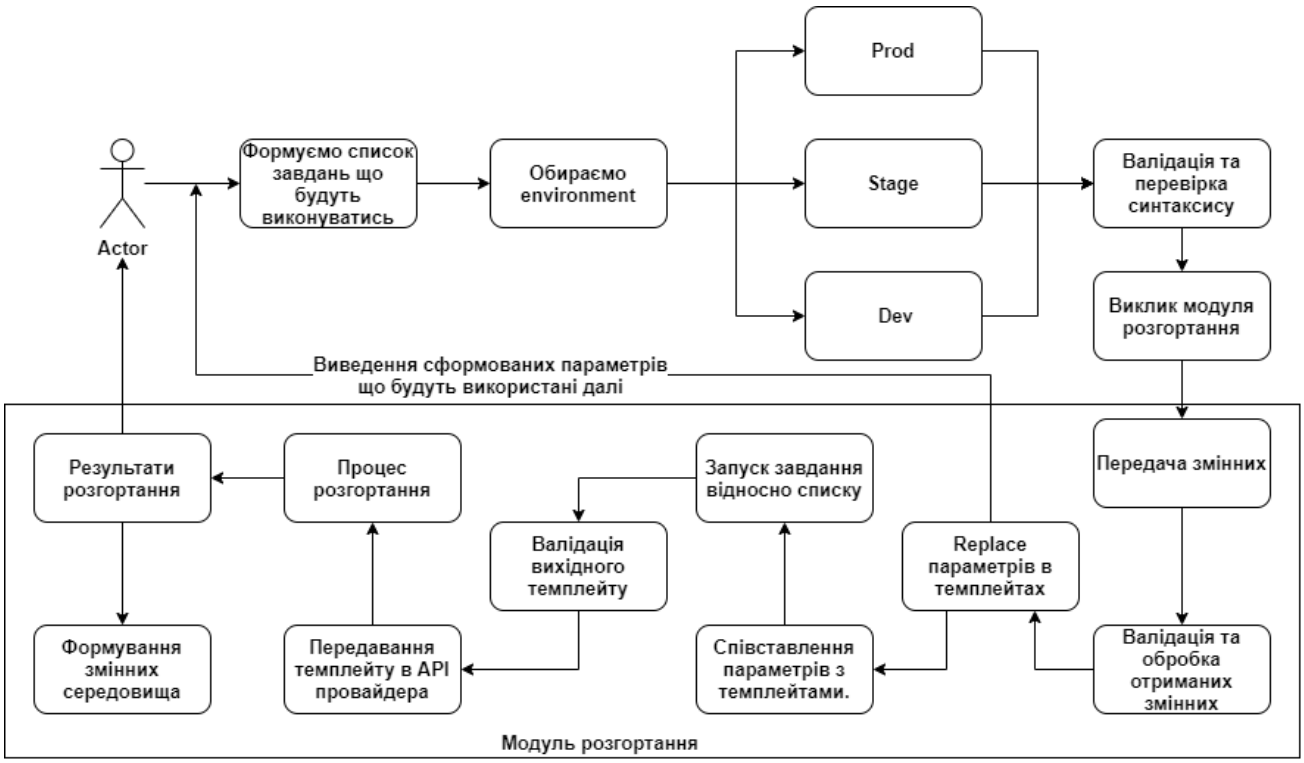

Рисунок 3.2 – Схема процесу розгортання інфраструктури

Після отримання готового для використання темплейту – виконується його валідація та передається до Azure API. Таким чином, ініціюється процес розгортання описанного в темплейті сервісу та обробка результатів. Результати можуть бути збережені до змінних середовища, які, при необхідності, можливо викликати у подальших завданнях. Таким чином зробивши посилання між елементами різних сервісів.

3.3 Програмна реалізація інформаційної технології з розгортання інфраструктури на основі хмарного провайдера Azure

Для взаємодії описанного процесу з хмарним провайдером Azure, необхідно підготувати Azure Service Connection (рис. 3.3).

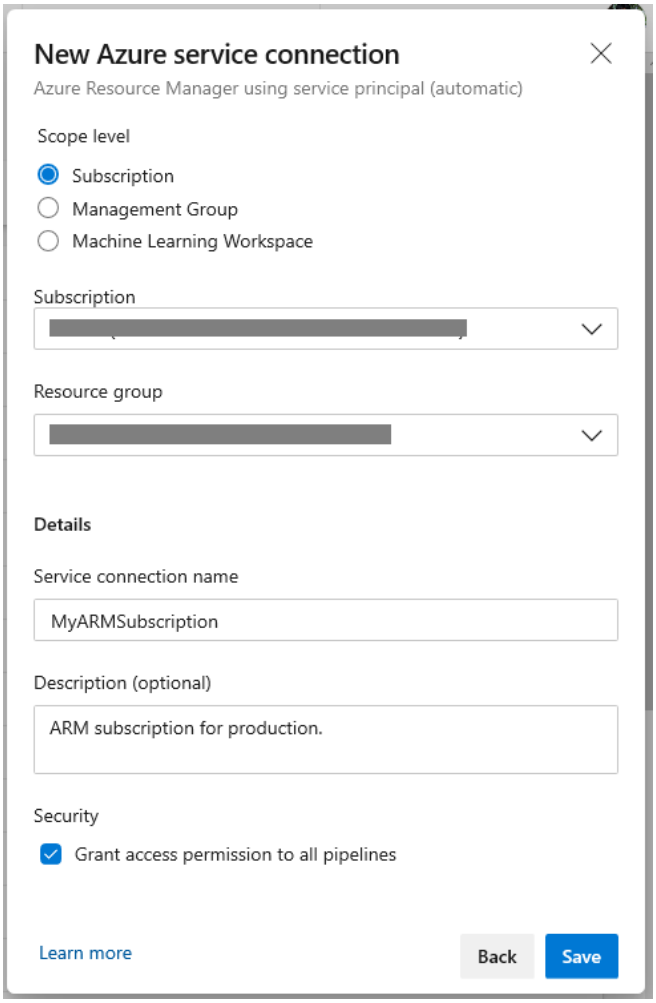

Рисунок 3.3 – Створення Azure Service Connection

Для запуску модуля розгортання нам необхідно ініціювати процес запуску pipeline що входить до комплексу Azure DevOps. Таким чином буде передано назву середовища з яким буде працювати модуль розгортання (рис. 3.4.).

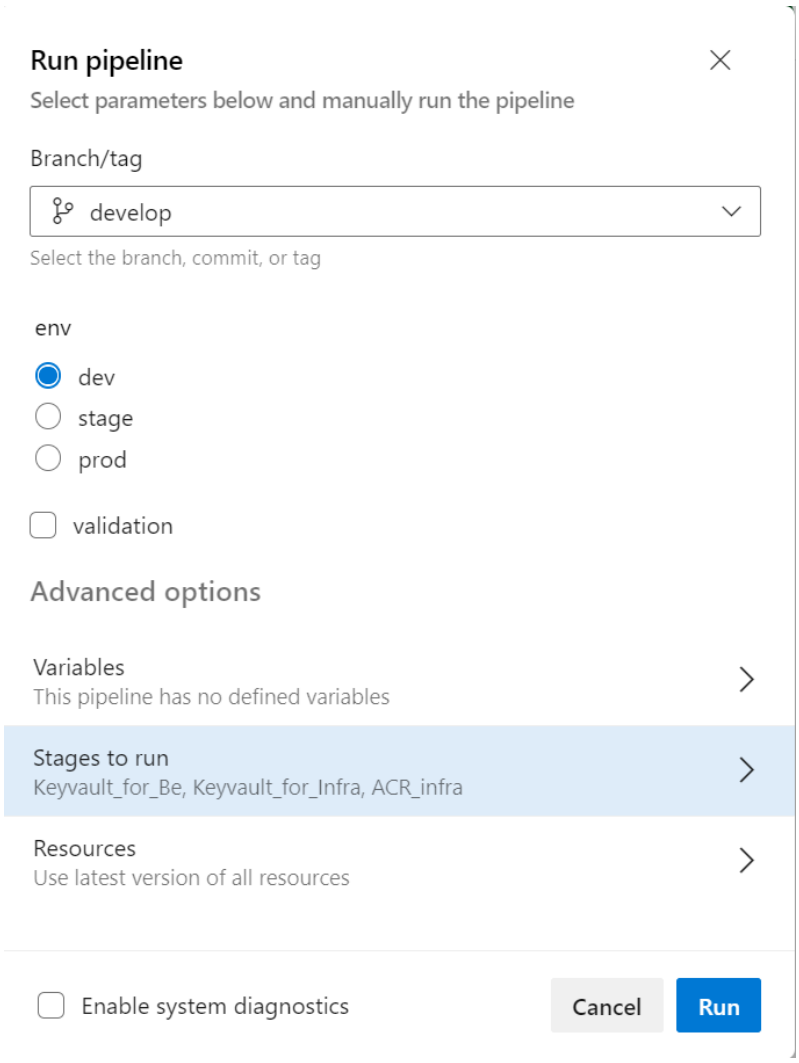

Рисунок 3.4 – Запуск pipeline

Таким чиному буде розпочато роботу усіх кроків описаних у даному pipeline. У випадку необхідності запуску лише деяких з цих задач – необхідно сформувати список завдань для поточного запуску. Цей список буде дійсним лише для поточного запуску (рис. 3.5.).

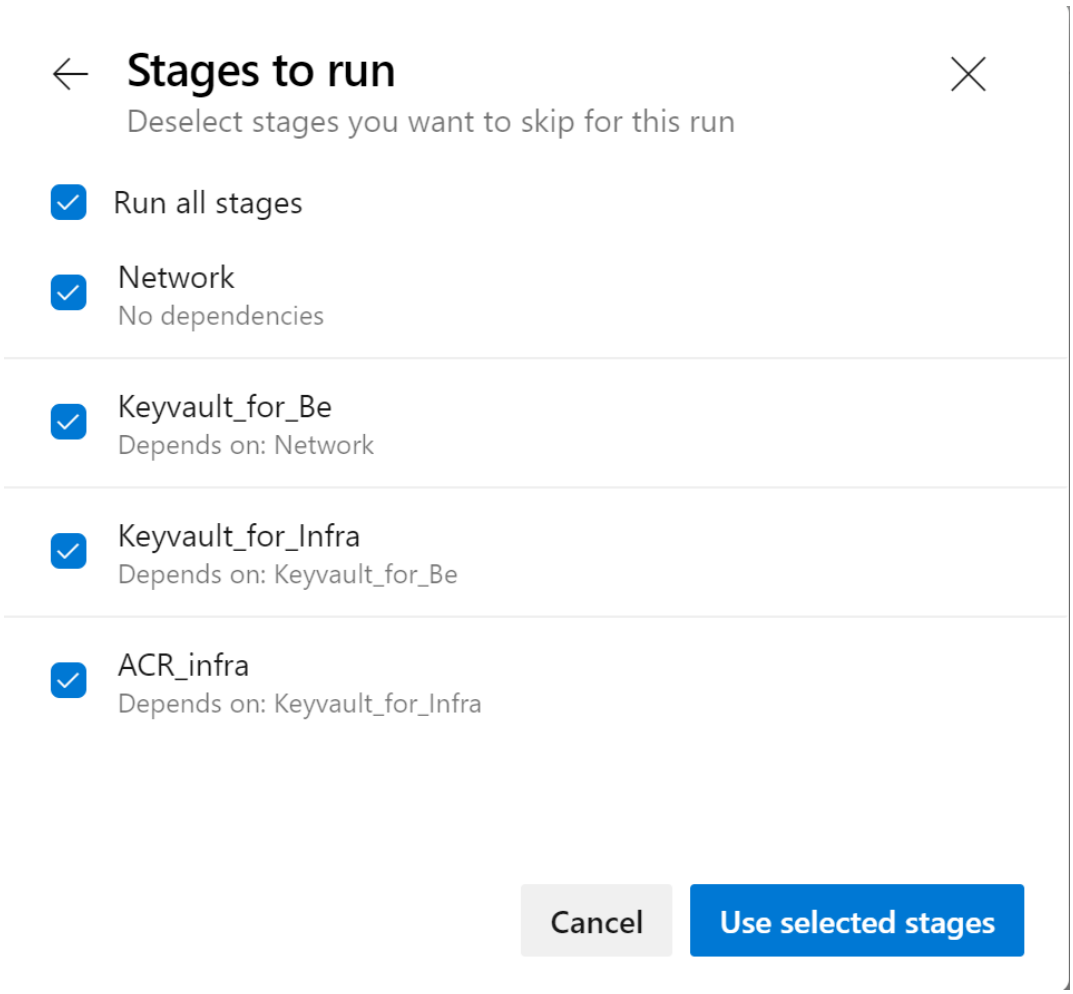

Рисунок 3.5 – Формування списку задач для виконання

Для запуску кожної задачі їй необхідно передати відповідні змінні, які будуть відповідати середовищу та в подальшому будуть конвертовані у json формат для корректної роботи пайплайна.

