Міністерство освіти і науки України
Харківський національний університет В.Н.Каразіна

Кваліфікаційна наукова праця
На правах рукопису

БИЗОВ ІВАН СЕРГІЙОВИЧ

УДК 004.4:004.9

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ СЕРВЕРНОЮ ІНФРАСТРУКТУРОЮ З ВИКОРИСТАННЯМ DEVOPS ІНСТРУМЕНТІВ

Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Науковий керівник: Яковлев Сергій Всеволодович,
доктор фізико-математичних наук, професор

Харків 2026

АНОТАЦІЯ

Бизов І.С. Інформаційна технологія автоматизації управління серверною інфраструктурою з використанням DevOps інструментів. Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії (PhD) за спеціальністю 122 Комп'ютерні науки. (Галузь знань 12 Інформаційні технології) Харківський національний університет ім. В.Н. Каразіна. Харків, 2026.

Дисертаційна робота присвячена розробці методів, моделей та інформаційної технології для автоматизованого управління серверною інфраструктурою в локальних, хмарних та гібридних середовищах. У роботі представлено концепцію і реалізацію автоматизованої системи для створення, конфігурації та моніторингу фізичних і віртуальних серверів із залученням таких технологій як Python, Terraform, Ansible, MySQL, хмарних провайдерів, використання хмарних API, а також веб-застосунку для централізованого керування інфраструктурою. Особливу увагу приділено адаптації під гібридні середовища та сценарії використання в освітніх, дослідницьких і комерційних проєктах. Дослідження охоплює всі етапи життєвого циклу управління серверною інфраструктурою — від ініціалізації до моніторингу, акцентуючи на важливості гнучкості, масштабованості та автоматизації.

Об'єктом дослідження - є процеси управління серверною інфраструктурою сучасних інформаційних систем у локальних, хмарних та гібридних середовищах.

Предметом дослідження - є методи, моделі та засоби автоматизації управління серверною інфраструктурою на основі DevOps-практик, зокрема інфраструктури як коду, безперервної інтеграції та розгортання, а також механізмів взаємодії з API хмарних провайдерів у гібридних середовищах.

Мета і задачі дослідження – розробка та обґрунтування інформаційної технології автоматизованого управління серверною інфраструктурою в

гібридних середовищах на основі DevOps-підходів для забезпечення централізації, відтворюваності, масштабованості та підвищення ефективності процесів управління інфраструктурою.

У вступі обґрунтовано актуальність теми дисертаційної роботи, пов'язаної з необхідністю автоматизації управління серверною інфраструктурою в умовах цифрової трансформації та гібридних середовищ, зазначено зв'язок роботи з науковими темами в галузі комп'ютерних наук та автоматизації IT-систем, сформульовано мету дослідження як обґрунтування та розробку інформаційної технології автоматизованого управління серверною інфраструктурою в гібридних середовищах на основі DevOps-підходів, що забезпечує централізацію, відтворюваність, масштабованість і підвищення ефективності процесів управління інфраструктурними ресурсами, визначено завдання дослідження, об'єкт як процеси управління серверною інфраструктурою сучасних інформаційних систем у локальних, хмарних та гібридних середовищах, предмет як методи, моделі та засоби автоматизації управління серверною інфраструктурою на основі DevOps-практик, зокрема інфраструктури як коду, безперервної інтеграції та розгортання, а також механізмів взаємодії з API хмарних провайдерів у гібридних середовищах, показано наукову новизну та практичне значення отриманих результатів, наведено інформацію про практичне використання, особистий внесок здобувача, апробацію результатів дослідження та їх висвітлення у публікаціях. Приведено відомості щодо структури та обсягу дисертаційної роботи.

У першому розділі проведено аналіз сучасного стану автоматизації управління серверною інфраструктурою. Проаналізовано існуючі рішення та інструменти, включаючи системи управління конфігураціями, платформи оркестрації контейнерів та хмарні провайдери, з оглядом сучасних технологій і методів управління, а також особливостей використання хмарних провайдерів таких як AWS, Google Cloud, Azure та Hetzner. Висвітлено принципи та підходи DevOps, зокрема основні принципи CI/CD, Infrastructure as Code, моніторинг та логування, з оглядом популярних інструментів таких

як Ansible, Kubernetes, Terraform та Jenkins. Розглянуто проблеми та виклики автоматизації в гібридних середовищах, включаючи труднощі інтеграції локальних і хмарних серверів, проблеми масштабованості, безпеки, моніторингу та управління великими інфраструктурами.

У другому розділі розроблено інформаційну технологію для автоматизації серверної інфраструктури. Описано архітектуру рішення на основі Django, з моделями взаємодії між серверною інфраструктурою та хмарними провайдерами. Розглянуто алгоритм автоматизації підняття серверів для швидкого розгортання різних типів серверів з використанням DevOps-інструментів. Проаналізовано інтеграцію з хмарними провайдерами та локальними серверами, включаючи технічні деталі взаємодії з API провайдерів таких як AWS, Azure, Google Cloud, та рішення проблем інтеграції локальних серверів з хмарними.

У третьому розділі проведено експериментальне дослідження ефективності розробленої технології. Описано середовище тестування з вибором серверного середовища для проведення експериментів, включаючи віртуальні машини, хмарні інстанси та локальні сервери. Встановлено постановку експериментів за типами, такими як швидкість розгортання серверів, продуктивність системи та стійкість до навантажень. Проаналізовано результати експериментів з оцінкою ефективності, порівнянням часу розгортання, надійності, автоматизації та з існуючими інструментами і рішеннями. Запропоновано шляхи подальшої оптимізації системи та рекомендації щодо її використання.

У четвертому розділі розглянуто впровадження та практичне застосування. Описано практичні аспекти впровадження з реальними сценаріями використання розробленої технології. Проаналізовано кейси застосування в різних інфраструктурах, включаючи гібридні середовища та підприємства. Розглянуто економічні аспекти впровадження DevOps-інструментів з аналізом економічної доцільності та витрат при впровадженні автоматизованих рішень.

У висновках наведено основні результати дисертаційної роботи щодо вирішення поставлених наукових задач дослідження. За результатами дослідження отримано такі **наукові результати**:

1. Обґрунтовано необхідність автоматизації управління серверною інфраструктурою в гібридних середовищах, проаналізовано обмеження існуючих підходів щодо централізації, масштабованості та керованості процесів.

2. Обґрунтовано концепцію інформаційної технології автоматизованого управління серверною інфраструктурою, яка інтегрує DevOps-практики, веб-орієнтовані засоби управління та механізми взаємодії з хмарними API.

3. Розроблено архітектурно-алгоритмічні рішення для автоматизованого розгортання, конфігурації та моніторингу серверних ресурсів у локальних, хмарних і гібридних середовищах.

4. Проведено експериментальне дослідження ефективності розробленої інформаційної технології з використанням показників часу розгортання, стабільності конфігурацій та експлуатаційної надійності.

5. Сформульовано практичні рекомендації щодо впровадження та подальшого розвитку запропонованої інформаційної технології в освітніх, наукових і комерційних IT-системах.

6. Розроблено інформаційну технологію автоматизованого управління серверною інфраструктурою в гібридних середовищах на основі DevOps-підходів, особливістю якої є інтеграція інфраструктури як коду, безперервної інтеграції та розгортання, механізмів взаємодії з API хмарних провайдерів і централізованого веб-управління в єдину керовану систему.

7. Результати роботи впроваджено у науково-дослідних роботах Навчально-наукового інституту комп’ютерних наук та штучного інтелекту Харківського національного університету імені В. Н. Каразіна.

8. Запропоновані методи та інформаційна технологія можуть бути повною мірою впроваджені в системи управління IT-інфраструктурою

організацій, а також сприяти подальшим дослідженням у сфері автоматизації на основі DevOps.

Ключові слова: *хмарні сервіси, виявлення аномалій, моніторинг, хмарна інфраструктура, хмарне середовище, критерій аномалії, сервісні домени, інформаційні системи, веб застосунок, динамічне конфігурування, програмне забезпечення, інформаційна технологія, програмний компонент, система управління, ефективність*.

ABSTRACT

Byzov I.S. Information technology for automated server infrastructure management using DevOps tools. Qualifying scientific work on the rights of a manuscript.

Dissertation for the degree of Doctor of Philosophy (PhD) in specialty 122 Computer Science. V.N. Karazin Kharkiv National University. Kharkiv, 2026.

The dissertation is devoted to the development of methods, models, and information technology for automated server infrastructure management in local, cloud, and hybrid environments. The work presents the concept and implementation of an automated system for creating, configuring, and monitoring physical and virtual servers using technologies such as Python, Terraform, Ansible, MySQL, cloud providers, cloud APIs, and a web application for centralized infrastructure management. Particular attention is paid to adaptation for hybrid environments and usage scenarios in educational, research, and commercial projects. The study covers all stages of the server infrastructure management lifecycle — from initialization to monitoring, emphasizing the importance of flexibility, scalability, and automation.

The object of the study is the processes of server infrastructure management of modern information systems in local, cloud, and hybrid environments.

The subject of the study is the methods, models, and tools for automating server infrastructure management based on DevOps practices, in particular infrastructure as code, continuous integration and deployment, as well as mechanisms for interaction with cloud provider APIs in hybrid environments.

The purpose and objectives of the study are to develop and substantiate an information technology for automated server infrastructure management in hybrid environments based on DevOps approaches to ensure centralization, reproducibility, scalability, and increased efficiency of infrastructure management processes.

The introduction substantiates the relevance of the dissertation topic related to the need for automated server infrastructure management in the context of digital transformation and hybrid environments, indicates the connection of the work with scientific topics in the field of computer science and automation of IT systems,

formulates the research goal, defines research tasks, object, subject, shows scientific novelty and practical significance, and provides information on practical use, personal contribution, approbation, and publications.

In the first chapter, an analysis of the current state of server infrastructure management automation is conducted. Existing solutions and tools, including configuration management systems, container orchestration platforms, and cloud providers (AWS, Google Cloud, Azure, Hetzner), are analyzed. The principles of DevOps, CI/CD, Infrastructure as Code, monitoring, and logging are highlighted. Problems and challenges of automation in hybrid environments are considered.

In the second chapter, the information technology for server infrastructure automation is developed. The architecture based on Django is described, including interaction models between the infrastructure and cloud providers. The algorithm for rapid server deployment using DevOps tools is considered. Integration with cloud providers and local servers is analyzed, including technical details of API interaction and solutions for local server integration.

In the third chapter, an experimental study of the developed technology's effectiveness is conducted. The testing environment, including virtual machines, cloud instances, and local servers, is described. Experiments are conducted to evaluate deployment speed, system performance, and load stability. The results are analyzed with an assessment of efficiency and comparison with existing tools.

In the fourth chapter, implementation and practical application are considered. Practical aspects with real usage scenarios are described, including cases in hybrid environments and enterprises. Economic aspects and the feasibility of implementing DevOps tools are analyzed.

The conclusions present the **main results of the dissertation:**

1. The necessity of automating server infrastructure management in hybrid environments is substantiated, and the limitations of existing approaches are analyzed.

2. The concept of information technology for automated server infrastructure management integrating DevOps practices, web-based management, and cloud APIs is substantiated.

3. Architectural and algorithmic solutions for automated deployment, configuration, and monitoring in local, cloud, and hybrid environments are developed.

4. An experimental study of the effectiveness of the developed information technology using indicators of deployment time, configuration stability, and operational reliability is conducted.

5. Practical recommendations for the implementation and further development of the proposed technology in educational, scientific, and commercial IT systems are formulated.

6. The information technology for automated server infrastructure management based on DevOps approaches is developed, featuring the integration of IaC, CI/CD, cloud APIs, and centralized web management.

7. The results have been implemented in the research work of the Educational and Scientific Institute of Computer Science and Artificial Intelligence of Karazin University.

8. The proposed methods and technology can be fully implemented in organizational IT infrastructure management systems.

Keywords: *cloud services, anomaly detection, monitoring, cloud infrastructure, cloud environment, anomaly criterion, service areas, information systems, web application, dynamic configuration, software, information technology, software component, management system, efficiency.*

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у фахових наукових виданнях:

1. Бизов І. С. Імплементація DevOps технології у ІТ-проекти та життєві цикли компанії: аналіз та статистика. Збірник наукових праць Національного університету кораблебудування імені адмірала Макарова. 2023. № 4(17). С. 117–120. DOI: 10.15589/znp2023.4(493).17. Категорія Б.