Приклад початкових змінних що описані з використанням синтаксису yaml:

```
Variables:
- name: parameters.privateEndpointName.value
  value: $(BusinessUnit)-be-$(Env)-endpoint-$(LocationShort)
- name: parameters.privateLinkServiceRG.value
  value: $(BusinessUnit)-rg-be-$(LocationShort)
- name: parameters.privateLinkServiceName.value
  value: $(parameters.kvName.value)
```

```
- name: parameters.privateLinkServiceType.value
  value: Microsoft.KeyVault/vaults
- name: parameters.groupId.value
  value: "vault"
- name: parameters.dnsZoneRG.value
  value: $(BusinessUnit)-rg-infra-$(LocationShort)
```

Після конвертації у json вони матимуть наступний вигляд:

```
{
   "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
   "contentVersion": "1.0.0.0",
   "parameters": {
      "privateEndpointName": {
         "value": "test-infra-dev-endpoint-we"
      },
      "privateLinkServiceName": {
         "value": "test-infra-dev-kv-we"
      },
      "privateLinkServiceType": {
         "value": "Microsoft.KeyVault/vaults"
      },
      "privateLinkServiceRG": {
         "value": ""
      },
      "vnetName": {
         "value": "test-infra-dev-vnet-we"
      },
      "subnetName": {
         "value": "test-infra-dev-subnet-we"
```

```
    },
    "groupId": {
      "value": "vault"
    },
    "dnsZoneName": {
      "value": "privatelink.test-infra-dev.azure-automation.com"
    },
    "dnsZoneRG": {
      "value": "test-rg-infra-we"
    },
    "dnsZoneSubscription": {
      "value": ""
    }
  }
```

3.4 Тестування та аналіз результатів роботи програмного забезпечення по розгортанню інфраструктури на основі хмарного провайдера Azure

Програмне забезпечення по розгортанню інфраструктури входить до комплексу Azure DevOps, що дозволяє взаємодіяти з системою контролю версій та виконувати маніпуляції з кодом використовуючи Web-версію. Для цього необхідно через Web–браузер перейти і авторизуватися у системі контролю версій, що зберігає останню версію коду. В більшості випадків ця система розміщена за посиланням https://dev.azure.com/{НАЗВА ОРГАНІЗАЦІЇ}.

Запуск вже налаштованої програми для розгортання не вимагає від користувача додаткової модифікації коду чи внесення змін в ручному режимі. Тому, для запуску програми достатньо буде авторизоватись у інтерфейсі, перейти у розділ Pipelines та запустити необхідній pipeline (рис. 3.6).

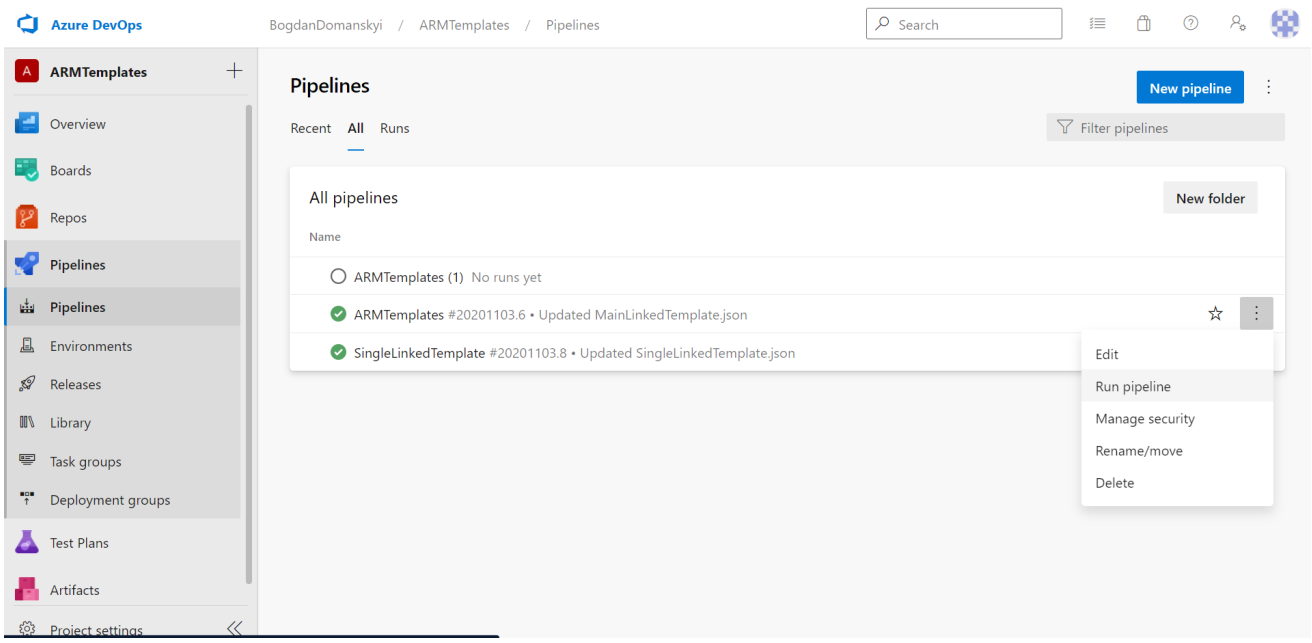

Рисунок 3.6 – Запуск pipeline через інтерфейс Azure DevOps

На рисунку 3.7. та 3.8 продемонстровано результати виконання pipeline та логів виконання що були сформовані. Їх можна переглянути натиснувши на необхідний запуск, що був ініційований.

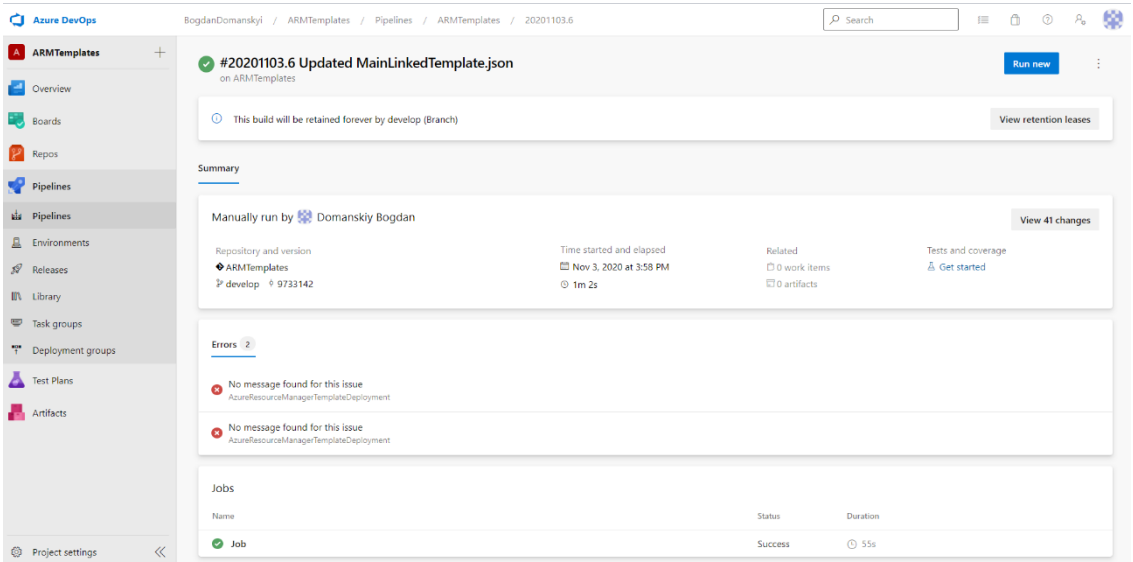

Рисунок 3.7 – Результати запуску pipeline через інтерфейс Azure DevOps

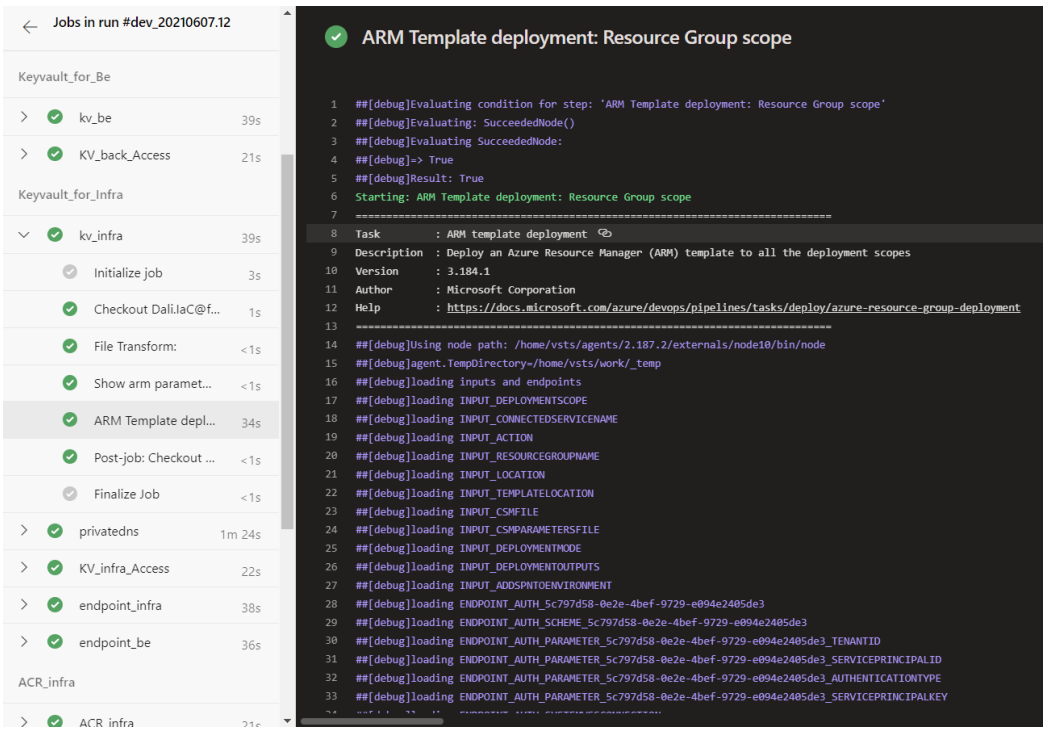

Рисунок 3.8 – Логи запуску pipeline через інтерфейс Azure DevOps

Результатом успішного виконання програмного забезпечення розгортання інфраструктури є нові сервіси які можна переглянути за посиланням https://portal.azure.com/ (рис. 3.9).

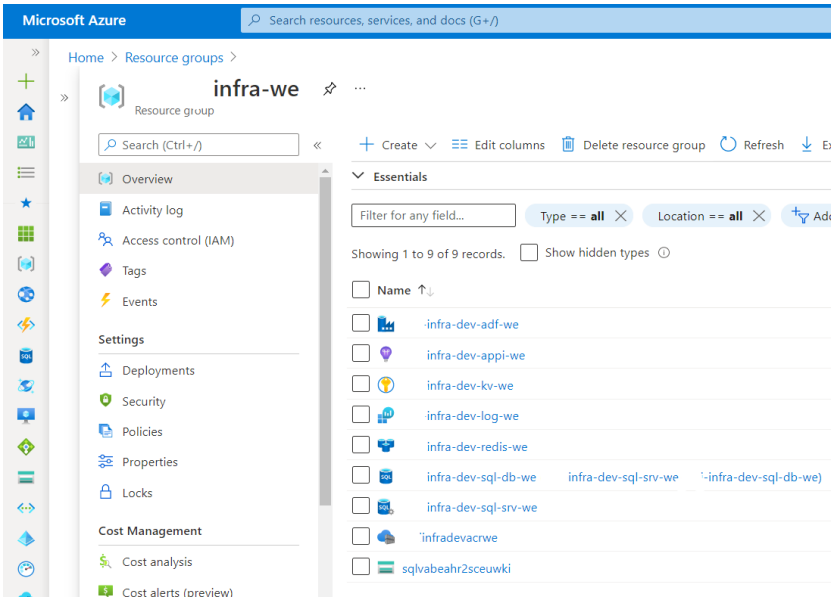

Рисунок 3.9 – Результат роботи програмного забезпечення по розгортанню інфраструктури на основі хмарного провайдера Azure

Отже, програмне забезпечення працює вірно, під час тестування помилок не виявлено.

Розроблене програмне забезпечення було протестовано серед користувачів в галузі ІТ. Для тестування на достовірність правдивості відповідей було обрано десять спеціалістів, що працюють в галузі розробки додатків на платформі .Net. Оцінка роботи виставлялась від 1 до 10, в залежності від того була дана програма корисною чи ні. Результати тестування наведені в таблиці 3.2

Таблиця 3.2 – Результати тестування інформаційної технології по розгортанню інфраструктури на снові хмарного провайдера Azure

Спеціаліст	1	2	3	4	5	6	7	8	9	10
Оцінка	9	8	10	10	8	9	8	9	9	10

Проаналізувавши дані, маємо загальну оцінку 90 із 100. Отже, можна дійти висновку що якість роботи інформаційної технології по розгортанню інфраструктури на снові хмарного провайдера Azure складає 90%. Це свідчить про досить високий рівень підвищення якості процесу розгортання інфраструктури додатків на платформі .Net.

3.5 Висновки до розділу 3

В даному розділі проаналізовано вибір мови програмування та середовища розробки інформаційної технології по розгортанню інфраструктури на снові хмарного провайдера Azure. Здійснено програмну реалізацію розробленої інформаційної технології у вигляді додатку. Проведення тестування розробленого програмного забезпечення на правиьлність функціонування всіх його складових.

При тестуванні створеної програми спеціалістами в галузі розробки додатків на платформі .Net зроблено висновок, що якість роботи інформаційної технології по розгортанню інфраструктури на снові хмарного провайдера Azure складає 90%. Це свідчить про досить високий рівень підвищення якості процесу розгортання інфраструктури додатків на платформі .Net.

4 ЕКОНОМІЧНА ЧАСТИНА

Ефективне впровадження науково-технічної розробки можливе лише за умови, що вона відповідає сучасним вимогам науково-технічного прогресу і враховує економічні аспекти. Надання оцінки економічної ефективності отриманих результатів науково-дослідної роботи є ключовою частиною цього процесу.

Магістерська робота, присвячена розробці та дослідженню "Інформаційної технології розгортання інфраструктури на основі хмарного провайдера Azure", віднесена до науково-технічних робіт, спрямованих на введення на ринок. Рішення про комерціалізацію розробки може бути прийняте протягом самої роботи, створюючи можливість виведення її на ринок. Цей напрямок розглядається як пріоритетний, оскільки розроблені результати можуть бути корисними для різних зацікавлених сторін і приносити економічні вигоди. Проте для успішного втілення цього процесу важливо знайти зацікавленого інвестора, який був би зацікавлений у реалізації цього проекту, і переконати його в обґрунтованості таких інвестицій.

Для цього визначені наступні етапи виконання робіт:

1. Проведено комерційний аудит науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу.

2. Розраховані витрати на реалізацію науково-технічної розробки.

3. Проведено розрахунок економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтовано економічну доцільність комерціалізації для інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure" є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в таблиці 4.1 [19].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	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 працездатність 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
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	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
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

Продовження таблиці 4.1.

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
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 до 5-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	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 повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки зведені до таблиці 4.2. Для опитування було залучені три експерти: Іванов О.І. (Lead DevOps. EPAM), Щербань В.С.(Senior DevOps. EPAM), Борка О.Ю. (Senior DevOps. EPAM).

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Іванов О. І.	Щербань В. С.	Борка О.Ю.
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	2	2	2
4. Ринкові переваги (технічні властивості)	4	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	2	3	3
7. Ринкові перспективи (конкуренція)	2	3	3
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	2	2	2
Сума балів	$СБ_1$=31	$СБ_2$=34	$СБ_3$=34
Середньоарифметична сума балів $СБ_c$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{31+34+34}{3} = 33$		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в таблиці 4.3 [19].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure" становить 33 бали, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

Результати магістерської кваліфікаційної роботи будуть цікаві компаніям/проектам що займаються розробкою додатків, що плануються розміщуватись на основі хмарного провайдера Azure.

4.2 Визначення рівня конкурентоспроможності розробки

Прямих аналогів нашої розробки немає, так як він може використовувати ті чи інші платформи на певних етапах роботи, але процес є уніфікованим під вимоги поставлені керівництвом або специфікою розробки програмного забезпечення. Для відповіді буду описувати аналог платформи для роз обробки подібного процесу - Jenkins.

Основними недоліками аналога є: високі технічні вимоги до інженера що налаштовує даний процес в середовищі Jenkins; потреба постійного оновлення платформи, що вимагає відповідних технічних знань та високу ймовірність недоступності платформи, протягом виконання таких робіт; платформа аналог потребує попереднього розміщення Hosted servers (Nodes) та подальшого підключення до основної платформи (Jenkins Master Node). Таким чином постає проблема курячого яйця.

У розробці дана проблема вирішується, що дана платформа надає вже завчасно налаштовані Hosted Agents, та решти взаємодії з цільовою платформою виконується в розробленому процесі.

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Одиничний параметричний індекс розраховуємо за формулою [19]:

$$q_i = \frac{P_i}{P_{базі}}. \quad (4.1)$$

де q_i – одиничний параметричний індекс, розрахований за i-м параметром;

P_i – значення i-го параметра виробу;

$P_{базі}$ – аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Загальні технічні та економічні характеристики розробки представлено в таблиці 4.4.

Таблиця 4.4 – Основні техніко-економічні показники аналога та розробки, що проектується

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Можлива недоступність платформи, %	10	0.99	10,1	30%
Напрацювання на відмову, год	3000	5000	1,7	40%
Оцінка кінцевої простоти використання, (%задач не вимагає спеціальних знань)	60	90	1,5	30%

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 – пристрій відповідає нормам і стандартам; 0 – не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховуємо як добуток частинних показників за кожним параметром за формулою [19]:

$$I_{НП} = \prod_{i=1}^{n} q_i \,, \tag{4.2}$$

де $I_{нп}$ – загальний показник конкурентоспроможності за нормативними параметрами;

q_i – одиничний (частинний) показник за i-м нормативним параметром;

n – кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра [19]:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \,, \tag{4.3}$$

де $I_{тп}$ – груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

q_i – одиничний параметричний показник i-го параметра;

α_i – вагомість i-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 4.4.

$$I_{тп} = 10{,}1 \cdot 0{,}3 + 1{,}7 \cdot 0{,}4 + 1{,}5 \cdot 0{,}3 = 4{,}16$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою [19]:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i , \qquad (4.4)$$

де $I_{ЕП}$ – груповий параметричний індекс за економічними показниками;

q_i – економічний параметр i-го виду;

β_i – частка i-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

m – кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$$I_{ЕП}=0{,}86 \cdot 0{,}5 + 0{,}9 \cdot 0{,}5 = 0{,}88.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою [19]:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}}, \qquad (4.5)$$

$$K_{ІНТ} = 1 \cdot 4{,}16 / 0{,}88 = 2{,}45.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему "Інформаційна технологія з розгортання інфраструктури на основі хмарного

провайдера Azure", під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

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

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [19]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.6)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 15000 \cdot 10 / 21 = 6818 \; грн.$$

Проведені розрахунки зведемо до таблиці 4.5.

Таблиця 4.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	15000	681,8	10	6818
Інженер-програміст	12000	545,5	55	30000
Всього				36818

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.7)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (36818) · 11 / 100% = 4050 грн.

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.8)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (36818+4050) · 22 / 100% = 8991 грн.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure".

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.9)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 … 1,15*);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір А 4	165	1	165
Ручка	14	1	14
Диск оптичний CD	15	1	15
Flesh-пам'ять	360	1	360
Всього			554
З врахуванням коефіцієнта транспортування			609,4

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.і}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 228 \cdot 2 \cdot 1,11 = 501,6$ грн.

Отримані результати зведемо до таблиці 4.7.

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Підписка Azure DevOps	2	228	501,6
Всього			501,6

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (35000 \cdot 1) / (2 \cdot 12) = 1458,33$ грн.

Проведені розрахунки зведемо до таблиці 4.8.

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	35000	2	1	1458,33
Приміщення лабораторії	270000	20	1	1125,00
Всього				2583,33

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,5$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,25 \cdot 295,0 \cdot 7,5 \cdot 0,5 / 0,8 = 345,7$ грн.

До статті «Службові відрядження» дослідної роботи на тему «Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих

розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.13)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (36818) · 20 / 100% = 7363,64 грн.

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

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

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.14)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (36818) · 50 / 100% = 18409,09 грн.

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (4.15)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (36818) · 100 / 100% = 36818,18 грн.

Витрати на проведення науково-дослідної роботи на тему "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure". розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \qquad (4.16)$$

$В_{заг}$=36818+4050+8991+609,4+501,6+2583,33+345,7+7363,64+18409,09+
+36818,18 = 116490,13 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \qquad (4.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ = 116490,13 / 0,7 = 166414,47 грн.

4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure" передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 27000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 1000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [19]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda=0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho=40\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta=18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(1·1000+2700·500)·0,83·0,4·(1-0,18/100%)=2391741,8 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(1·1000+2700·(500+400))·0,83·0,4·(1-0,18/100%)=4305827,8 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(1·1000+2700·(500+400+300))·0,83·0,4·(1-0,18/100%)=5740770,4 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП=\sum_{i=1}^{T}\frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.19)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=18%;

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

ПП =2391741,8 /(1+0,18)^1+4305827,8 /(1+0,18)^2+5740770,4 /(1+0,18)^3=

= 8320933,59 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{4.20}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 166414,47 грн.

$PV = k_{інв} \cdot ЗВ$= 2 · 166414,47 = 332828,94 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{4.21}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 8320933,59 грн;

PV – теперішня вартість початкових інвестицій, 332828,94 грн.

$E_{абс} = ПП - PV$ = 8320933,59 - 332828,94 = 7988104,66 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.22}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{в} = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+7988104,66 /332828,94)^{1/3} - 1 = 2,66.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{4.23}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 2,66 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія онтологічного моделювання бази знань з організації бібліотеки» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}}, \tag{4.24}$$

де $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 2,66 = 0,4 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.5 Висновок до розділу 4

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure" становить 33 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,45 рази.

Також термін окупності становить близько 5 місяців, що менше 3-х років, і це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure".

ВИСНОВКИ

В магістерській кваліфікаційній роботі розроблено інформаційну технологію з розгортання інфраструктури додатків на основі хмарного провайдера Azure.

В роботі проведено аналіз технологій розгортання інфраструктури додатків на платформі .Net. Виконано постановку задач дослідження до створюваної інформаційної технології по розгортанню інфраструктури для додатків на платформі .Net на основі хмарного провайдера Azure. Проаналізовано особливості роботи додатків на платформі .Net. Обгрунтовано вибір програмних інструментів для розгортання інфраструктури додатків, на основі порівняльної характеристики з програмами аналогами: Jenkins, TeamCity, Travis CI, Circle CI, Gitlab, Go. В результаті було обрано програмний інструмент Azure DevOps. Детально проаналізовано переваги та недоліки використання сервісів від хмарних провайдерів Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP). Описано стадії розробки інформаційної технології з розгортання інфраструктури додатків на основі хмарного провайдера Azure. Розроблено UML-діаграму розгортання інфраструктури додатків на основі хмарного провайдера Azure. Розроблено структуру інформаційної технології з розгортання інфраструктури додатків на основі хмарного провайдера Azure.

Здійснено програмну реалізацію розробленої інформаційної технології у вигляді додатку. При тестуванні створеної програми спеціалістами в галузі розробки додатків на платформі .Net зроблено висновок, що якість роботи інформаційної технології з розгортання інфраструктури на снові хмарного провайдера Azure складає 90%. Це свідчить про досить високий рівень підвищення якості процесу розгортання інфраструктури додатків на платформі .Net.

Згідно проведених економічних розрахунків виявлено, що рівень комерційного потенціалу розробки за темою "Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure" становить 33 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки вище середнього.

При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,45 рази.

В роботі проведено оцінювання комерційного потенціалу розробки інформаційної технології з розгортання інфраструктури на основі хмарного провайдера Azure, спрогнозовано витрати на виконання наукової роботи та впровадження результатів, які склали 166414,47 грн, розраховано період окупності – 0,4 року.Термін окупності становить близько 5 місяців, що менше 3-х років, і це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже, всі поставленні завдання виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. С.В. Барабан, Б.В. Доманський. Розгортання інфраструктури додатків .NET на основі хмарного провайдера Azure Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» − [Електронний ресурс]. – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/19669/16302

2. IT-інфраструктура. Чим Вона допомагає в роботі підприємства [Електронний ресурс] – Режим доступу: https://indevlab.com/uk/blog-ua/it-infrastruktura-yak-rozibratisya-v-tsomu-skladnomu-ponyatti-i-chim-vona-dopomagaye-v-roboti-pidpriyemstva

3. Public cloud infrastructure [Електронний ресурс] – Режим доступу: https://www.statista.com/statistics/505251/worldwide-infrastructure-as-a-service-revenue/

4. On-Premises vs Cloud Computing. [Електронний ресурс] – Режим доступу: https://www.intellias.com/cloud-computing-vs-on-premises-comparison-guide/

5. What is Right for Your Business? [Електронний ресурс] – Режим доступу: https://phoenixnap.com/blog/on-premise-vs-cloud.