2. Бизов І. С. Terraform vs Ansible: When and how to use infrastructure tools as code. Technologies and Engineering. 2025. № 6. С. 11–17. DOI: 10.30857/2786-5371.2024.6.1. Категорія Б.

Статті у фахових наукових виданнях у співавторстві

3. Бизов І. С., Яковлев С. В. Streamlined management of physical and cloud infrastructure through a centralized web interface. Systems Research and Information Technologies. 2025. № 2. С. 25–41. DOI: 10.20535/SRIT.2308-8893.2025.2.02. ISSN 1681–6048. Категорія А (Scopus Q4).

(Особистий внесок здобувача: розробка архітектури централізованої системи управління гібридною інфраструктурою, реалізація веб-додатка на базі Django, інтеграція Paramiko для SSH-доступу, керування Docker-контейнерами та хмарними API (Hetzner, AWS), реалізація моніторингу, планувальника завдань та кастомних команд, розробка редактора коду та шаблонів автоматизації, механізмів шифрування ключів, тестування на фізичних серверах та хмарних інстансах, проведення case study та вимірювання ефективності (скорочення часу конфігурації серверів на 44 %), підготовка схем, діаграм та практичного матеріалу статті. Результати відображено у розділах, присвячених архітектурі системи, методам управління інфраструктурою, автоматизації операцій, кейс-стаді та висновкам.

Особистий внесок С.В. Яковлева: наукове консультування щодо постановки завдання та структури статті, редагування тексту; відображено у вступі та формулюванні дослідницьких цілей.)

4. Бизов І. С., Яковлев С. В. Information technology for automation of server infrastructure management using DevOps tools. Radioelectronic and Computer Systems. 2025. № 3. С. 257–269. DOI: 10.32620/reks.2025.3.18. ISSN: 2663-2012. *Категорія А (Scopus Q2).*

(Особистий внесок здобувача: розробка архітектури автоматизованої системи управління серверною інфраструктурою, реалізація Python-скриптів для генерації конфігурацій Terraform, інтеграція з MySQL та DigitalOcean DNS API, автоматизація розгортання на платформі Hetzner, створення Ansible-playbook'ів, реалізація алгоритмів масштабування, формальна модель черги M/G/c для прогнозування часу розгортання, проведення експериментів (1–100 серверів), підготовка діаграм, таблиць та аналіз ефективності (скорочення часу розгортання на 90 % порівняно з ручними методами та на 50 % порівняно з Ansible), а також написання основного тексту статті. Результати відображено у розділах 2–5.

Особистий внесок С.В. Яковлева: наукове консультування, методологічна підтримка, аналіз результатів та редагування тексту; відображено у розділах 1 та 1.3.)

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Бизов І. С., Хруслов М. М. Імплементація DevOps технології у IT-проекти. Комп'ютерне моделювання у наукоємних технологіях (КМНТ): матеріали міжнародної науково-технічної конференції. Харків, 2023.

2. Бизов І. С., Хруслов М. М. Управління, налаштування та моніторинг інфраструктури через веб-застосунок із застосуванням технологій DevOps. Комп'ютерне моделювання у наукоємних технологіях (КМНТ): матеріали міжнародної науково-технічної конференції. Харків, 2024.

3. Бизов І. С., Яковлев С. В. Сучасна інформаційна технологія для автоматизованого управління серверними середовищами. Інтелектуальні технології у міждисциплінарних дослідженнях: матеріали міжнародної науково-практичної конференції. Харків, 2025.

4. Byzov I., Korobchynskyi K., Strukov V. Handling Hybrid Infrastructure Automation: Intelligent Management System with Anomaly Detection. ProfIT AI 2025 : Proceedings of the International Conference. Kharkiv, October 15–17, 2025. P. 305–311.

Сертифікати

У межах виконання дисертаційної роботи були створені та зареєстровані авторські права на програмні продукти, що є практичною реалізацією розробленої інформаційної технології автоматизованого управління серверною інфраструктурою. Реєстрація авторського права підтверджує оригінальність розроблених рішень та їх відповідність науковим завданням дисертації, а також надає правові підстави для подальшого використання та впровадження у наукових, освітніх і комерційних проектах.

- **Комп'ютерна програма:** «Алгоритм швидкого розгортання серверних ресурсів з інтеграцією Terraform-Ansible-Vault»:

– **Сертифікат:** УКРІНОВІ №144242

– **Дата реєстрації:** 10 березня 2026 р.

– **Автор:** Бизов Іван Сергійович

Програма реалізує автоматизований алгоритм підняття серверів у локальних, хмарних та гібридних середовищах з використанням ключових DevOps-інструментів, таких як Terraform для управління інфраструктурою як кодом, Ansible для конфігурації серверів та HashiCorp Vault для безпечного управління секретами. Це забезпечує централізоване, відтворюване та масштабоване розгортання серверних ресурсів, що корелює з архітектурно-алгоритмічними рішеннями, описаними у розділі 2 дисертації. Алгоритм також дозволяє прискорити процес інтеграції нових серверів у існуючу інфраструктуру та підвищує ефективність управління хмарними та локальними ресурсами.

- **Комп'ютерна програма:** «Interactive Service Backend System – Інтерактивна система серверного обслуговування (ISB System)»

– **Сертифікат:** УКРІНОВІ №141764

– **Дата реєстрації:** 21 січня 2026 р.

– **Автор:** Бизов Іван Сергійович

Програма ISB System забезпечує інтерактивне управління серверною інфраструктурою, включаючи моніторинг стану серверів, контроль навантаження та оперативне управління службами. Вона реалізує принципи централізованого веб-управління та інтеграції з хмарними API провайдерів (AWS, Azure, Google Cloud), що дозволяє здійснювати управління гібридною інфраструктурою у реальному часі. Розробка системи є практичним втіленням інформаційної технології, описаної у розділі 2 дисертації, та використовується для проведення експериментальних досліджень, описаних у розділі 3, щодо ефективності, надійності та масштабованості автоматизованого управління серверною інфраструктурою.

Обидва програмні продукти демонструють науковий та практичний внесок здобувача у розвиток методів автоматизації серверної інфраструктури на основі DevOps-підходів. Реєстрація авторських прав підтверджує інноваційність запропонованих рішень та їх застосування у реальних сценаріях управління локальними та хмарними ресурсами, що є важливою складовою практичної реалізації результатів дисертаційного дослідження.

Таким чином, сертифіковані програмні продукти є безпосередньою реалізацією розроблених методів, моделей та алгоритмів, що підтверджує особистий внесок здобувача у створення ефективної інформаційної технології автоматизованого управління серверною інфраструктурою.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface – програмний інтерфейс прикладного рівня

Ansible – інструмент автоматизації налаштувань та розгортання програмного забезпечення

CAPEX – Capital Expenditure – капітальні витрати на придбання та модернізацію інфраструктури

CI – Continuous Integration – безперервна інтеграція

CD – Continuous Delivery / Continuous Deployment – безперервна доставка / розгортання

CLI – Command Line Interface – інтерфейс командного рядка

CRUD – Create, Read, Update, Delete – базові операції над даними в базах даних

CPASI – Complex Pipeline for Automated Server Infrastructure – комплексний конвеєр автоматизації розгортання та управління серверною інфраструктурою

CSS – Cascading Style Sheets – каскадні таблиці стилів

DBMS – Database Management System – система керування базами даних

DevOps – Development and Operations – методологія інтеграції розробки та експлуатації програмного забезпечення

Django – веб-фреймворк для Python із відкритим кодом

DNS – Domain Name System – система доменних імен

HTML – HyperText Markup Language – мова розмітки гіпертексту

IaC – Infrastructure as Code – інфраструктура як код

IaaS – Infrastructure as a Service – інфраструктура як послуга

JS – JavaScript – мова програмування клієнтської частини вебзастосунків

JSON – JavaScript Object Notation – формат обміну даними

Mirohost – український хостинг-провайдер, що надає VPS і хмарні сервіси

MySQL – система керування реляційними базами даних

OPEX – Operational Expenditure – операційні витрати на підтримку та експлуатацію інфраструктури

PaaS – Platform as a Service – платформа як послуга

REST – Representational State Transfer – архітектурний стиль побудови API

RAM – Random Access Memory – оперативна пам'ять

SaaS – Software as a Service – програмне забезпечення як послуга

SSL – Secure Sockets Layer – протокол захищеного з'єднання

Terraform – інструмент Infrastructure as Code для автоматизації хмарних ресурсів

UML – Unified Modeling Language – універсальна мова моделювання

VPS – Virtual Private Server – віртуальний приватний сервер

VM – Virtual Machine – віртуальна машина

Webterminal – веб-застосунок для командного керування сервером або хостинг-акаунтом

Hetzner – німецький хмарний провайдер інфраструктурних послуг

ШІ – штучний інтелект

ВСТУП

Стрімке зростання обсягів даних, цифровізація державного та приватного інформаційного секторів, а також посилення вимог до гнучкості, масштабованості та надійності IT-систем зумовлюють необхідність перегляду традиційних підходів до адміністрування серверних ресурсів. У гібридних середовищах, які об'єднують локальні сервери та хмарні платформи, класичні методи управління виявляються недостатньо ефективними для забезпечення централізованого контролю, відтворюваності конфігурацій, автоматизації процесів та швидкого реагування на зміну навантажень і бізнес-вимог.

У цьому контексті підхід DevOps (Development & Operations), що передбачає безперервну інтеграцію, доставку та тісну взаємодію між розробниками та операційними фахівцями, розглядається як ключовий механізм оптимізації життєвого циклу IT-інфраструктури. Інтеграція DevOps-практик з інструментами Infrastructure as Code, хмарними API та системами моніторингу дозволяє не лише автоматизувати розгортання та управління серверними ресурсами, але й зменшити витрати на обслуговування та мінімізувати вплив людського чинника на стабільність систем.

Особливу значимість така технологія набуває у сферах, де швидкість розгортання, масштабованість і безпека є критичними: в освітніх та наукових установах, державних інформаційних системах, електронній комерції, а також у компаніях з динамічними IT-навантаженнями та обмеженими ресурсами. В умовах посилення кіберзагроз і зростання складності інфраструктури здатність IT-систем до оперативної адаптації, резервування та безпечного функціонування стає критичною.

Таким чином, актуальність дослідження визначається необхідністю розробки інформаційної технології автоматизованого управління серверною інфраструктурою, яка поєднує практики DevOps, можливості хмарних платформ, інструменти автоматизації конфігурацій та централізоване управління ресурсами. Запропонована технологія повинна забезпечувати

масштабованість, інтеграцію з різними екосистемами та ефективне функціонування у гібридних середовищах з різними обмеженнями ресурсів і часу.

Аналіз останніх досліджень і публікацій свідчить, що автоматизація управління IT-інфраструктурою, зокрема підходи DevOps та практики Infrastructure as Code (IaC), є предметом активного наукового інтересу. У публікаціях останніх років розглядаються практичні аспекти використання IaC для управління ресурсами у гібридних хмарних середовищах, включно з питаннями узгодження конфігурацій, виявлення дрейфу станів та інтеграції у CI/CD-потоки, що дозволяє досягати відтворюваності та послідовності розгортань.

Зокрема, у роботі [1], розглянуто сучасні тенденції автоматизації управління інфраструктурою та контейнеризації додатків, що дозволяє підвищити гнучкість, масштабованість і надійність хмарних систем. Дослідження авторів [2] присвячене автоматизації розгортання серверлес-застосунків із використанням підходу Infrastructure as Code, що підтверджує ефективність автоматизації у реальних проєктах.

У роботі [3] запропоновано модель реінжинірингу серверної інфраструктури із застосуванням Terraform та Ansible, що забезпечує підвищення продуктивності, надійності та зниження експлуатаційних витрат. Праця [4] присвячена аналізу сучасних методів автоматизації управління хмарною інфраструктурою та впровадженню ефективних стратегій оптимізації використання хмарних ресурсів. Автори детально досліджують підходи до моделювання обчислювальних та мережевих ресурсів у хмарних середовищах, оцінюють їх продуктивність і вартість, а також розглядають інструменти для автоматизованого розгортання та масштабування сервісів, зокрема Terraform та платформи Azure Cloud.

Дослідження [5] демонструє ефективність інтеграції хмарних сервісів і програмно-визначених мереж для підвищення відмовостійкості та масштабованості інфраструктури. У роботі [6] проведено аналіз використання

хмарних сервісів для оптимізації системного адміністрування, що підтверджує переваги автоматизації та стандартизації процесів управління.