6. Infrastructure as code. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Infrastructure_as_code

7. Джастін Гарісон, Кріс Нова Cloud Native – O'Reilly Media, Inc. 2017. С.195-197.

8. NET Framework [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/.NET_Framework#.D0.90.D1.80.D1.85.D0.B8.D1.82.D0.B5.D0.BA.D1.82.D1.83.D1.80.D0.B0_.NET.

9. What are ARM templates? [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/ru-ru/azure/azure-resource-manager/templates/overview

10. Amazon Web Services [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/devops/continuous-delivery/

11. CI – wikipedia. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Неперервна_інтеграція

12. CD - wikipedia. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Безперервна_доставка

13. Rafal Leszko Continuous Delivery with Docker and Jenkins: Delivering software at scale: 2018. 140 с.

14. Jenkins-Docker Continuous Integration & Continuous Deployment Pipeline - medium. [Електронний ресурс] – Режим доступу: https://medium.com/prakashkumar0301/docker-jenkins-cicd-pipeline-dd54854125f3

15. OVERVIEW How Ansible Works- medium. [Електронний ресурс] – Режим доступу: https://www.ansible.com/overview/how-ansible-works

16. What is Terraform? [Електронний ресурс] – Режим доступу: https://cloudacademy.com/course/managing-infrastructure-with-terraform/what-terraform/

17. Mono. Mono is a cross platform, open source .NET development framework. [Електронний ресурс] – Режим доступу: http://www.mono-project.com/Main_Page.

18. Troelsen, A., Japikse, P. (2017). Building and Configuring Class Libraries. In: Pro C# 7. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3018-3_14

19. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

20. Crane, Richard; Resnick, Steve; Bowen, Chris. Essential Windows Communication Foundation (WCF): For .NET Framework 3.5 (англ.). Pearson Education. ISBN 978-0-13-270160-0.

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Інформаційна технологія з розгортання інфраструктури на основі хмарного провайдера Azure

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 87,9% Схожість 12,1%

Аналіз звіту подібності (відмітити потрібне):

✓ 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 виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 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.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Доманський Б.В.

Керівник роботи Барабан С.В.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
parameters:
  parametersFile: ''
  templateFile: ''
  armFolder: ''
  validate: False

steps:
- task: FileTransform@1
  displayName: 'File Transform: '
  inputs:
    folderPath: ${{parameters.armFolder}}
    fileType: json
    targetFiles: '**\${{parameters.parametersFile}}'
- script: cat $(System.DefaultWorkingDirectory)/${{parameters.armFolder}}/${{parameters.parametersFile}}
  displayName: Show arm parameters
- task: AzureResourceManagerTemplateDeployment@3
  displayName: 'ARM Template deployment: Resource Group scope'
  inputs:
    azureResourceManagerConnection: $(serviceConnectionName)
    resourceGroupName: '$(ResourceGroupName)'
    location: '$(location)'
    templateLocation: 'Linked artifact'
    csmFile: '$(System.DefaultWorkingDirectory)/${{parameters.armFolder}}/${{parameters.templateFile}}'
    csmParametersFile: '$(System.DefaultWorkingDirectory)/${{parameters.armFolder}}/${{parameters.parametersFile}}'
    ${{ if eq(parameters.validate, true) }}:
      deploymentMode: 'Validation'
    ${{ if eq(variables.deployOutput, true) }}:
    deploymentOutputs: 'templateDeployOutput'


name: ${{parameters.env}}_$(date:yyyyMMdd)$(rev:.r)
```

```
trigger: none

parameters:
  - name: env
    type: string
    default: dev
    values:
      - dev
      - stage
      - prod
  - name: validation
    type: boolean
    default: False
  - name: endpoint_integration
    displayName: Integrate infrastructure with private endpoint
    type: boolean
    default: True
  - name: Network_deployment
    displayName: Redeploy private network
    type: boolean
    default: False

variables:
  - name: endpoint_integration
    value: ${{parameters.endpoint_integration}}
  - template: ../variables/global.yaml
  - template: ../variables/${{parameters.env}}/${{parameters.env}}.yaml

stages:
  - stage: Network
    condition: and(eq(${{parameters.Network_deployment}}, true), eq(${{parameters.endpoint_integration}}, true))
    variables:
    - name:  ApplicationUnit
      value: infra
    jobs:
      ### Virtual Network deployment
      - job: vnet
```

```
  variables:
    - template: ../variables/${{parameters.env}}/infra/network.yaml
  steps:
    - template: ./templates/deploy.yaml
      parameters:
        validate: ${{parameters.validation}}
        parametersFile: azuredeploy-vnet.parameters.json
        templateFile: azuredeploy-vnet.template.json
        armFolder: arm
### Network security group deployment
- job: nsg
  dependsOn: vnet
  variables:
    - template: ../variables/${{parameters.env}}/infra/network.yaml
  steps:
    - template: ./templates/deploy.yaml
      parameters:
        validate: ${{parameters.validation}}
        parametersFile: azuredeploy-nsg.parameters.json
        templateFile: azuredeploy-nsg.template.json
        armFolder: arm

### Infrustructure subnet deployment
- job: subnet_infra
  dependsOn: nsg
  variables:
    - template: ../variables/${{parameters.env}}/infra/network.yaml
  steps:
    - template: ./templates/deploy.yaml
      parameters:
        validate: ${{parameters.validation}}
        parametersFile: azuredeploy-subnet.parameters.json
        templateFile: azuredeploy-subnet.template.json
        armFolder: arm
### Frontend subnet deployment
- job: subnet_fe
  dependsOn: nsg
  variables:
```

```
      - template: ../variables/${{parameters.env}}/infra/sub-fe.yaml
    steps:
      - template: ./templates/deploy.yaml
        parameters:
          validate: ${{parameters.validation}}
          parametersFile: azuredeploy-subnet.parameters.json
          templateFile: azuredeploy-subnet.template.json
          armFolder: arm
  ### Backend subnet deployment
  - job: subnet_be
    dependsOn: nsg
    variables:
      - template: ../variables/${{parameters.env}}/infra/sub-be.yaml
    steps:
      - template: ./templates/deploy.yaml
        parameters:
          validate: ${{parameters.validation}}
          parametersFile: azuredeploy-subnet.parameters.json
          templateFile: azuredeploy-subnet.template.json
          armFolder: arm
  ### Subnet deployment of Backend KV
  - job: subnet_kv_be
    dependsOn: nsg
    variables:
      - template: ../variables/${{parameters.env}}/infra/sub-kv-be.yaml
    steps:
      - template: ./templates/deploy.yaml
        parameters:
          validate: ${{parameters.validation}}
          parametersFile: azuredeploy-subnet.parameters.json
          templateFile: azuredeploy-subnet.template.json
          armFolder: arm

- stage: Keyvault_for_Be
  variables:
  - name:  ApplicationUnit
    value: be
  jobs:
```

```
    ### Key Vault deployment for Backend
    - job: KV_backend
      variables:
        - template: ../variables/${{parameters.env}}/be/kv.yaml
      steps:
        - template: ./templates/deploy.yaml
          parameters:
            validate: ${{parameters.validation}}
            parametersFile: azuredeploy-kv.parameters.json
            templateFile: azuredeploy-kv.template.json
            armFolder: arm
    ### Create access policy for Pipeline's ClientId to backend KV
    - job: KV_back_Access
      dependsOn: KV_backend
      variables:
        - template: ../variables/${{parameters.env}}/be/kv.yaml
      steps:
        - template: ./templates/deploy.yaml
          parameters:
            validate: ${{parameters.validation}}
            parametersFile: azuredeploy-kv-access.parameters.json
            templateFile: azuredeploy-kv-access.template.json
            armFolder: arm

  - stage: Keyvault_for_Infra
    variables:
    - name:  ApplicationUnit
      value: infra
    jobs:
      ### Key Vault deployment
      - job: kv_infra
        variables:
          - template: ../variables/${{parameters.env}}/infra/kv.yaml
        steps:
          - template: ./templates/deploy.yaml
            parameters:
              validate: ${{parameters.validation}}
              parametersFile: azuredeploy-kv.parameters.json
```

```
        templateFile: azuredeploy-kv.template.json
        armFolder: arm
    ### Create access policy for Pipeline's ClientId to infra KV
    - job: KV_infra_Access
      dependsOn: kv_infra
      variables:
        - template: ../variables/${{parameters.env}}/infra/kv.yaml
      steps:
        - template: ./templates/deploy.yaml
          parameters:
            validate: ${{parameters.validation}}
            parametersFile: azuredeploy-kv-access.parameters.json
            templateFile: azuredeploy-kv-access.template.json
            armFolder: arm
    ### Private DNS deployment
    - job: privatedns
      condition: eq(${{parameters.endpoint_integration}}, true)
      variables:
        - template: ../variables/${{parameters.env}}/infra/kv.yaml
      steps:
        - template: ./templates/deploy.yaml
          parameters:
            validate: ${{parameters.validation}}
            parametersFile: azuredeploy-privatedns.parameters.json
            templateFile: azuredeploy-privatedns.template.json
            armFolder: arm
    ### Private endpoint and Network Interfase deployment for Infra
    - job: endpoint_infra
      condition: and(eq(${{parameters.Network_deployment}}, true),
eq(${{parameters.endpoint_integration}}, true))
      dependsOn: [ privatedns, kv_infra ]
      variables:
        - template: ../variables/${{parameters.env}}/infra/kv.yaml
      steps:
        - template: ./templates/deploy.yaml
          parameters:
            validate: ${{parameters.validation}}
            parametersFile: azuredeploy-endpoint.parameters.json
```

```
            templateFile: azuredeploy-endpoint.template.json
            armFolder: arm