Окрему увагу приділено дослідженням автоматизації міграції програмного забезпечення у хмарні середовища із використанням інструментів Infrastructure as Code, зокрема Terraform у середовищі AWS, що розглянуто у роботі [7].

Серед зарубіжних дослідників, які зробили вагомий внесок у розвиток DevOps та автоматизації інфраструктури, слід відзначити роботи [8–12], спеціалістів яких заклали теоретичні та практичні основи сучасних підходів до автоматизації управління IT-інфраструктурою.

У статті [13] досліджено ефективність застосування методології Infrastructure as Code (IaC) для створення, масштабування та управління хмарною інфраструктурою. Проведено порівняльний аналіз сучасних інструментів IaC, зокрема Terraform, Pulumi, AWS CloudFormation та Ansible, за критеріями автоматизації, масштабованості, швидкості розгортання, гнучкості та надійності конфігурацій.

Отримані результати досліджень зазначених авторів знаходять практичне відображення у сучасних рішеннях автоматизації управління інфраструктурою та підтверджуються результатами прикладних досліджень.

Наприклад, у роботі спеціалістів [14] з університету Мічіган, система NSync використовує AI-агентів для автоматичного виявлення дрейфу через моніторинг API, досягаючи 97% точності в реконциляції, що зменшує ручне втручання та підвищує надійність у гібридних хмарах AWS. Окремі дослідження [15-16] присвячені оцінці ефективності методів IaC у створенні, масштабуванні та управлінні хмарною інфраструктурою, що підкреслює значну роль цієї методології у цифровій трансформації IT-середовищ.

Зокрема, при розрахунках в даній роботі, використання IaC зменшує залежність від людського фактора, покращує швидкість розгортання на 85% та надійність конфігурацій до 98%, сприяючи DevOps та CI/CD. У наукових працях [17-19] також аналізуються інструментальні засоби, такі як Terraform

та Ansible, у контексті їхнього об'єднання для управління гібридними середовищами, що розкриває потенціал інтегрованих конвеєрів автоматизації для забезпечення безперервного життєвого циклу інфраструктури.

Такий огляд літератури демонструє наявність наукового фундаменту для подальшого розвитку автоматизованих технологій управління серверною інфраструктурою, але одночасно виявляє пробіли у повній інтеграції рішень для гібридних середовищ, таких як обмежена підтримка мультихмар та наміри дрейфу. Дисертаційна робота виконана в рамках наукових досліджень з напрямів комп'ютерні науки та автоматизації IT-систем управління.

Метою дисертаційної роботи - розробка та обґрунтування інформаційної технології автоматизованого управління серверною інфраструктурою в гібридних середовищах на основі DevOps-підходів, що забезпечує централізацію, відтворюваність, масштабованість і підвищення ефективності процесів управління інфраструктурними середовищами.

Основні завдання дисертації:

1. Проаналізувати сучасний стан і тенденції автоматизації управління серверною інфраструктурою, зокрема з використанням DevOps-інструментів, хмарних платформ та підходів Infrastructure as Code, для обґрунтування необхідності автоматизації в гібридних середовищах.
2. Виявити обмеження існуючих підходів і рішень щодо управління серверною інфраструктурою в умовах гібридних середовищ з точки зору централізації, масштабованості та керованості процесів.
3. Обґрунтувати концепцію інформаційної технології автоматизованого управління серверною інфраструктурою, яка інтегрує DevOps-практики, веб-орієнтовані засоби управління та механізми взаємодії з хмарними API.
4. Розробити архітектурно-алгоритмічні рішення для автоматизованого розгортання, конфігурації та моніторингу серверних ресурсів у локальних, хмарних і гібридних середовищах.

5. Розробити та програмно реалізувати інформаційну технологію автоматизованого управління серверною інфраструктурою в гібридних середовищах, особливістю якої є інтеграція інфраструктури як коду, безперервної інтеграції та розгортання, механізмів взаємодії з API хмарних провайдерів і централізованого веб-управління в єдину систему.

6. Провести експериментальне дослідження ефективності розробленої інформаційної технології з використанням показників часу розгортання, стабільності конфігурацій та експлуатаційної надійності.

7. Здійснити практичну апробацію та впровадження результатів роботи у науково-дослідну діяльність і навчальний процес закладів вищої освіти.

8. Сформулювати практичні рекомендації щодо впровадження запропонованої технології в комерційних IT-системах та визначити напрями її подальшого розвитку, зокрема у сфері інтелектуальної автоматизації на основі DevOps.

Об'єктом дослідження є процеси управління серверною інфраструктурою сучасних інформаційних систем у локальних, хмарних та гібридних середовищах.

Предметом дослідження є методи, моделі та засоби автоматизації управління серверною інфраструктурою на основі DevOps-практик, зокрема інфраструктури як коду, безперервної інтеграції та розгортання, а також механізмів взаємодії з API хмарних провайдерів у гібридних середовищах.

Методи дослідження базуються на використанні комплексу теоретичних та практичних методів: моделювання, системний аналіз, логіко-структурне проектування, розробка програмного забезпечення, експериментальне тестування та порівняльний аналіз рішень. Також застосовуються емпіричні методи, що передбачають створення гетерогенного тестового середовища, вимірювання продуктивності за допомогою вбудованих засобів профілювання мови Python та оцінку результатів у реальних сценаріях.

У роботі використовуються наступні моделі:

Основними засобами та інструментами автоматизації в роботі є:

- Terraform — засіб ініціалізації ресурсів за декларативним підходом;
- Ansible — система управління конфігураціями для налаштування програмного оточення;
- Django та Python — високорівневий вебфреймворк та мова програмування для розробки централізованої панелі керування;
- HashiCorp Vault — засіб безпечного зберігання облікових даних на основі конвертного шифрування;
- API хмарних провайдерів (Hetzner Cloud, AWS, Azure, Google Cloud) та DNS-провайдерів (CloudFlare) для динамічної взаємодії з інфраструктурою.

Для автоматизації повного циклу розгортання серверних ресурсів розроблено алгоритм комплексного конвеєра CPASI (Complex Pipeline for Automated Server Infrastructure), що базується на механізмі атомарних транзакцій інфраструктури.

Наукова новизна отриманих результатів полягає у розв'язанні науково-прикладної задачі формалізації та розробки інформаційної технології автоматизованого управління серверною інфраструктурою в гібридних середовищах на основі DevOps-підходів, що відрізняється інтеграцією інфраструктури як коду, безперервної інтеграції та розгортання, механізмів взаємодії з API хмарних провайдерів і централізованого веб-управління в єдину керовану систему.

- **Уперше:** Обґрунтовано та сформульовано концепцію інформаційної технології як цілісного науково-прикладного об'єкта, що інтегрує практики Infrastructure as Code, CI/CD та механізми взаємодії з хмарними API в єдину керовану систему. Це дозволило перейти від фрагментарного використання інструментів до наскрізного управління життєвим циклом інфраструктури, забезпечивши приріст загальної ефективності управління на 44%.

- **Уперше:** Розроблено алгоритм CPASI (Complex Pipeline for Automated Server Infrastructure), який, на відміну від існуючих, забезпечує атомарність транзакцій («все або нічого») при одночасному розгортанні віртуальних ресурсів, налаштуванні DNS та конфігурації ПЗ. Завдяки впровадженню паралелізму в алгоритмі у рамках експериментального дослідження час розгортання кластера з 10 серверів було зменшено з приблизно 2,5 годин у разі ручної конфігурації до близько 15 хвилин при використанні запропонованого алгоритму, що відповідає прискоренню приблизно у 10 разів.
- **Уперше:** Запропоновано архітектурно-алгоритмічну модель реалізації технології, побудовану на трирівневій абстракції (інфраструктурній, операційній та сервісній). Це дозволило уніфікувати управління гетерогенними ресурсами незалежно від їх локації та забезпечити провайдер-агностичність системи.
- **Удосконалено:** Підходи до автоматизації управління гібридними середовищами шляхом впровадження механізмів делегованого виконання команд через захищені SSH-тунелі. Це дозволило включити локальні сервери за NAT-шлюзами у загальний контур автоматизації без зміни існуючих мережевих політик безпеки.
- **Удосконалено:** Методи оцінки ефективності розгортання інфраструктури через введення інтегральних показників, які враховують не лише час операцій, а й ймовірність помилок та стабільність конфігурацій. Встановлено, що автоматизація знижує ймовірність помилок оператора з 5% до менш ніж 2%.
- **Дістало подальшого розвитку:** Теоретичні засади застосування DevOps як інформаційної технології управління в умовах цифрової трансформації, зокрема через перехід від статичного адміністрування до моделей «інфраструктури на вимогу» (IaS). Це забезпечило економічну доцільність навчання, знизивши вартість утримання лабораторних середовищ у 15 разів.

- **Дістало подальшого розвитку:** Методи предиктивного моніторингу інфраструктури шляхом обґрунтування застосування алгоритму Isolation Forest для виявлення аномалій у багатовимірних метриках серверів. Математично обґрунтовано, що цей метод дозволяє ідентифікувати приховані «сірі збої» з ймовірністю понад 80%, які ігноруються традиційними пороговими системами.

Розроблену комп'ютерну програму, та алгоритм, зареєстровано як об'єкти авторського права (сертифікати УКРІНОВІ №144242 та №141764), що підтверджує оригінальність програмних рішень і право автора на використання результатів у навчальному та науковому процесі.

Особистий внесок здобувача. Дисертаційне дослідження було виконане здобувачем самостійно, всі сформульовані в роботі положення, висновки та рекомендації були обґрунтовані особистими дослідженнями автора. Окремі положення були аргументовані з використанням робіт інших науковців, що мають відповідні посилання в тексті роботи. В індивідуальних наукових роботах використані лише авторські напрацювання та ідеї. Автор дисертації активно брав участь у наукових дискусіях та написанні наукових статей, що були опубліковані за темою дисертації. Також автор доповідав результати досліджень на міжнародних наукових конференціях. У роботі не використовувалися напрацювання співвиконавців дослідницьких проєктів. Результатом всіх зазначених наукових робіт стало написання матеріалів дисертаційної роботи.

Практичне значення одержаних результатів полягає у розробці та впровадженні цілісної інформаційної технології, що дозволяє автоматизувати повний цикл управління серверною інфраструктурою в гібридних середовищах. Основні результати дослідження спрямовані на вирішення прикладних задач адміністрування, що забезпечує підвищення ефективності, масштабованості та стабільності сучасних IT-систем.

Ключові аспекти практичної цінності роботи:

- Створено та впроваджено веб-орієнтовану систему централізованого управління, яка дозволяє об’єднувати локальні фізичні сервери та хмарні ресурси різних провайдерів у єдиний керований контур без необхідності встановлення додаткового клієнтського програмного забезпечення.
- Розроблено та апробовано алгоритм CPASI, який мінімізує операційне навантаження на адміністраторів, забезпечуючи автоматичне розгортання та конфігурацію інфраструктури, що дозволяє скоротити час виконання типових операцій на 44%.
- Реалізовано механізми безпечної оркестрації та захищеного SSH-тунелювання, що дає змогу керувати серверами, які знаходяться за NAT-шлюзами, без порушення існуючих політик мережевої безпеки організацій.
- Запропоновано модель «інфраструктури на вимогу» для освітнього процесу, яка дозволяє розгортати лабораторні середовища протягом 5 хвилин з мінімальними фінансовими витратами (біля 7,5 грн за заняття), яку підтверджено теоретичними розрахунками, що оцінюють час розгортання, використання ресурсів та вартість
- Сформульовано методичні рекомендації щодо вибору мультихмарних стратегій та використання провайдер-агностичних інструментів (Terraform, Ansible), що дозволяє організаціям уникати залежності від конкретних постачальників послуг.

Результати дисертаційного дослідження інтегровані в науково-дослідні роботи ХНУ імені В. Н. Каразіна, а також знайшли практичне застосування у діяльності ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС». Підтвердженням впровадження результатів роботи є відповідна довідка, наведена у додатку до дисертації.

Апробація результатів дисертації. Основні результати дисертаційного дослідження обговорювалися на засіданнях кафедри комп’ютерних наук, інформаційних технологій та програмної інженерії, а також презентувалися на науково-практичних конференціях, зокрема:

1. Науково-технічна міжнародна конференція «Комп'ютерне моделювання у наукоємних технологіях (КМНТ – 2023)». – Харків, 2023.

2. Науково-технічна міжнародна конференція «Комп'ютерне моделювання у наукоємних технологіях (КМНТ – 2024)». – Харків, 2024.

3. Інтелектуальні технології у міждисциплінарних дослідженнях : матеріали Міжнародної науково-практичної конференції. – Харків, 2025.

4. ProfIT AI 2025 : Proceedings of the International Conference. – Kharkiv, 2025.

Результати роботи можуть бути використані в навчальному процесі, в межах дисциплін із комп'ютерних наук, програмної інженерії та інформаційних технологій, а також у реальних проектах, що передбачають управління хмарною та гібридною інфраструктурою.

Публікації результатів дисертації. Основні наукові та практичні результати досліджень опубліковано у 8 роботах, серед яких: 4 статті у наукових фахових виданнях України, 4 – у матеріалах науково-практичних конференцій

Структура та обсяг дисертації. Дисертаційна робота складається з анотації двома мовами, чотирьох розділів, висновків та списку використаних джерел. Загальний обсяг дисертації складає 135 сторінок, у тому числі: таблиці - 9 (на 9 сторінках) та рисунки - 26. Список використаних джерел налічує 126 найменувань (на 16 сторінках), у тому числі іноземною мовою - 99.

РОЗДІЛ 1. СУЧАСНИЙ СТАН АВТОМАТИЗАЦІЇ УПРАВЛІННЯ СЕРВЕРНОЮ ІНФРАСТРУКТУРОЮ

1.1 Аналіз існуючих рішень та інструментів

Сучасна парадигма побудови інформаційних систем характеризується переходом від статичних апаратних конфігурацій до динамічних програмно-визначених середовищ (Software-Defined Infrastructure). У цьому контексті автоматизація управління серверною інфраструктурою трансформується з допоміжного засобу адміністрування у критичний компонент забезпечення життєвого циклу IT-систем. Науково-технічна проблема полягає не стільки у наявності інструментарію, скільки у відсутності єдиних підходів до інтеграції гетерогенних компонентів — локальних серверів, віртуальних машин та хмарних сервісів — у цілісну керовану систему.

Аналіз наукових джерел [20, 21] свідчить, що еволюція засобів автоматизації відбувається у напрямку підвищення рівня абстракції: від імперативних скриптів до декларативних моделей опису інфраструктури (Infrastructure as Code — IaC). Існуючі рішення можна класифікувати за функціональним призначенням на три основні групи:

- засоби ініціалізації ресурсів (Provisioning),
- системи управління конфігураціями (Configuration Management)
- платформи оркестрації.

Ключовим аспектом дослідження є порівняльний аналіз підходів до управління станом системи. Інструменти ініціалізації, такі як Terraform, використовують декларативний підхід, де цільовий стан інфраструктури описується у вигляді графа залежностей ресурсів. Ефективність цього підходу при розгортанні хмарних ресурсів визначається можливістю передбачення змін та мінімізації «дрейфу конфігурацій». Математично задачу підтримки актуального стану інфраструктури S можна представити як мінімізацію функції відхилення реального стану S_{real} від цільового S_{target} у момент часу t:

$$\min \ \Delta\left(\mathrm{S_{target}(t)}, \mathrm{S_{real}(t)}\right) \rightarrow 0 \qquad (1.1)$$

Формула 1.1 - моделі підтримки цільового стану інфраструктури

де функція Δ визначає розбіжність параметрів конфігурації. Формула формалізує ціль автоматизації задає критерій оптимальності та визначає, що система є коректною, якщо $\Delta = 0$ або $\Delta < \varepsilon$, де ε це малий поріг точності**,** який використовується замість ідеального нуля в реальних системах. Однак, обмеженням існуючих IaC-інструментів є їхня орієнтація переважно на API хмарних провайдерів без глибокої інтеграції з операційними системами (Bare Metal), що є критичним для гібридних середовищ [22].

Системи управління конфігураціями (Ansible, Chef, Puppet) вирішують задачу налаштування програмного оточення всередині серверів. На відміну від агентних систем (Puppet, Chef), без-агентний підхід Ansible, що базується на протоколі SSH та ідемпотентних модулях, демонструє вищу гнучкість у гетерогенних мережах [23]. Проте, при масштабуванні інфраструктури виникає проблема лінійного зростання часу виконання конфігураційних сценаріїв T_{exec}, що залежить від кількості вузлів N .

$$T_{exec} \approx \sum_{i=1}^{N} \mathrm{C}(n_i) + L_{network}(N) \qquad (1.2)$$

Формула 1.2 - Модель часу виконання конфігураційних сценаріїв у безагентних системах управління конфігураціями

де $\mathrm{C}(n_i)$ — обчислювальна складність налаштування i -го вузла, а $L_{network}(N)$ — латентність мережі яка також залежить від кількості вузлів N. Це створює "вузьке місце" при управлінні великими кластерами без централізованого механізму паралелізації та черги завдань.

Окремою проблемою є фрагментація екосистеми хмарних провайдерів (AWS, Azure, GCP, Hetzner) які також були досліджені іншими авторами

робіт [24, 25]. Попри наявність стандартизованих протоколів (REST, gRPC), кожен провайдер реалізує власні специфікації API для управління ресурсами. Це призводить до виникнення проблеми прив'язки до постачальника та ускладнює побудову мультихмарних архітектур. Існуючі інструменти надають абстракції (наприклад, Providers у Terraform), однак вони не забезпечують єдиного інтерфейсу для оперативного моніторингу та управління життєвим циклом серверів у реальному часі через веб-інтерфейс, що вимагає від адміністраторів використання консольних команд та розрізнених панелей керування.

Схематично поточний стан автоматизації та розрив між інструментами можна зобразити наступним чином (Рис. 1.1).

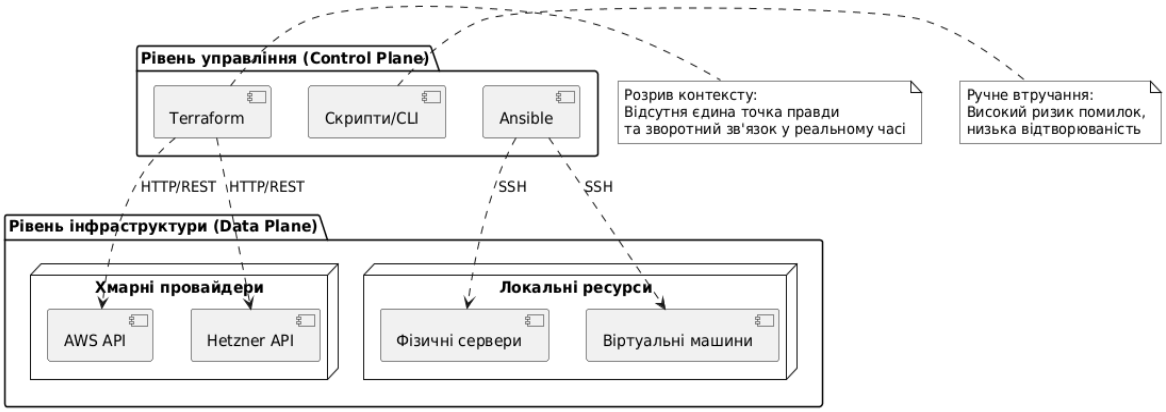

Рис. 1.1— Схема фрагментованої взаємодії інструментів автоматизації в гібридних середовищах

Аналіз показує, що хоча ринок насичений потужними спеціалізованими інструментами, спостерігається концептуальний розрив між засобами ініціалізації (IaC) та засобами конфігурації. Відсутність інтегрованої інформаційної технології, яка б об'єднувала ці процеси під єдиним веб-інтерфейсом з підтримкою рольової моделі доступу та аудиту, що буде призводити до підвищення операційних витрат та ризиків безпеки.

Таким чином, аналіз існуючих рішень дозволяє виділити такі системні недоліки:

1. Фрагментарність управління: Необхідність використання різних інструментів для створення серверів (Terraform) та їх налаштування (Ansible) без єдиного центру оркестрації [26, 27].

2. Складність інтеграції гібридних середовищ: Відсутність готових рішень для безшовної інтеграції локальних серверів та бюджетних хмарних провайдерів (наприклад, Hetzner) у єдиний контур управління [28].

3. Високий поріг входження: Ефективне використання існуючих інструментів вимагає глибоких знань специфічних мов опису (HCL, YAML) та навичок роботи з командним рядком, що ускладнює делегування задач.

Зазначені обмеження обумовлюють необхідність розробки нової інформаційної технології, яка забезпечить автоматизацію повного циклу управління серверною інфраструктурою шляхом інтеграції DevOps-інструментів, API хмарних провайдерів та веб-технологій.

1.1.1 **Огляд сучасних технологій і методів управління серверною інфраструктурою**

Еволюція підходів до адміністрування обчислювальних ресурсів характеризується переходом від імперативних методів ручного керування до декларативних моделей, що об’єднуються концепцією «Інфраструктура як код» (Infrastructure as Code — IaC). Сучасна методологія управління базується на принципі ідемпотентності, який гарантує, що багаторазове виконання однієї й тієї ж операції приводить систему до незмінного кінцевого стану, незалежно від початкових умов.

В основу сучасних технологій покладено розподіл відповідальності між інструментами ініціалізації та управління конфігураціями. Аналіз літературних джерел дозволяє стверджувати, що найбільш ефективною є гібридна модель, де інструменти на кшталт Terraform відповідають за створення ресурсів (IaaS), а системи типу Ansible — за налаштування програмного оточення. Формула ефективності автоматизації розгортання (E_{auto}) відображає обернену залежність між ефективністю процесу, часом розгортання (T_{deploy}) та ймовірністю виникнення помилок (P_{error}).

$$E_{auto} = 1 / (T_{deploy} \cdot (1 + P_{error})) \quad (1.3)$$

Формула 1.3 - ефективності автоматизації розгортання

Згідно з емпіричними даними розрахованими у статтях [29, 30], застосування IaC дозволяє скоротити час розгортання на 50–70% порівняно з ручним налаштуванням, при цьому ймовірність помилок знижується до значень, близьких до нуля, завдяки відсутності людського фактора.

Критичним аспектом сучасних методів є управління станом інфраструктури (State Management). Декларативні інструменти зберігають поточний стан об'єктів у спеціалізованих файлах (state files), що дозволяє виявляти несанкціоновані зміни («дрейф конфігурацій») та автоматично повертати систему до еталонного вигляду. Процес узгодження стану можна представити у вигляді діаграми діяльності (Activity Diagram), що відображає логіку роботи IaC-агента (Рис. 1.2).

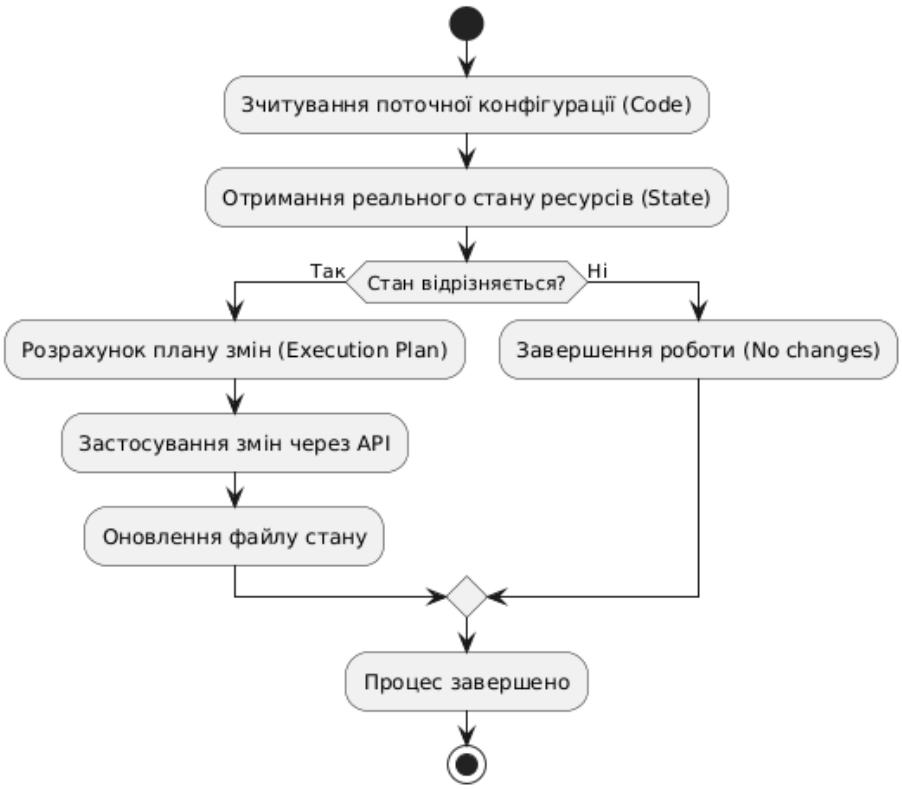

Рис. 1.2. — Алгоритм узгодження стану інфраструктури в декларативних системах

Поряд із IaC, важливим технологічним трендом є контейнеризація та оркестрація (Docker, Kubernetes), які дозволяють абстрагуватися від операційної системи та керувати додатками як ізольованими одиницями розглядаються в роботах [31-33]. Однак, впровадження таких систем у гібридних середовищах часто ускладнюється необхідністю підтримки застарілих додатків, які не можуть бути контейнеризовані без значного рефакторингу архітектури, по даній темі слід виділити декілька статей [34-36] в яких спеціалісти демонструють свої підходи, а також практичні кейс-стаді з реальним legacy-проєктом.