        ### Private endpoint and Network Interfase deployment for Be
        - job: endpoint_be
          dependsOn: [ privatedns ]
          condition: and(eq(${{parameters.Network_deployment}}, true), eq(${{parameters.endpoint_integration}}, true))
          variables:
            - template: ../variables/${{parameters.env}}/be/kv.yaml
          steps:
            - template: ./templates/deploy.yaml
              parameters:
                validate: ${{parameters.validation}}
                parametersFile: azuredeploy-endpoint.parameters.json
                templateFile: azuredeploy-endpoint.template.json
                armFolder: arm
```

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    },
    "privateEndpointName": {
      "type": "string"
    },
    "privateLinkServiceName": {
      "type": "string"
    },
    "privateLinkServiceType": {
      "type": "string"
    },
    "privateLinkServiceRG": {
```

```
            "type": "string"
          },
          "vnetName": {
            "type": "string"
          },
          "subnetName": {
            "type": "string"
          },
          "environmentTag": {
            "type": "string"
          },
          "ownerTag": {
            "type": "string"
          },
          "groupId": {
            "type": "string",
            "metadata": "Target sub-resource type to which created edpoint will be connected. For example: registry, vault, dataFactory, etc. "
          },
          "dnsZoneName": {
            "type": "array"
          },
          "subDomainDnsName": {
            "type": "array"
          },
          "dnsZoneRG": {
            "defaultValue": "[resourceGroup().name]",
            "type": "string"
          },
          "dnsZoneSubscription": {
            "defaultValue": "[subscription().subscriptionId]",
            "type": "string"
          }
        },
        "variables":{
          "replacedPrivateLinkServiceName": "[if(equals(parameters('groupId'), 'registry'), replace(parameters('privateLinkServiceName'),'-',''), parameters('privateLinkServiceName'))]",
          "copy": [
```

```
            {
              "name": "privateDnsZone",