Окремим напрямом еволюції засобів управління серверною інфраструктурою є розвиток веб-орієнтованих інтерфейсів (Web-based Management Interfaces), написана стаття [37] підкреслює потреби у централізованому веб-інтерфейсі для управління фізичними та хмарними серверами. У роботах [38-40] які розглядають системи моніторингів, що забезпечують візуалізацію стану системи та можливість віддаленого адміністрування без необхідності встановлення спеціалізованого клієнтського програмного забезпечення. Актуальність таких рішень обумовлена необхідністю зниження порогу входження для операторів системи та забезпечення мобільності управління .

На ринку програмного забезпечення представлено кілька класів веб-панелей, які відрізняються за рівнем абстракції та цільовим призначенням. До першої групи належать інструменти системного адміністрування, такі як Webmin та Cockpit [41, 42]. Система Webmin, (рис. 1.3) написана на Perl, дозволяє конфігурувати операційну систему (користувачі, дискові квоти, сервіси, конфігураційні файли) через веб-інтерфейс, проте її архітектура є застарілою і складно інтегрується у сучасні DevOps-конвеєри. Більш сучасним аналогом є проект Cockpit (рис. 1.4), що розвивається за підтримки Red Hat. Він використовує системні API (systemd, DBus) для моніторингу та управління серверами в реальному часі, надаючи зручний графічний інтерфейс для перегляду логів, управління контейнерами та мережевими налаштуваннями.

Проте, суттєвим обмеженням Cockpit є його орієнтація переважно на управління окремими вузлами, що ускладнює його використання для оркестрації великих кластерів у гібридних середовищах. У роботі [43] показані також інші аспекти схожих веборієнтованих систем управління сервером під час навчання майбутніх фахівців.

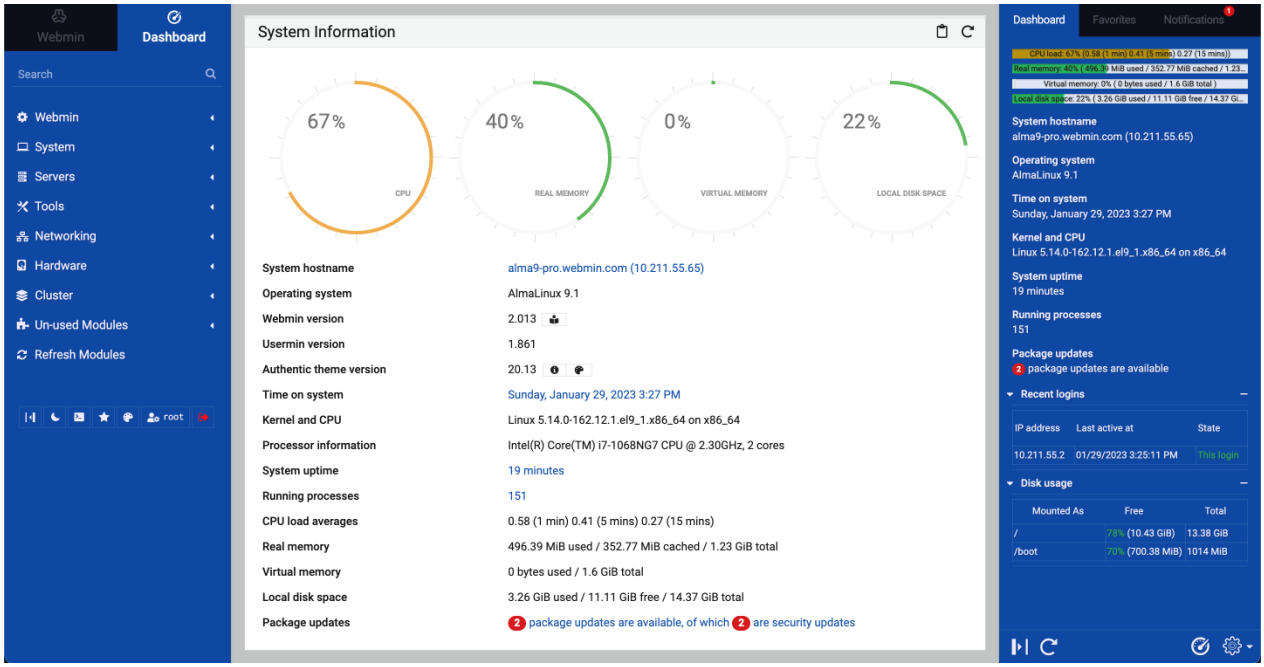

Рис. 1.3. — зовнішній вигляд системи webmin

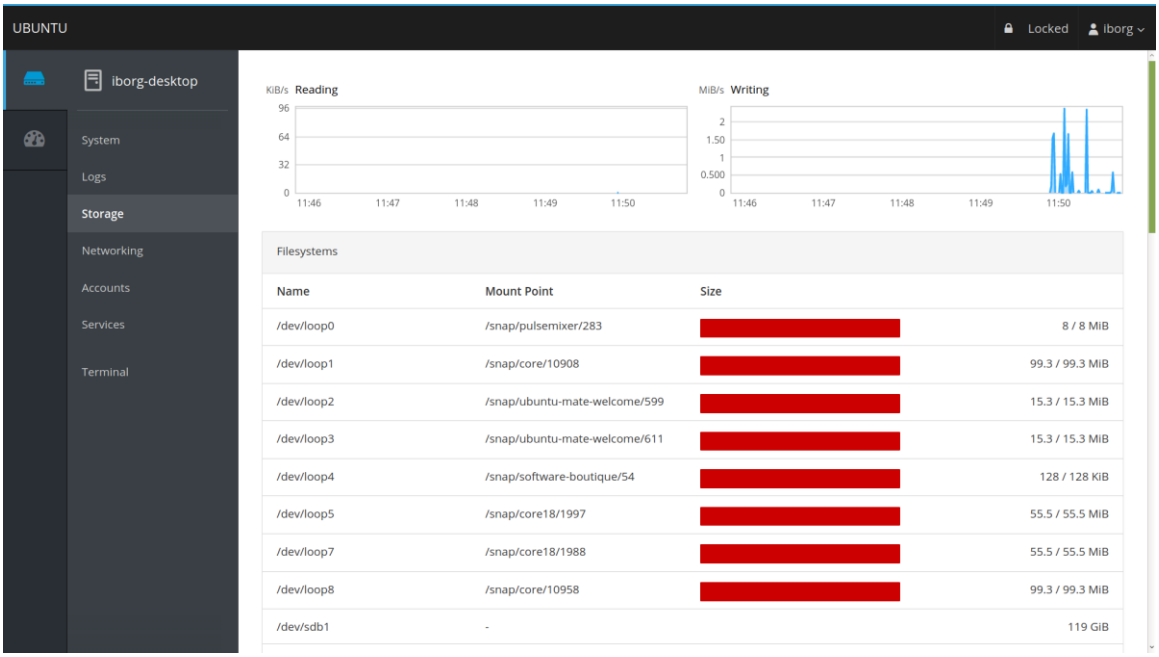

Рис. 1.4. — зовнішній вигляд системи cockpit

Другу групу складають комерційні панелі управління хостингом, такі як cPanel (рис. 1.5), Plesk (рис. 1.6) та DirectAdmin [44 - 46]. Автори [47,48] в своїх статтях які зосереджені на панелях керування, містять порівняння з комерційними панелями (cPanel, Plesk) та обговорюють архітектуру веб-хостингів та їх безпеку. Ці рішення забезпечують високий рівень автоматизації рутинних завдань (управління веб-серверами, базами даних, поштовими скриньками), однак вони є пропрієтарними, мають високу вартість ліцензування та надлишковий функціонал для задач, що не пов'язані з наданням послуг shared-хостингу. Крім того, вони часто вносять значні зміни у конфігурацію операційної системи, що конфліктує з принципами імутабельності інфраструктури (Immutable Infrastructure).

Порівняльний аналіз хмарного та власного хостингу за критеріями безпеки, зокрема генерації ключів (PKI), автентифікації та фільтрації спаму, показав, що обидва підходи мають специфічні вразливості провів спеціаліст у статті [49].