              "count": "[length(parameters('dnsZoneName'))]",
              "input": {
                "name":
"[if(empty(parameters('subDomainDnsName')),variables('replacedPrivateLinkServiceName'),parameters('subDomainDnsName')[copyIndex('privateDnsZone')])]",
                "properties": {
                  "privateDnsZoneId":
"[if(empty(parameters('dnsZoneSubscription')),resourceId(parameters('dnsZoneRG'),'Microsoft.Network/privateDnsZones',parameters('dnsZoneName')[copyIndex('privateDnsZone')]),resourceId(parameters('dnsZoneSubscription'),parameters('dnsZoneRG'),'Microsoft.Network/privateDnsZones',parameters('dnsZoneName')[copyIndex('privateDnsZone')]))]"
                }
              }
            }
          ]
        },
        "resources": [
          {
            "apiVersion": "2019-04-01",
            "name": "[parameters('privateEndpointName')]",
            "type": "Microsoft.Network/privateEndpoints",
            "tags": {
              "environment": "[parameters('environmentTag')]",
              "owner": "[parameters('ownerTag')]"
            },
            "location": "[parameters('location')]",
            "properties": {
              "privateLinkServiceConnections": [
                {
                  "name": "[parameters('privateEndpointName')]",
                  "properties": {
                    "privateLinkServiceId":
"[if(empty(parameters('privateLinkServiceRG')),resourceId(parameters('privateLinkServiceType'),variables('replacedPrivateLinkServiceName')),resourceId(parameters('privateLinkServiceRG'),parameters('privateLinkServiceType'),variables('replacedPrivateLinkServiceName')))]",
                    "groupIds": [
```

```
                    "[parameters('groupId')]"
                  ]
                }
              }
            ],
            "subnet": {
              "id":
"[resourceId(parameters('dnsZoneRG'),'Microsoft.Network/virtualNetworks/subnets',parameters('vnetName'),para
meters('subnetName'))]"
            }
          }
        },
        {
          "type": "Microsoft.Network/privateEndpoints/privateDnsZoneGroups",
          "apiVersion": "2020-03-01",
          "name": "[concat(parameters('privateEndpointName'),'/dnsgroupname')]",
          "location": "[parameters('location')]",
          "dependsOn": [
            "[resourceId('Microsoft.Network/privateEndpoints', parameters('privateEndpointName'))]"
          ],
          "properties": {
            "privateDnsZoneConfigs": "[variables('privateDnsZone')]"
          }
        }
      ]
    }.
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ З РОЗГОРТАННЯ ІНФРАСТРУКТУРИ НА ОСНОВІ ХМАРНОГО ПРОВАЙДЕРА AZURE

Виконав: студент 2 курсу, групи 2КН-22м
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Доманський Б.В.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Барабан С.В.
(прізвище та ініціали)
«07» 12 2023 р.

Вінниця ВНТУ – 2023 рік

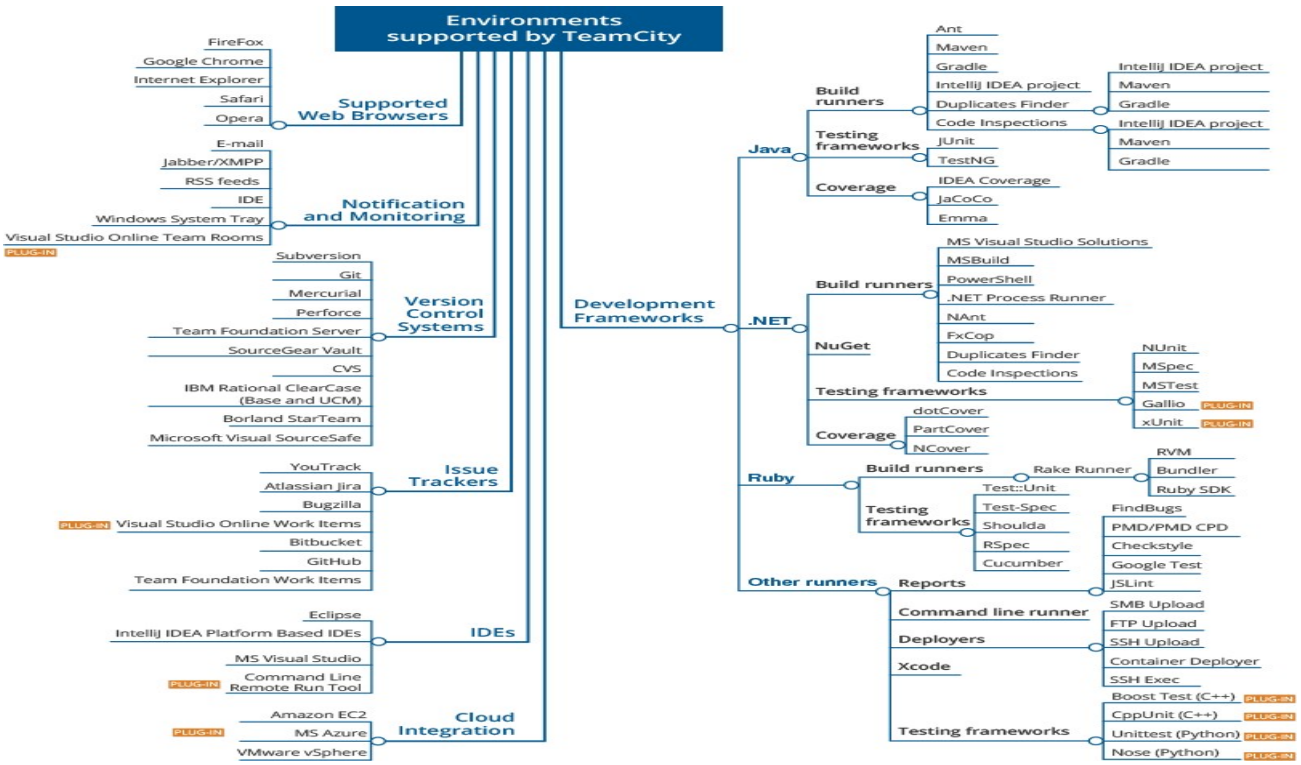

Рисунок В.1 – Схема основних зовнішніх інтеграцій інструменту TeamCity

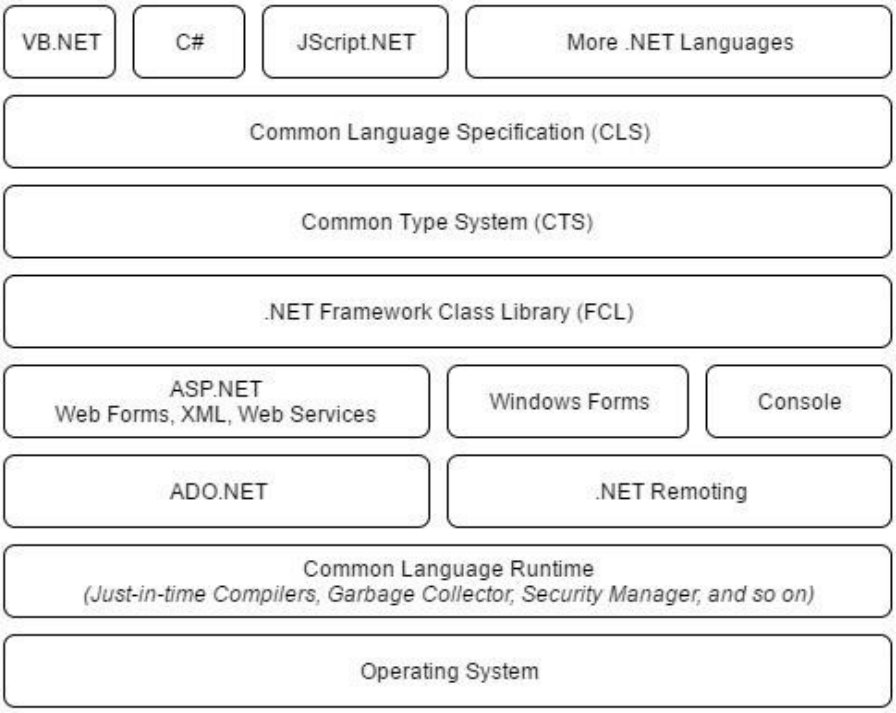

Рисунок В.2 – Архітектура платформи .NET

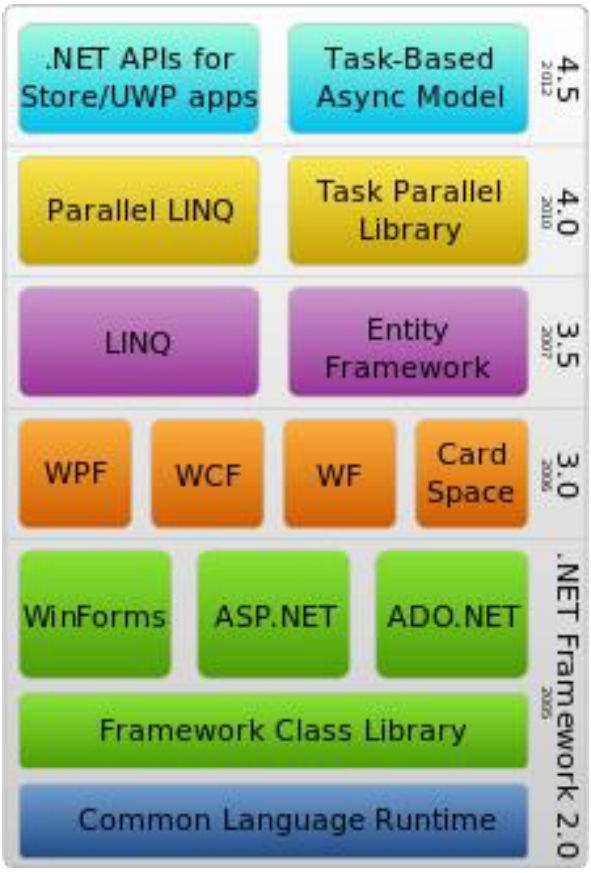

Рисунок В.3 – Стек технологій .NET Framework

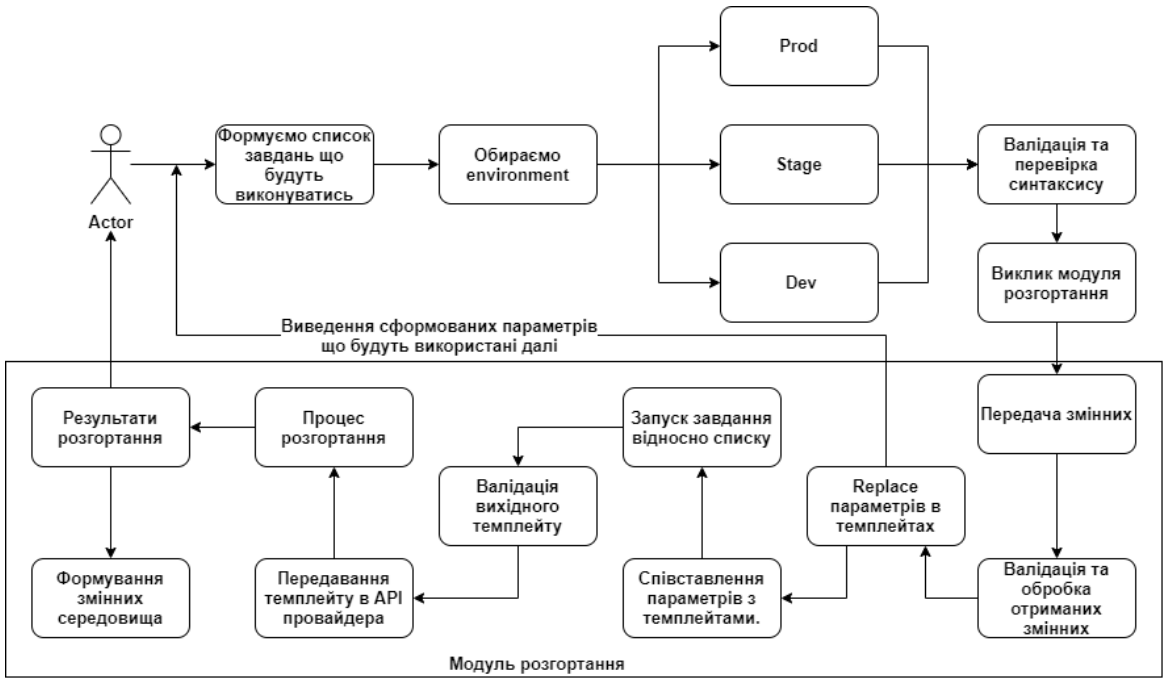

Рисунок В.4 – Структура інформаційної технології по розгортанню інфраструктури на основі хмарного провайдера Azure

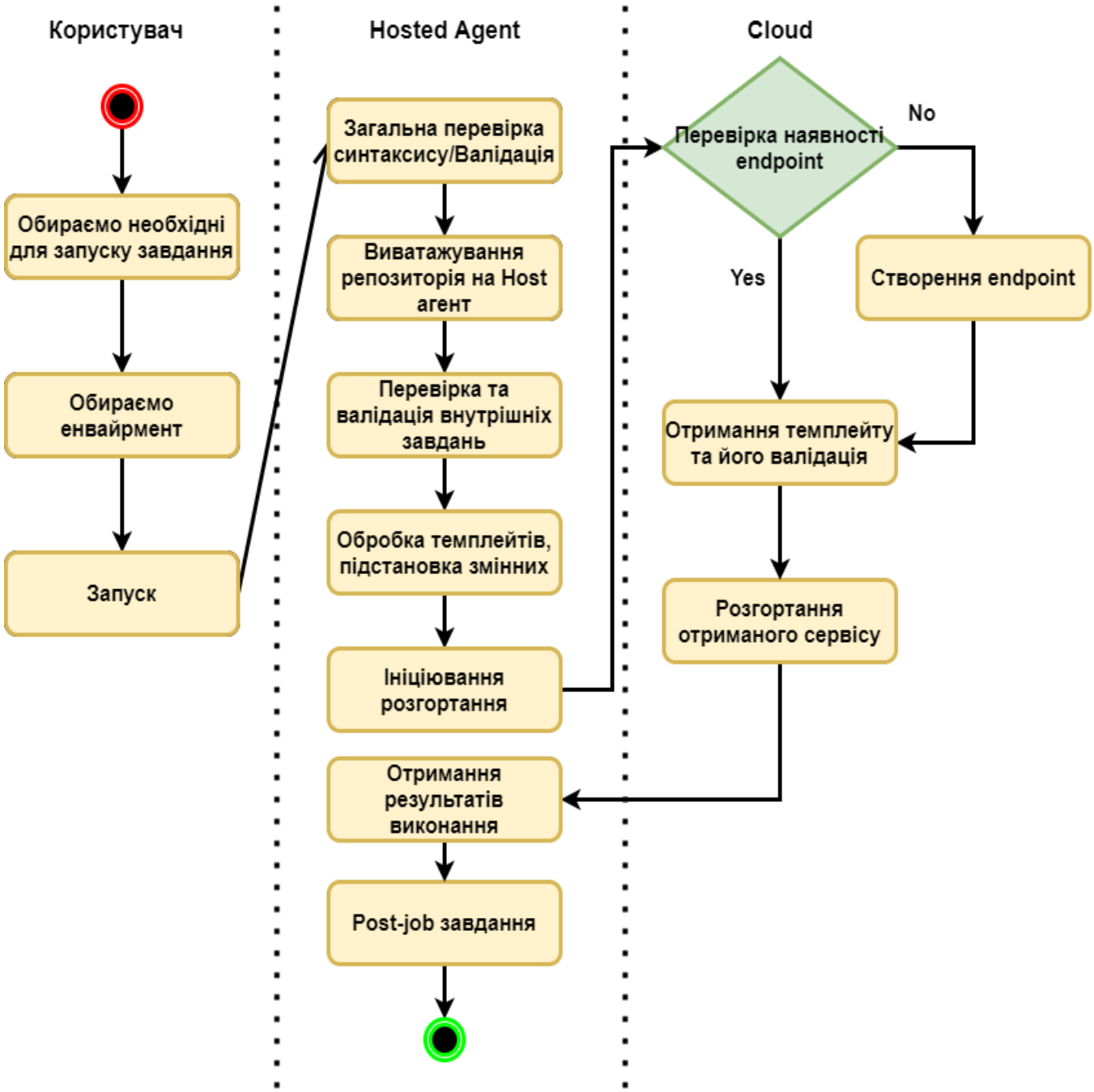

Рисунок В.5 – UML діаграми розгортання інфраструктури на основі хмарного провайдера Azure

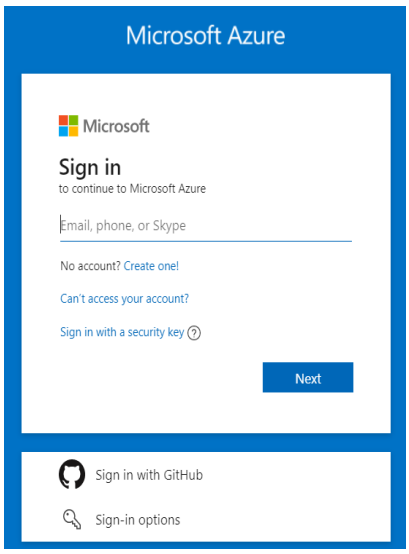

Рисунок В.6 – Сторінка авторизації в платформу Azure DevOps

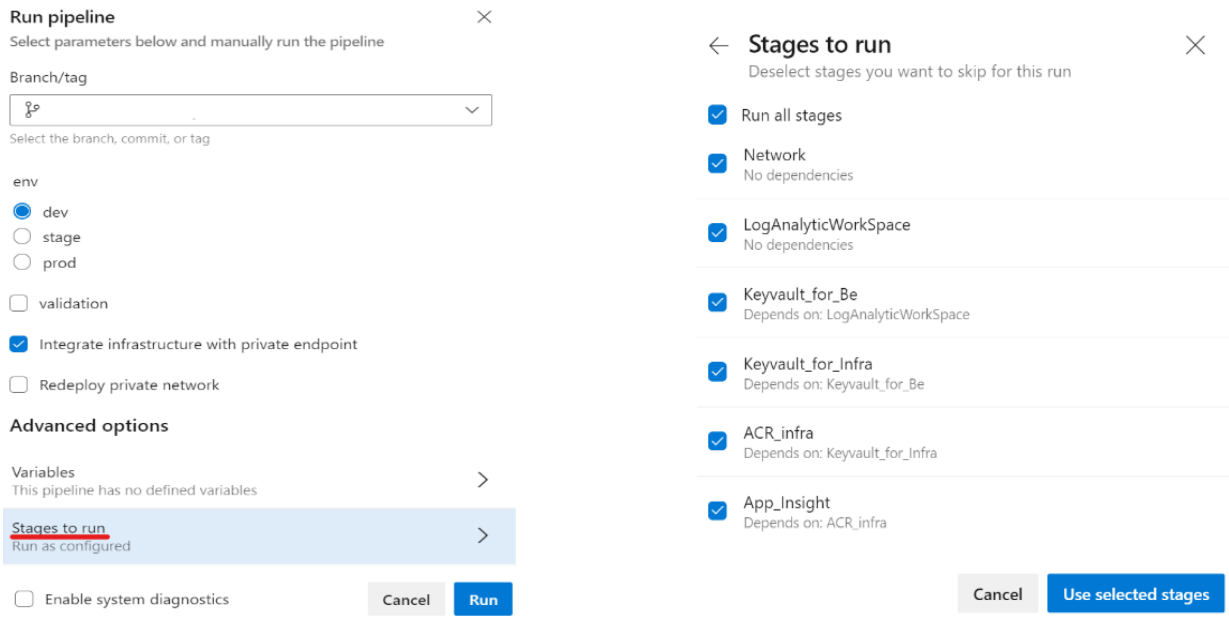

Рисунок В.7 – Графічний інтерфейс задавання вхідних параметрів

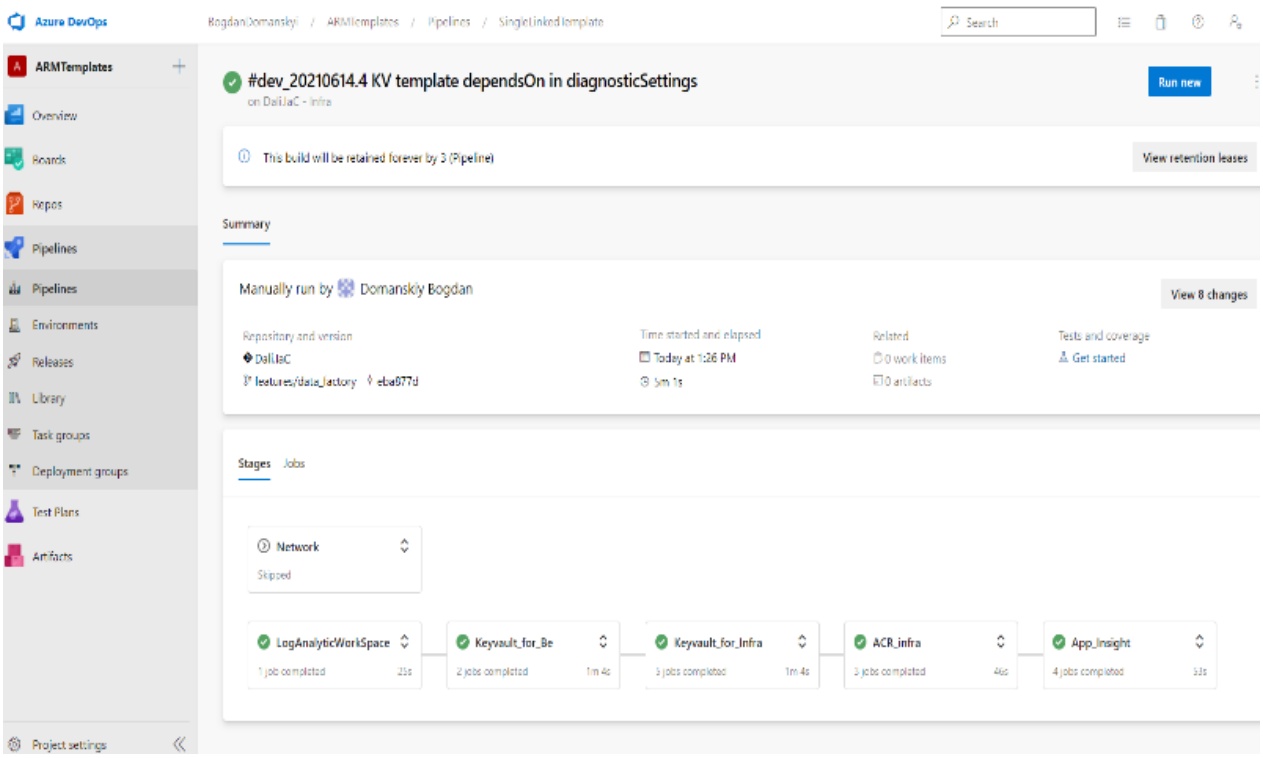

Рисунок В.8 – Список запущених етапів

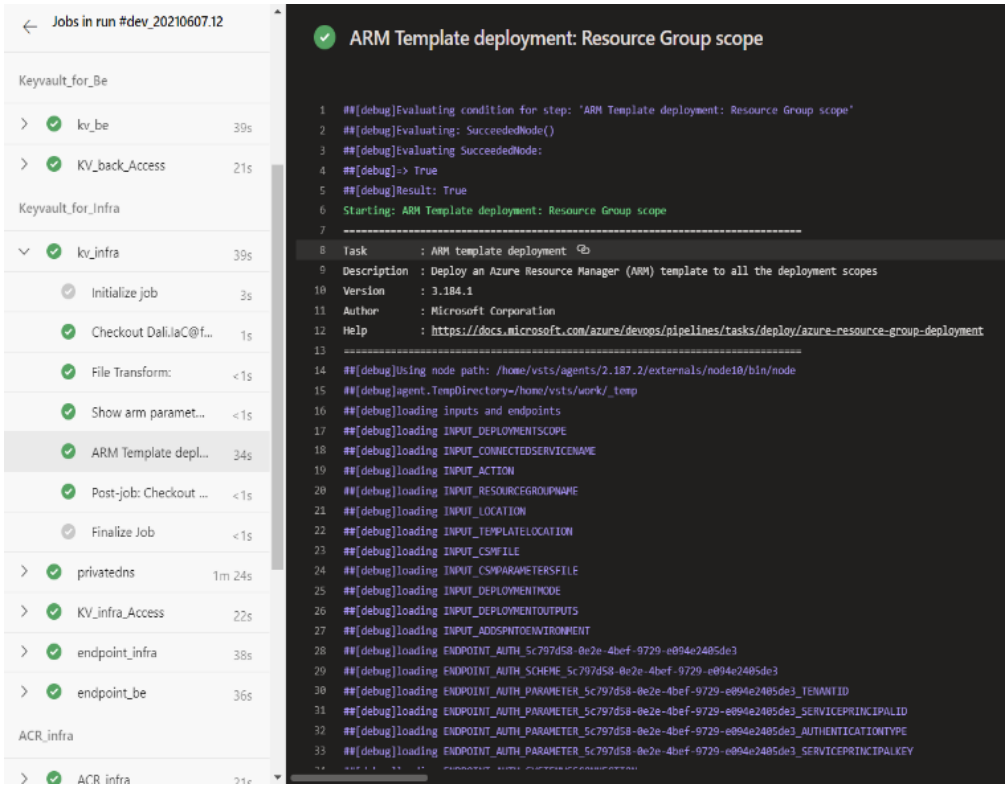

Рисунок В.9 – Детальна інформація одного з етапів

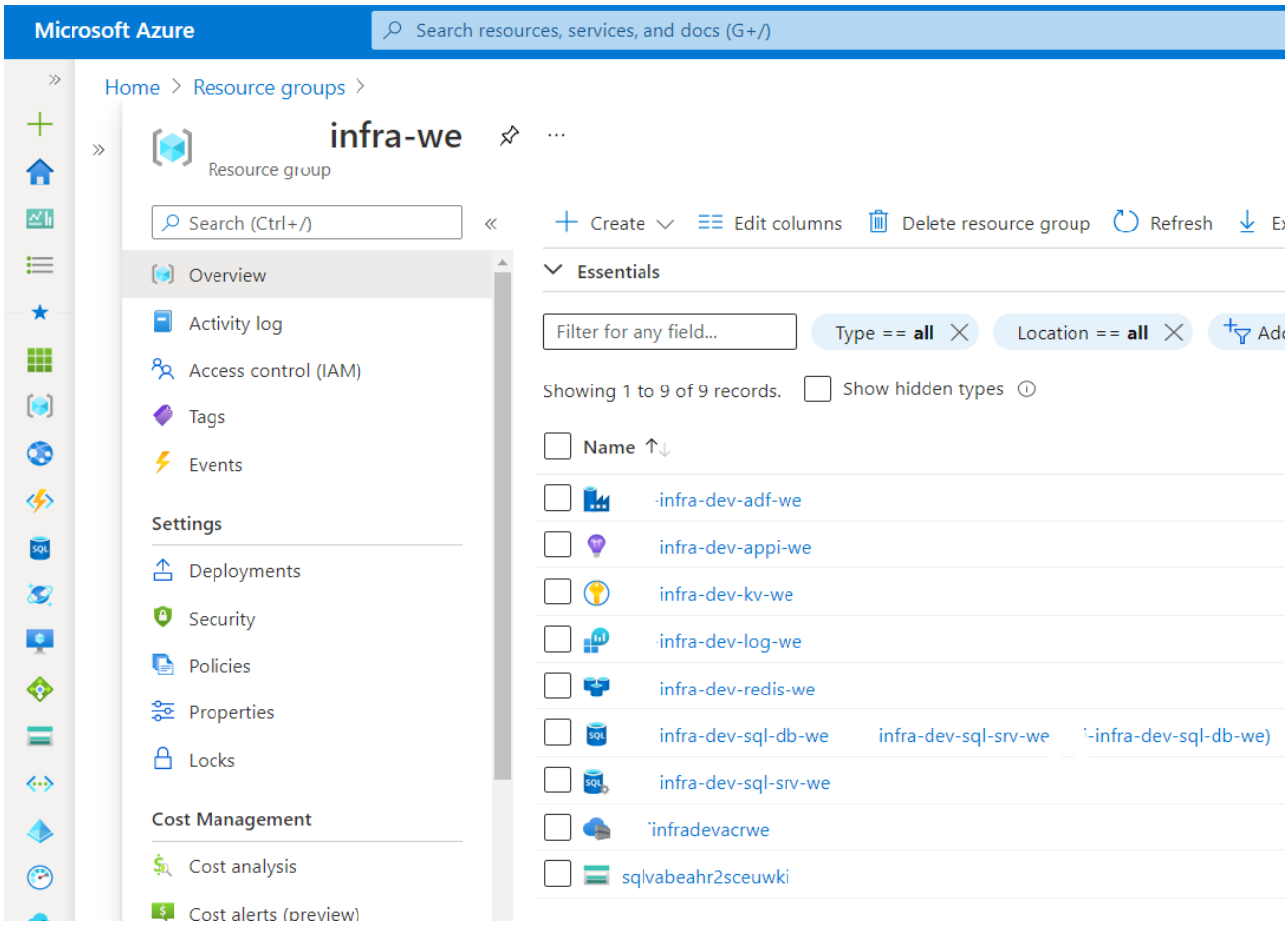

Рисунок В.10 –Огляд розгорнутої інфраструктури

Додаток Г (довідниковий)

Інструкція користувача

1. Виконати авторизацію в особистий акаунт Microsoft з використанням інтегрованої логін форми як це показано на рисунку Г.1;

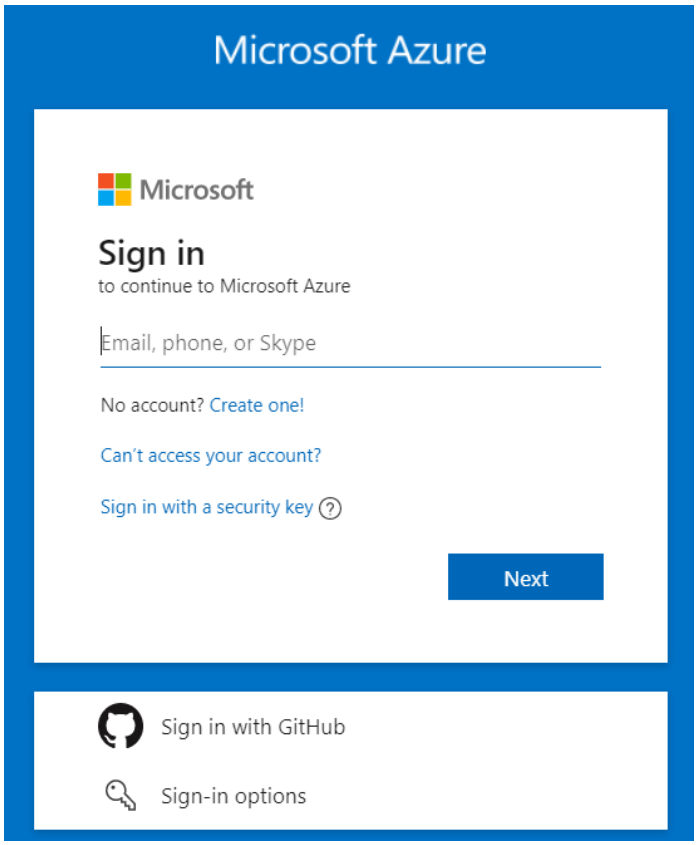

Рисунок Г.1 – Вхід до системи