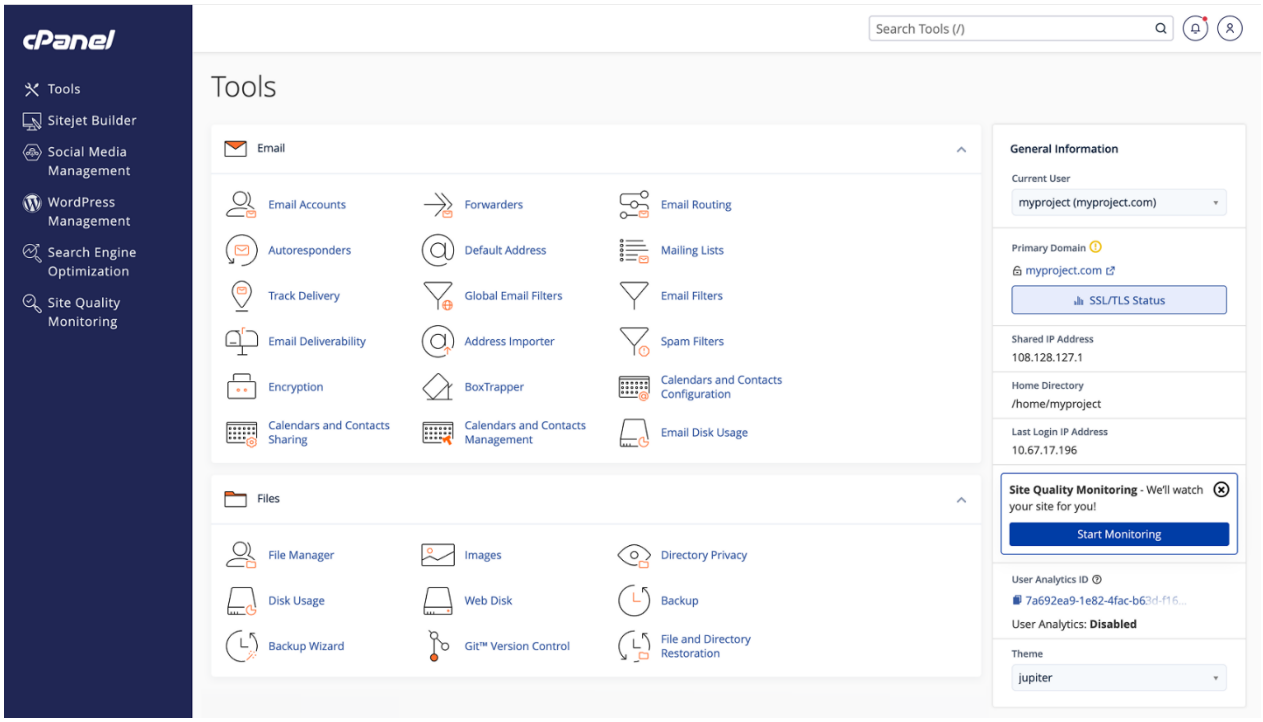

Рис. 1.5. — зовнішній вигляд системи cpanel

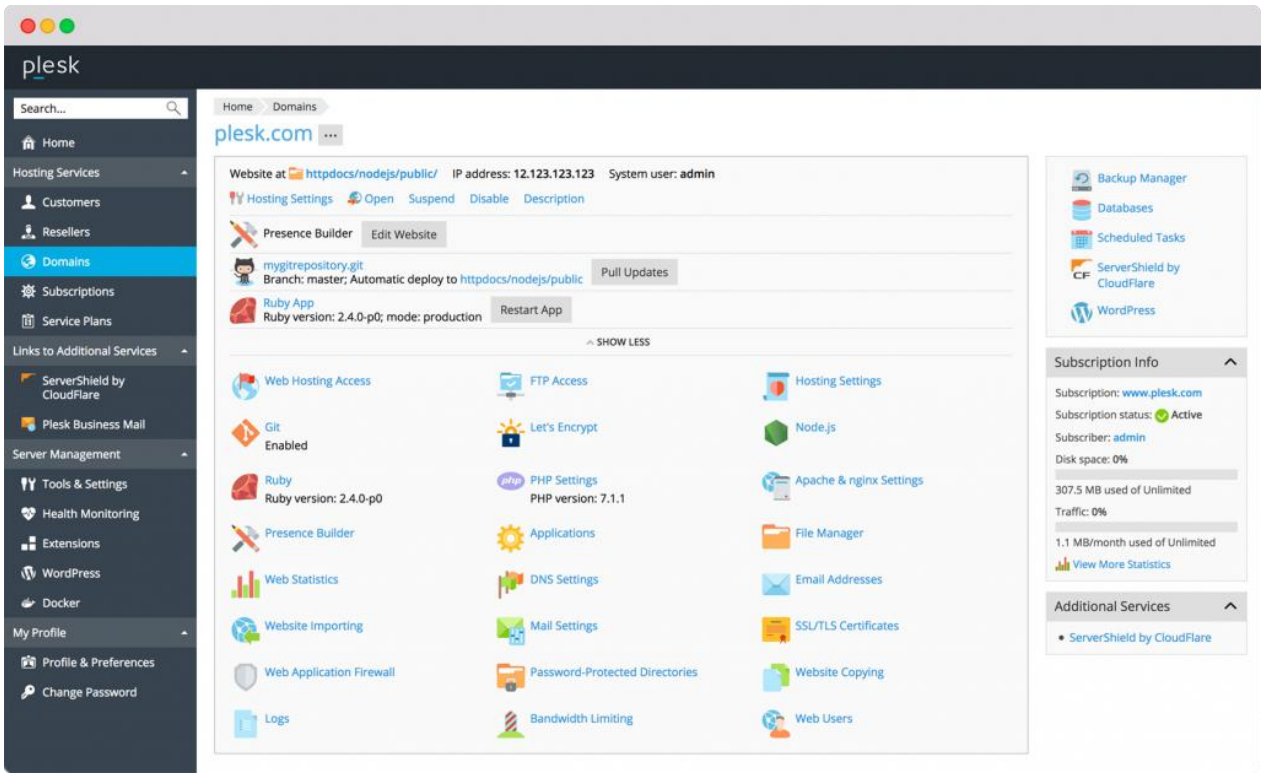

Рис. 1.6. — зовнішній вигляд системи plesk

Важливим технологічним аспектом реалізації веб-управління є забезпечення доступу до консолі сервера (CLI) безпосередньо через браузер. Сучасні рішення використовують технологію WebSockets для створення двонаправленого каналу зв'язку між клієнтом та сервером, транслюючи потік даних SSH у веб-інтерфейс. Реалізації на базі бібліотек xterm.js або шлюзів типу Apache Guacamole дозволяють емулювати повноцінний термінал, підтримуючи інтерактивні сесії [50, 51]. Такий підхід значно підвищує безпеку, оскільки дозволяє централізувати доступ, вести аудит команд та уникати необхідності відкриття SSH-портів для прямого доступу з публічних мереж, у роботі [52] автори проводять порівняння методик оцінки ризиків безпеки для веб-додатків, зокрема рекомендації щодо багатофакторної автентифікації та мінімізації часу сесій.

Проте аналіз існуючих веб-рішень виявляє суттєву прогалину: на ринку фактично відсутні системи, які б поєднували можливості низькорівневого адміністрування (Web-terminal), управління життєвим циклом інфраструктури (через інтеграцію з Terraform/Ansible) та підтримку гібридних хмар у єдиному

інтерфейсі. Більшість існуючих CMP-платформ (Cloud Management Platforms), таких як OpenStack Horizon або ManageIQ, є надмірно складними для розгортання у малих та середніх проектах, про це, також говорять дослідники, у своїх дослідженнях [53, 54]. Які прямо вказують, що розгортання OpenStack є складним процесом із кривою навчання, і пропонують фреймворк автоматизації як спосіб зменшити час у 4 рази, тоді як прості панелі не забезпечують необхідного рівня автоматизації. Це підтверджує доцільність розробки спеціалізованої інформаційної технології, яка б заповнила цю нішу.

1.1.2 **Особливості використання хмарних провайдерів**

У сучасній науковій літературі [55,56] вибір хмарного провайдера розглядається як багатокритеріальна задача прийняття рішень, що базується на аналізі техніко-економічних показників, архітектурних особливостей та вимог до безпеки даних. Порівняльна характеристика хмарних платформ для автоматизації інфраструктури показана у таблиці (1.1), а також у дослідженні [57]. Ринок хмарних обчислень характеризується високим рівнем концентрації, де основна частка належить так званим «гіперскейлерам» (Hyperscalers), за даними дослідження Gartner [58], у 2024 році п'ять найбільших провайдерів контролювали 82,1% ринку, проте спостерігається стійка тенденція до зростання популярності альтернативних провайдерів, що пропонують оптимізовані за вартістю рішення [59].

Домінуючу позицію на ринку займають платформи Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Згідно з дослідженнями [60], ці платформи пропонують найбільш повний спектр сервісів (понад 200 найменувань), що охоплюють моделі IaaS, PaaS, SaaS та FaaS (Function as a Service).

- Amazon Web Services (AWS) є технологічним лідером, що забезпечує найвищий рівень доступності та глобального покриття. Ключовою перевагою AWS є зріла екосистема та наявність спеціалізованих інструментів для кожного аспекту інфраструктури (наприклад, AWS Lambda для безсерверних обчислень, Aurora для баз даних). Однак, науковці [61]

відзначають високу складність білінгових моделей AWS, де кінцева вартість формується з безлічі мікро-транзакцій (обчислювальний час, операції вводу/виводу, трафік, запити API), що ускладнює прогнозування витрат.

- Microsoft Azure орієнтований на корпоративний сегмент та гібридні сценарії. Його архітектурною особливістю є тісна інтеграція з продуктами Microsoft та наявність рішень Azure Stack для розгортання хмарних сервісів у локальних дата-центрах. Це робить його оптимальним вибором для організацій, що проходять етап цифрової трансформації наявних активів [62].
- Google Cloud Platform (GCP) вирізняється інноваційними підходами у сфері аналітики великих даних та машинного навчання (Vertex AI, BigQuery). Дослідження [63] показують, що GCP часто пропонує кращі показники продуктивності мережі завдяки власній глобальній оптоволоконній інфраструктурі, проте має меншу частку ринку порівняно з конкурентами.

Для задач, що вимагають високої економічної ефективності обчислень без необхідності використання специфічних PaaS-сервісів, доцільним є використання альтернативних провайдерів, таких як Hetzner Online та DigitalOcean. У роботі [64], присвяченій оптимізації витрат на хмарну інфраструктуру та досвідом роботи з Hetzner, виділяється, що ці платформи забезпечують значно краще співвідношення «ціна/продуктивність» для базових ресурсів (Compute, Storage).

- Hetzner Online є прикладом провайдера, що фокусується на наданні «чистих» обчислювальних потужностей (Bare Metal, VPS) за цінами, що можуть бути на 50–70% нижчими за аналоги у гіперскейлерів [65].
- Дослідження [66-68] показують, що продуктивність віртуальних машин у хмарних середовищах суттєво залежить від рівня мультиорендності, що призводить до варіативності затримок і продуктивності дискових операцій. Використання NVMe-накопичувачів у віртуалізованих середовищах дозволяє досягати продуктивності, близької до bare-metal, з мінімальними накладними витратами та зниженням затримки. Додатково, дослідження мережевої

продуктивності в хмарах [69], показують, що варіативність мережевих затримок є важливим фактором, який впливає на ефективність розподілених систем.

- DigitalOcean позиціонується як платформа, орієнтована на розробників, з акцентом на простоту використання та передбачуваність ціноутворення. На відміну від AWS, DigitalOcean пропонує фіксовані тарифні плани та спрощений API, що знижує поріг входження для автоматизації. Наукові порівняння [70,71] вказують на переваги DigitalOcean у сценаріях стартапів та малого бізнесу, де критичним є контроль операційних витрат та швидкість розгортання.

Таблиця 1.1

Порівняльна характеристика хмарних платформ для автоматизації інфраструктури

Провайдер	Тип архітектури	Складність API	Підтримка Terraform/Ansible	Економічна ефективність (Cost Efficiency)
AWS	Hyperscale, Enterprise	Висока	Повна (Native Providers)	Низька (для малих проектів)
Azure	Enterprise, Hybrid	Висока	Повна	Середня
GCP	Data-centric	Середня	Повна	Висока (Spot instances)
Hetzner	IaaS-focused	Низька	Підтримується (Community Providers)	Максимальна

Основною проблемою при використанні хмарних провайдерів у гібридних системах є відсутність стандартизації API. Кожен провайдер використовує власні структури даних для опису ідентичних сутностей (наприклад, aws_instance проти hcloud_server у Terraform), що унеможливлює просту міграцію або балансування навантаження між хмарами без додаткового шару абстракції [72]. Проведений аналіз дозволяє стверджувати, що жоден із провайдерів не є універсальним рішенням. Виникає науково-прикладна проблема оптимізації архітектури, яка вирішується шляхом застосування мультихмарної стратегії (Multi-cloud). У такій моделі AWS або GCP

використовуються для високотехнологічних задач (ШІ, аналітика), а Hetzner або DigitalOcean — для розміщення основного масиву серверів та баз даних з метою економії бюджету.

Проте, реалізація такого підходу стримується проблемою інтероперабельності: API та формати даних різних провайдерів є несумісними. Одним із рішень запропонував автор статті [73], що представили модульну архітектуру брокера для оптимального розміщення сервісів між хмарами за критеріями вартості та продуктивності, і показали, що динамічний розподіл інфраструктури між хмарами зменшує витрати порівняно з фіксованим розміщенням, а також демонструють про 25–40% зниження операційних витрат для AI-навантажень через мультихмарні стратегії розміщення.

Це підтверджує актуальність розробки інтегрованої інформаційної технології, яка б виступала шаром абстракції, дозволяючи уніфіковано керувати ресурсами як глобальних гіперскейлерів, так і бюджетних європейських провайдерів у єдиному інтерфейсі.

1.2 **DevOps: принципи та підходи**

Сучасна інженерія програмного забезпечення характеризується переходом від лінійних моделей управління життєвим циклом (SDLC) до ітеративних та безперервних процесів. Методологія DevOps (Development and Operations) виникла як еволюційна відповідь на проблему «організаційних розривів» між підрозділами розробки та експлуатації, що стримували швидкість доставки цінності кінцевому користувачу. У наукових роботах зазначається, що імплементація DevOps-технологій у проєктну діяльність дозволяє не лише скоротити час виходу продукту на ринок , але й змінити саму парадигму забезпечення якості через зміщення контролю на ранні етапи розробки.

Вибір для порівняння традиційних систем обумовлений тим, що, незважаючи на давність каскадної моделі, значна кількість компаній, згідно з дослідженнями [74, 75], і досі використовує традиційні підходи або обирає старіші моделі управління, якщо вони краще відповідають поставленим

завданням, та які порівнюють Waterfall, Scrum і Kanban за критеріями гнучкості, документації, управління ризиками та змінами, а також подають конкретні дані: успішність Agile-проєктів ~40% проти 15% у Waterfall, а рівень провалів 10% проти 30% та ропонують дерево рішень для вибору методології.

Для обґрунтування доцільності застосування DevOps у системах управління інфраструктурою доцільно провести порівняння з традиційними моделями — каскадною (Waterfall) та гнучкою (Agile). Автори [76] аналізують тренд переходу від традиційних моделей (Waterfall, Iterative, RAD) через Scrum до DevOps-based Software Engineering (DOSE), підкріплюючи це даними Google Trends та Stack Overflow. Якщо Waterfall базується на послідовному виконанні етапів із жорсткою фіксацією вимог, а Agile фокусується на адаптивності розробки програмного коду, то DevOps розширює принципи гнучкості на операційне середовище.

Математично ефективність моделі E можна оцінити як функцію від частоти релізів f_{rel} та часу циклу зворотного зв'язку $T_{feedback}$:

$$E \approx \frac{k \cdot f_{rel}}{T_{feedback} + t_{deploy}} \quad (1.4)$$

Формула 1.4 - ефективності моделі DevOps

де t_{deploy} — час, необхідний для розгортання змін, а k — коефіцієнт якості. У моделі DevOps мінімізація $T_{feedback}$ та t_{deploy} досягається шляхом автоматизації, що забезпечує значно вище значення E порівняно з аналогами.

Результати порівняльного аналізу методологій наведено в таблиці (1.2).

Таблиця 1.2

Компаративний аналіз методологій управління розробкою та експлуатацією

Критерій порівняння	Waterfall (Каскадна)	Agile (Гнучка)	DevOps
Основа процесу	Послідовність, документування	Ітеративність, комунікація	Безперервність, автоматизація

Продовження таблиці 1.2

Життєвий цикл	Дискретні стадії (Аналіз -> Код -> Тест -> Експлуатація)	Спринти (Розробка -> Тест -> Демо)	Безперервний потік (CI/CD Pipeline)
Управління релізами	Рідкісні, масштабні, високий ризик ("Big Bang")	Регулярні, в кінці спринта	Часті, малі зміни (On-demand)
Відповідальність за якість	QA-відділ (постфактум)	Команда розробки (під час спринта)	Спільна (Dev + QA + Ops)
Інфраструктура	Статична, ручне налаштування	Частково автоматизована	Динамічна, як код
Реакція на збої	Високий MTTR (середній час відновлення)	Середній MTTR	Мінімальний MTTR

Як видно з таблиці (1.2), DevOps забезпечує найвищий рівень адаптивності та надійності, що підтверджується дослідженнями впровадження цієї технології в IT-проєкти.

Архітектурна відмінність полягає у зміні характеру потоку робіт. В Agile-моделі (без DevOps) часто спостерігається розрив контексту після завершення спринта, коли готовий код передається команді експлуатації. У DevOps-моделі цей процес є безшовним [77].

Для візуалізації відмінностей між Agile-орієнтованим процесом (де Ops відокремлено) та інтегрованим DevOps-потоком розроблено відповідну схему взаємодії (рис. 1.7).

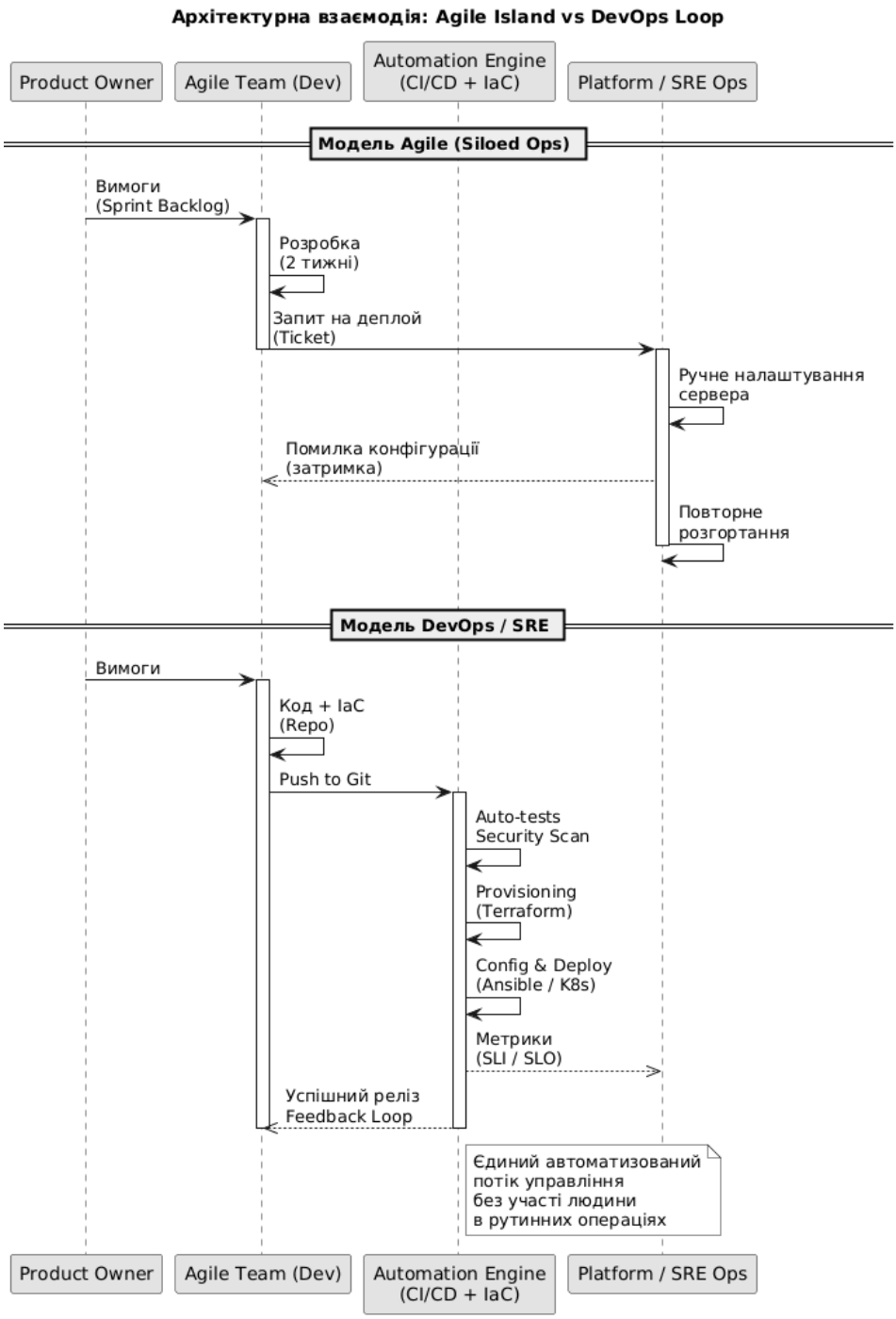

Рис. 1.7. — схема взаємодій

Таким чином, сучасна архітектурна модель управління інфраструктурою не заперечує Agile, а є його логічним продовженням у площину експлуатації. Вона вимагає переходу від ручного адміністрування на запит до створення платформ самообслуговування, де розробники можуть самостійно ініціювати створення ресурсів у межах визначених політик. Це

підтверджує необхідність розробки інформаційної технології, яка б інструментально підтримувала саме таку модель взаємодії, автоматизуючи рутинні операції згідно з практиками SRE.

1.2.1 **Основні принципи DevOps**

У сучасних дослідженнях DevOps розглядається як сукупність організаційних принципів та технічних практик, серед яких ключову роль відіграють:

1. Інфраструктура як код (Infrastructure as Code — IaC). Цей принцип передбачає управління інфраструктурою через декларативні конфігураційні файли, а не через інтерактивні інструменти налаштування. Використання IaC дозволяє застосовувати до інфраструктури практики версіонування (Git), тестування та рецензування коду. Як зазначається у дослідженнях [78,79] порівняльної ефективності інструментів, комбінація Terraform та Ansible забезпечує найвищий рівень автоматизації в гібридних середовищах, дозволяючи досягти ідемпотентності операцій.

Математично, це формула (1.5), що функція застосування конфігурації f до стану системи S задовольняє умові ідемпотентності з формули, яка свідчить про те, що повторне застосування конфігурації не змінює систему після першого застосування — саме це дозволяє Terraform, Ansible та іншим IaC-інструментам безпечно повторно запускати сценарії :

$$\forall S, f(f(S)) = f(S) \qquad (1.5)$$

Формула 1.5 - умови ідемпотентності

2. Безперервна інтеграція та доставка (CI/CD). Автоматизований конвеєр, що забезпечує трансляцію вихідного коду у готовий до експлуатації продукт. Для серверної інфраструктури це означає автоматичне тестування конфігурацій (наприклад, через terraform plan або ansible-lint) та їх застосування без втручання оператора. Це знижує ризик людської помилки, який є домінуючим фактором збоїв у ручних моделях управління.

Ефективність впровадження CI/CD ($E_{\frac{ci}{cd}}$) можна оцінити як формулу (1.6) функцію від частоти розгортання (F_{dep}) та часу відновлення після збоїв ($T_{restore}$):

$$E_{\frac{ci}{cd}} = \alpha \cdot F_{dep} + \beta \cdot \frac{1}{T_{restore}} \quad (1.6)$$

Формула 1.6 - Ефективності впровадження

де α та β — вагові коефіцієнти, що визначають пріоритетність швидкості доставки та стабільності системи відповідно. Дослідження показують, що організації, які використовують CI/CD, значно скорочують час виходу продукту на реліз.

3. Безперервний моніторинг та спостережуваність. Перехід від реактивного моніторингу (реагування на інциденти) до проактивного аналізу метрик. У роботах [80-83] підкреслюється важливість інтеграції систем моніторингу безпосередньо у веб-застосунки управління, що дозволяє отримувати дані про стан серверів у реальному часі та автоматично масштабувати ресурси.

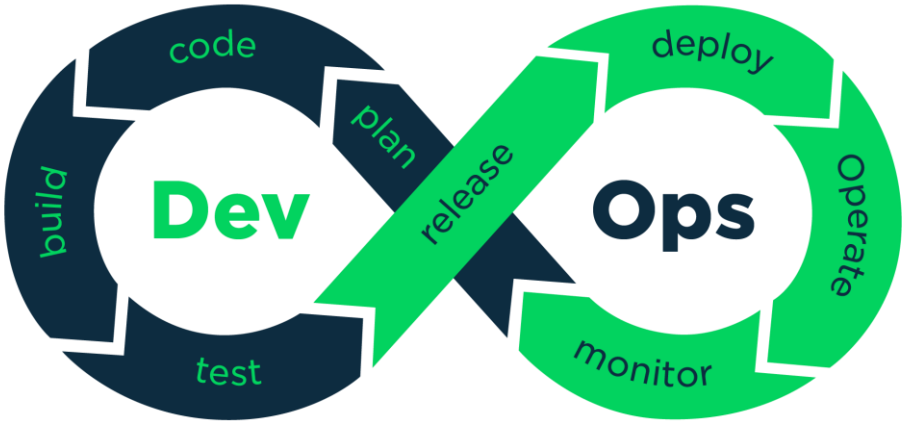

Рис.1.8 – Загальна схема DevOps циклу в розробці

Підсумовуючи, перехід до моделі DevOps є необхідною умовою для побудови сучасних масштабованих систем. Це вимагає не лише зміни інструментарію, а й впровадження нових архітектурних патернів, що

забезпечують безшовну інтеграцію між процесами розробки та управління інфраструктурою.

1.2.2. **Інструментальний базис автоматизації: порівняльний аналіз та архітектурна роль**

Вибір інструментарію для реалізації DevOps-підходу розглядається як задача побудови гетерогенної системи, де кожен компонент відповідає за специфічний шар абстракції: від ініціалізації фізичних ресурсів до оркестрації прикладного програмного забезпечення. Аналіз сучасного стеку технологій [84] дозволяє виділити чотири ключові категорії інструментів, інтеграція яких формує так званий «Toolchain»: ініціалізація, конфігурація, оркестрація та автоматизація (CI/CD).

У класі засобів *Infrastructure as Code (IaC)* домінуюче положення займає Terraform. Його архітектурна відмінність від аналогів (CloudFormation, Pulumi) полягає у використанні декларативної моделі на основі графів залежностей (Directed Acyclic Graph — DAG). Terraform розглядає інфраструктуру як граф G(V, E) де вершини V — це ресурси (сервери, мережі), а ребра E — залежності між ними.