2. Перейти до репозиторію котролю версій за посиланням https://dev.azure.com/{Назва організації}/ та перейти до вкладки Pipelines щоб відобразити список нещодавно запущених завдань розгортання інфраструкутри. Як це продемонстровано на рисунку Г.2;

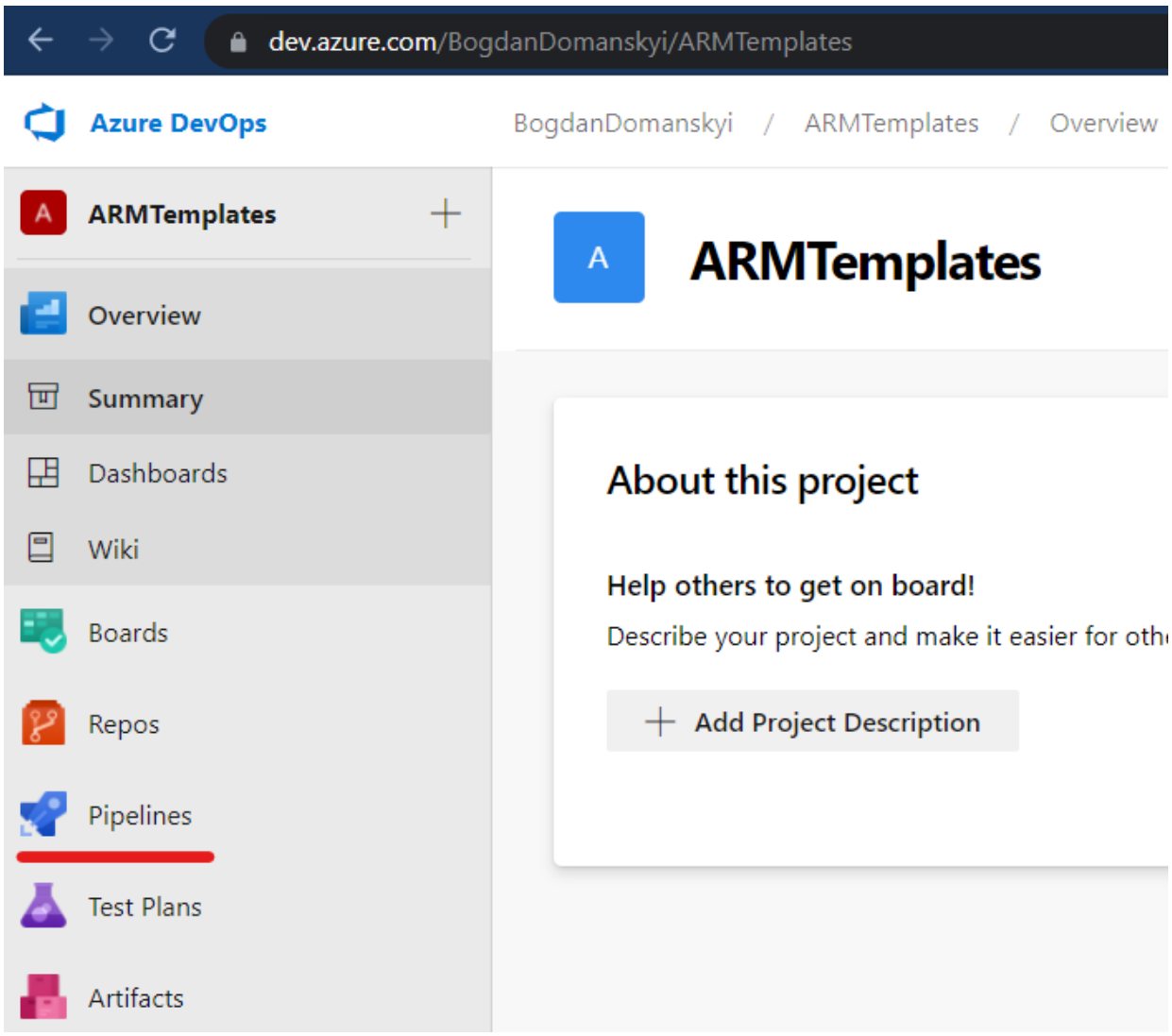

Рисунок Г.2 –Відображення списку Pipelines

3. Тепер необхідно задати конфігурації, що будуть вичитуватись в процесі розгортання інфраструктури та відповідати за запуск відповідні функціональні складові. Це можна побачити на рисунку Г.3. За необхідності розгортання лише певної частини інфраструктури – потрібно обрати задачі, що відповідають за процес створення сервісів які нас цікавлять, натиснувши кнопку “Stages to Run” як це продемонстровано на рисунку Г.3 та Г.4;

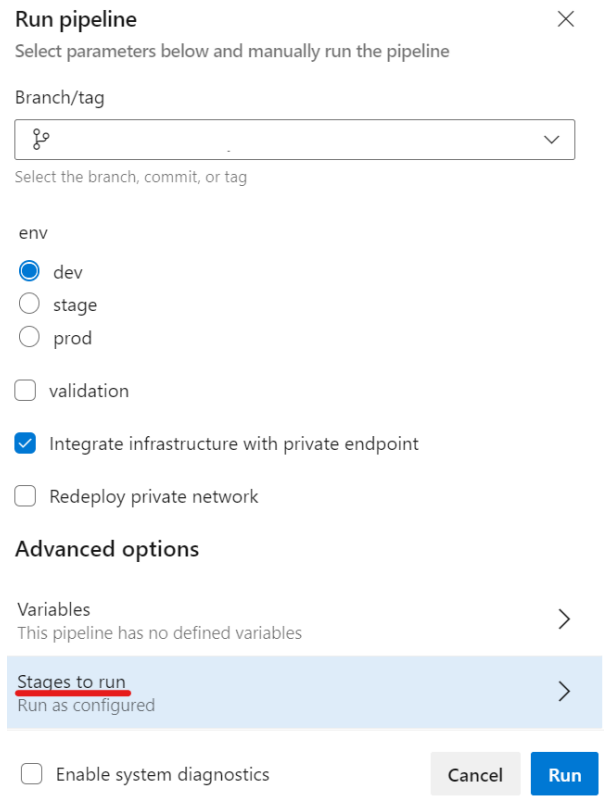

Рисунок Г.3 – Задання вхідних конфігурацій

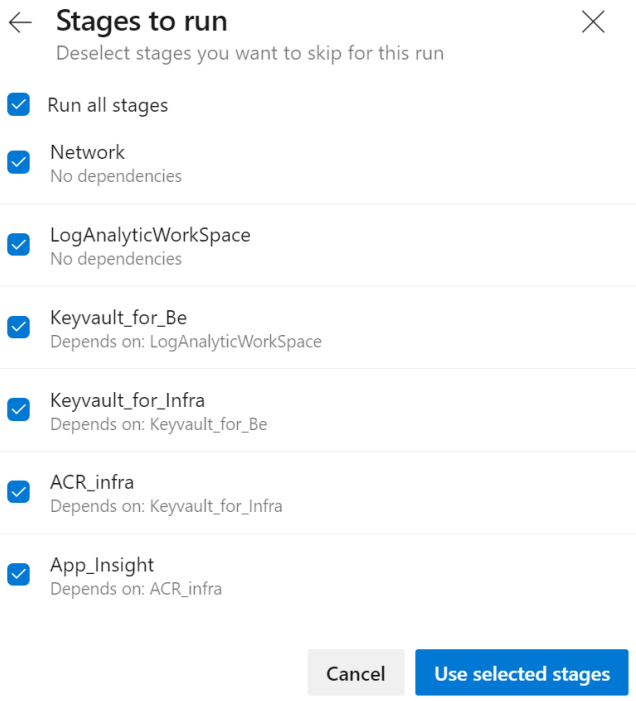

Рисунок Г.4 – Визначення списку завдань для створення сервісів

4. Для перегляду більш детальної інформації процесу розгортання інфраструктури необхідно відкрити потрібний нам Pipeline та обрати результати, які нас цікавлять (рисунок Г.5);

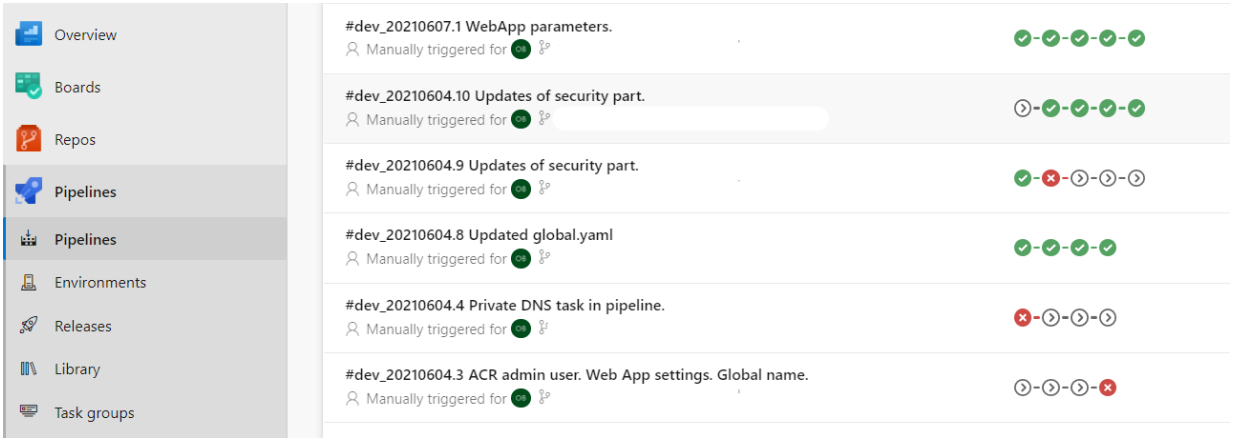

Рисунок Г.5 – Сторінка історії запуску Pipeline

5. Ще в процесі розгортання інфраструктури з'явиться можливість перегляду більш детальної інформації по кожній з задач, лише необхідно натиснути на ту, що цікавить (рисунок Г.6);

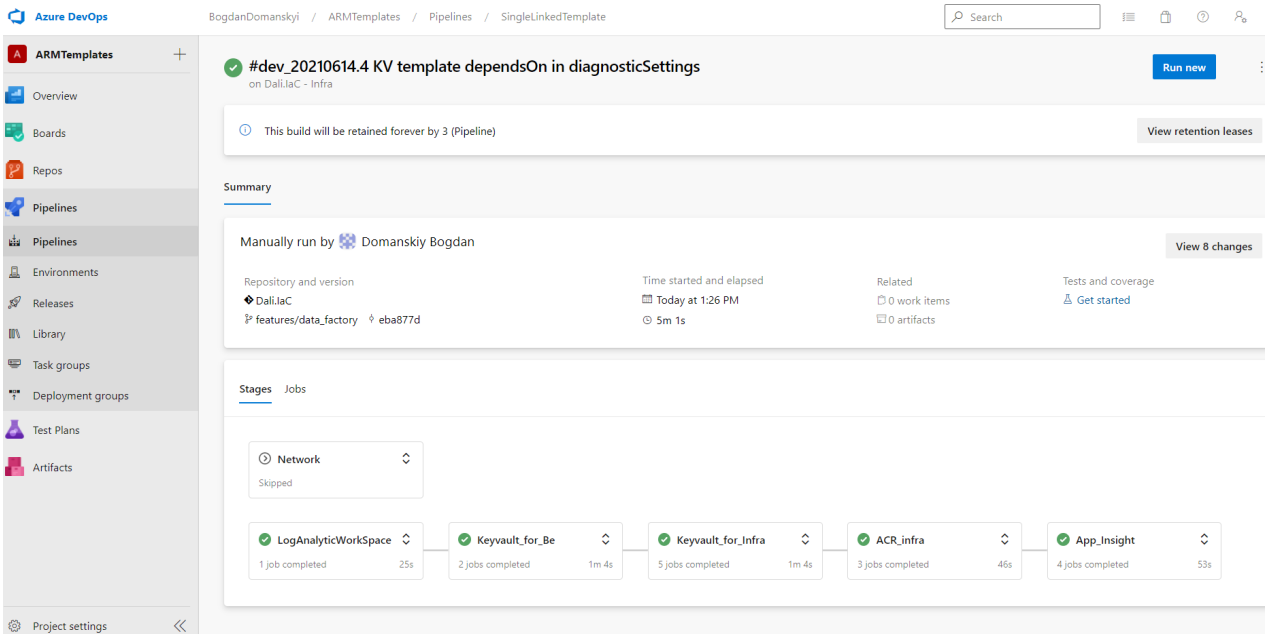

Рисунок Г.6 – Вікно статусу задач

6. Після успішного виконання pipeline можна переглянути більш детальну інформацію по кожному з завдань, що є складовими певної задачі (рисунок Г.7);

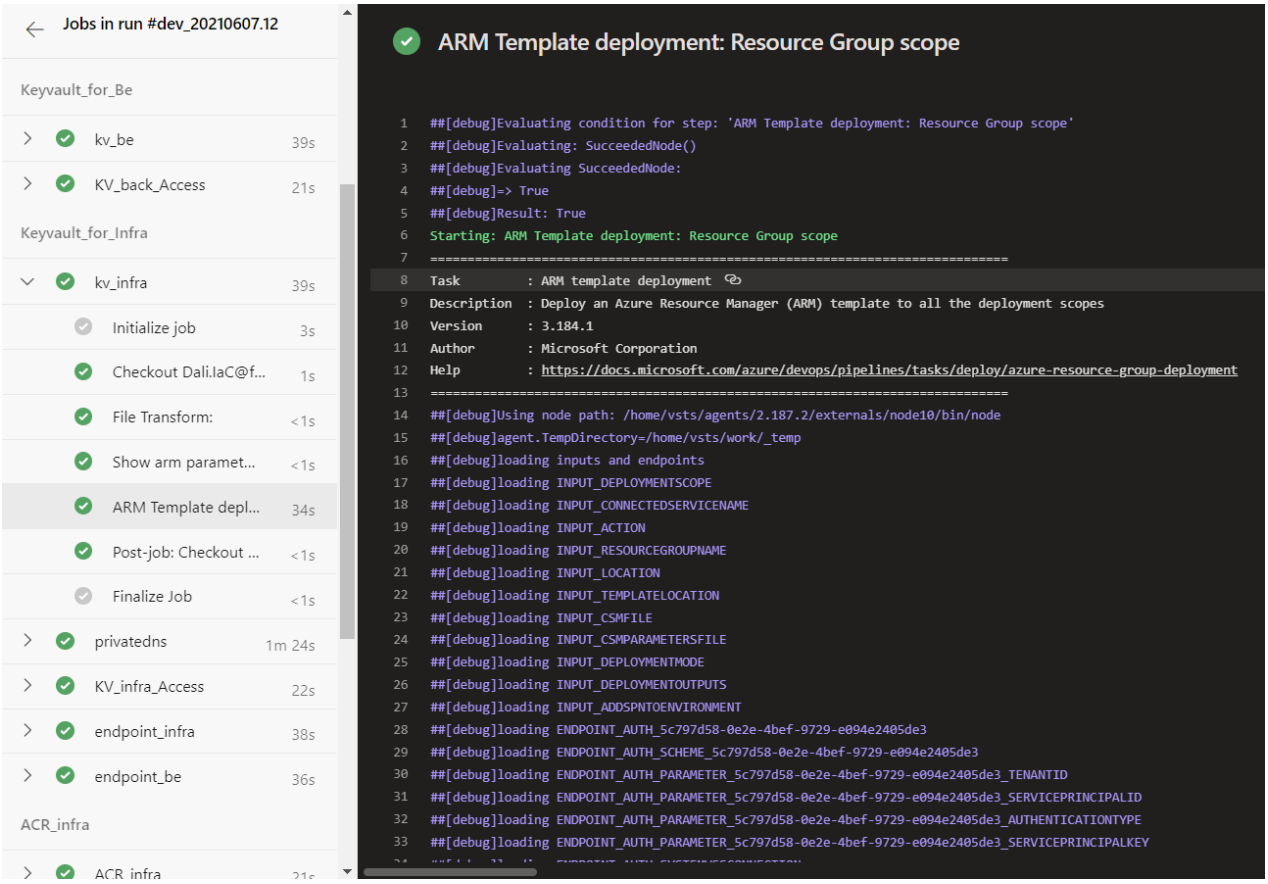

Рисунок Г.7 – Результати виконання завдань

7. Для ознайомлення з ресурсами та самою інфраструктурою, що була створена в результаті роботи додатку, необхідно перейти за посиланням https://portal.azure.com/ де й можна продовжити продовжити їхнє використання як це показано на рисунку Г.8.

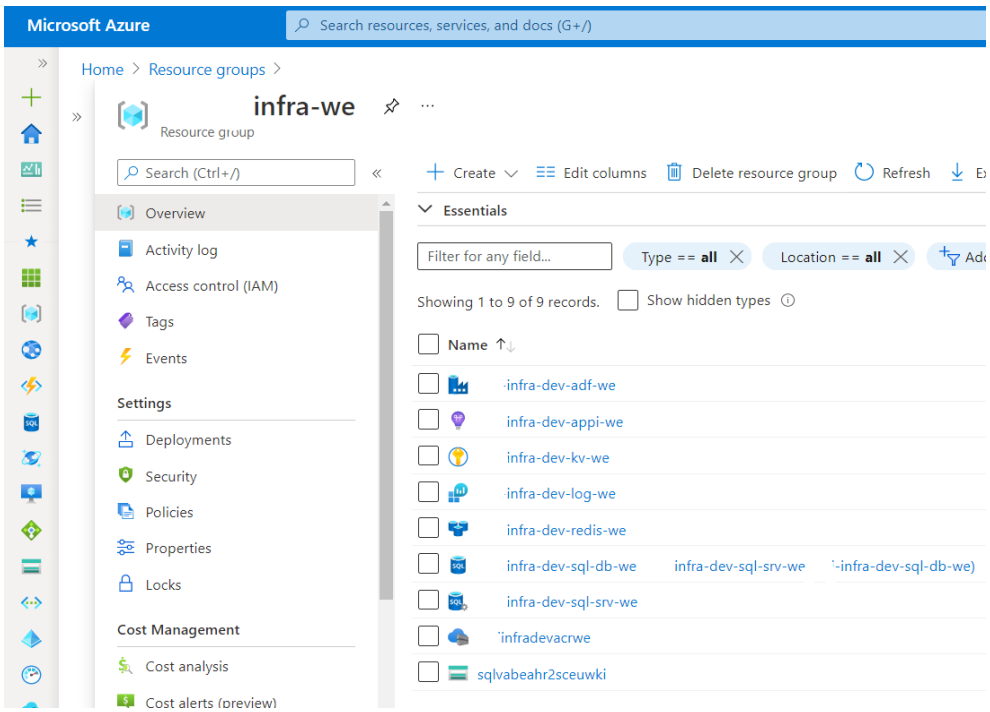

Рисунок Г.8 – Результат роботи програмного забезпечення

Всі інші функції та можливості програмного забезпечення не потребують детального опису і їх можна зрозуміти базуючись на офіційній документації Microsoft.