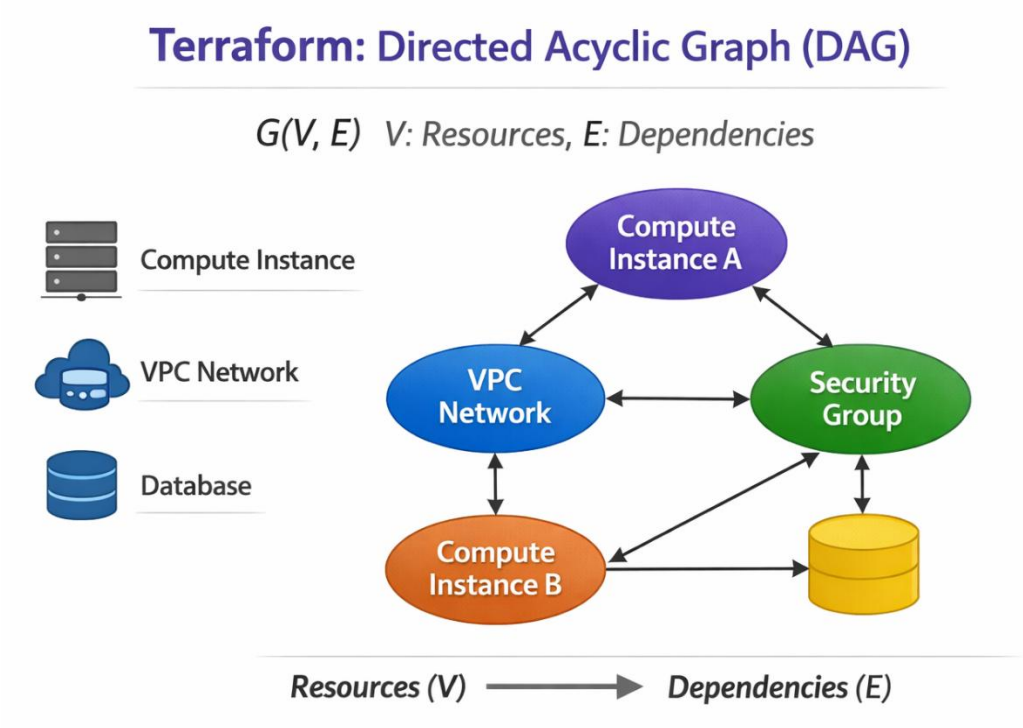

Рис. 1.9. – граф залежностей IAC

Задача Terraform полягає у мінімізації ентропії системи шляхом приведення поточного стану $S_{current}$ до цільового S_{target}, описаного в формулі:

$$\lim_{t\to\infty} \Delta\left(S_{target}, S_{current}(t)\right) = 0 \quad (1.7)$$

Формула 1.7 - Умови асимптотичної збіжності до цільового стану

Наукова цінність Terraform полягає у механізмі управління станом, який дозволяє зберігати метадані про ресурси, створені через API різних провайдерів (AWS, Azure, Hetzner), у єдиному файлі. Це вирішує проблему синхронізації у мультихмарних середовищах, хоча і створює ризики блокування при командній розробці яке зазначено в роботі [85]. Автор [86] пропонує концепцію «Policy-Driven Infrastructure Lifecycle Control Plane» для Terraform — фреймворк, який перетворює Terraform з інструменту деплойменту на повноцінний движок-життевого циклу з безперервним контролем дрейфу конфігурацій.

Якщо Terraform відповідає за створення «скелета» інфраструктури, то Ansible вирішує задачу наповнення її функціоналом. На відміну від агентних систем (Chef, Puppet), Ansible використовує архітектуру «Push», що базується на стандартному протоколі SSH. Це робить його оптимальним для гібридних середовищ, де встановлення агентів на застаріле обладнання може бути неможливим.

Ефективність виконання сценаріїв Ansible для множини серверів N залежить від ступеня паралелізму P та середнього часу виконання задачі τ:

$$T_{total} = \lceil \frac{N}{P} \rceil \cdot \tau + \delta_{net}(P) \quad (1.8)$$

Формула 1.8 - оцінки часу виконання

де δ_{net} — мережева латентність. У роботах, присвячених оптимізації DevOps-процесів, зазначається, що Ansible є де-факто стандартом для налаштування операційних систем. Незважаючи на декларативний підхід до опису стану, виконання playbook має послідовно-імперативний характер, що за відсутності строгої ідемпотентності задач може призводити до розбіжностей при повторних запусках.

Масштабне емпіричне дослідження [87], що включало аналіз 59 157 постів зі Stack Overflow, Reddit та Ansible Forum, а також 16 напівструктурованих інтерв'ю, дозволило виявити ключові проблемні зони використання Ansible. Автори сформулювали чотири основні рекомендації: рефакторинг для підвищення продуктивності, реструктуризація мовних конструкцій вищого рівня, покращення інструментів налагодження та звітності про помилки, а також удосконалення документації.

Перехід до мікросервісної архітектури зумовив необхідність впровадження систем оркестрації, серед яких абсолютним лідером є Kubernetes (K8s). З точки зору теорії управління, K8s є системою автоматичного регулювання, що підтримує задану кількість реплік додатку. Він вирішує класичну задачу пакування (Bin Packing Problem) — розміщення контейнерів k з вимогами до ресурсів r_i на вузлах x_{ij} з ємністю C_j так, щоб мінімізувати кількість задіяних вузлів:

$$\sum_{i=1}^{k} r_i \cdot x_{ij} \leq C_j, \forall j \tag{1.9}$$

Формула 1.9 - розрахунку Лінійне обмеження допустимості розподілу ресурсів у системі оркестрації

Дослідження [88, 89] показують, що Kubernetes забезпечує найвищий рівень надійності для stateless-додатків, проте його впровадження для stateful-

сервісів (баз даних) у гібридних хмарах пов'язане зі значними складнощами організації персистентних сховищ (Persistent Volumes).

Роль диспетчера, що об'єднує вищезгадані інструменти в єдиний конвеєр, виконує Jenkins. Попри появу новіших рішень (GitLab CI, GitHub Actions), Jenkins залишається еталоном гнучкості завдяки своїй плагінній архітектурі. Він дозволяє реалізувати складні алгоритми розгортання, які можуть бути описані як кінцеві автомати (Finite State Machines). Проте, архітектурним недоліком Jenkins є його монолітна природа та високе споживання ресурсів Java Virtual Machine (JVM), що спонукає до пошуку більш легковагових альтернатив у хмарно-нативних середовищах. Аналіз статей [90 - 93] доводить що Jenkins — найгнучкіший, але вимагає ручної конфігурації. Систематизація характеристик розглянутих інструментів дозволяє виявити їхні архітектурні відмінності (Таблиця 1.3).

Таблиця 1.3

Порівняльна характеристика інструментів за рівнем абстракції та методом управління

Інструмент	Клас задачі	Модель управління	Рівень абстракції	Ключова перевага для дослідження
Terraform	Provisioning (IaaS)	Декларативна (Stateful)	API Хмари	Мультихмарна підтримка (AWS, Hetzner, etc.)
Ansible	Config Mgmt	Процедурна/Декларативна	ОС (Linux/Win)	Безагентна архітектура (SSH), простота
Kubernetes	Orchestration	Декларативна (Loop)	Контейнер	Самовідновлення та масштабування
Jenkins	CI/CD	Імперативна (Pipeline)	Процес (Workflow)	Необмежена розширюваність

Для ілюстрації взаємозв'язків між інструментами у типовому гібридному середовищі розроблено компонентну діаграму (рис. 1.10).

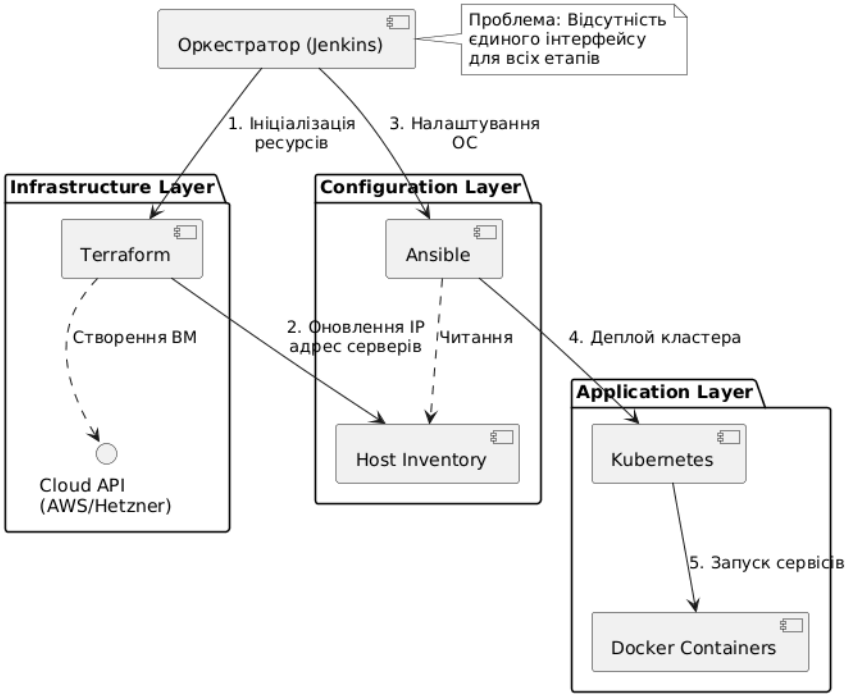

Рис. 1.10 — Схема взаємодії інструментального стеку при розгортанні інфраструктури

Проведений аналіз демонструє, що існуючий інструментарій (Terraform, Ansible, Kubernetes, Jenkins) покриває всі технічні аспекти автоматизації. Однак, наукова проблема полягає у відсутності наскрізної інтеграції: ці інструменти функціонують як розрізнені модулі. Передача даних між етапом створення сервера (Terraform) та його налаштуванням (Ansible) часто вимагає розробки додаткових проміжних скриптів або ручного втручання для оновлення файлів інвентаризації. Це підтверджує необхідність розробки інтегруючої інформаційної технології, яка б автоматизувала цей потік даних та приховала складність низькорівневої взаємодії за єдиним веб-інтерфейсом.

1.3 **Проблеми і виклики автоматизації в гібридних середовищах**

Впровадження автоматизованих систем управління в сучасних організаціях дедалі частіше відбувається в умовах гібридної інфраструктури, що об'єднує локальні обчислювальні потужності (On-Premises) та ресурси публічних хмарних провайдерів. Такий архітектурний підхід дозволяє поєднувати контроль над критично важливими даними, властивий локальним серверам, із гнучкістю та масштабованістю хмарних сервісів. Проте,

гетерогенність гібридних середовищ породжує низку специфічних проблем, які не можуть бути вирішені шляхом простого перенесення практик, розроблених виключно для хмарних або виключно для локальних систем. Серед інших варіантів вирішення задач інтеграції та міграції до гібридних хмарних середовищ у наукових роботах пропонуються як архітектурні, так і модельно-орієнтовані підходи.

Зокрема, автори [94] пропонують рішення Hybrid Cloud Connector, яке централізує нефункціональні аспекти (безпека, управління, відповідність вимогам) через керовану політиками точку контролю, зменшуючи складність інтеграції.

У роботі [95] автор застосовує підхід модельно-орієнтованої інженерії, що дозволяє автоматизувати перетворення застарілих систем через використання платформонезалежних та платформозалежних моделей.

У дослідженні [96] представлено практичний п'ятифазний підхід міграції з використанням патерну «задушливого фігового дерева» та підходів інфраструктури як коду, що дозволило суттєво скоротити час розгортання та обсяг ручних операцій.

Оглядова стаття [97] узагальнює ключові підходи до інтеграції, включаючи стратегії вилучення, перетворення та завантаження даних, використання проміжного програмного забезпечення та архітектур, орієнтованих на прикладні програмні інтерфейси, у гібридних середовищах.

У цілому, ці дослідження демонструють тенденцію до переходу від монолітних інтеграційних рішень до більш гнучких, автоматизованих та інкрементальних підходів, що забезпечують зниження складності, підвищення керованості та поступову модернізацію систем.

1.3.1 **Труднощі інтеграції локальних і хмарних серверів**

Фундаментальною проблемою інтеграції є технологічний розрив між застарілими локальними системами та сучасними хмарними стандартами. Локальні сервери часто функціонують на базі пропрієтарного програмного забезпечення або застарілих версій операційних систем, що унеможливлює

використання стандартних агентів автоматизації та API-інтерфейсів, характерних для хмарних платформ.

Формула (1.10) використовується для узагальненої оцінки технічної складності інтеграції інформаційних систем шляхом врахування кількості різнорідних інтерфейсів, ступеня сумісності протоколів та впливу мережевих затримок. Технічна складність інтеграції (C_{int}) може бути представлена як функція від кількості різнорідних інтерфейсів (N_{if}) та ступеня сумісності протоколів (k_{compat}):

$$C_{int} = \sum_{i=1}^{n} \frac{1}{k_{compat(i)}} \cdot \log\left(N_{if(i)}\right) + L_{net} \tag{1.10}$$

де L_{net} — фактор мережевої затримки, що виникає при організації захищених каналів зв'язку (VPN, Direct Connect) між локальним дата-центром та хмарою. Синхронізація даних у реальному часі ускладнюється через латентність мережі та обмежену пропускну здатність каналів, що призводить до виникнення колізій та неузгодженості станів баз даних у розподілених системах .

Окремим викликом є фрагментація механізмів управління. Хмарні ресурси керуються через уніфіковані API провайдерів (наприклад, AWS API), тоді як локальні сервери вимагають низькорівневого доступу (IPMI, SSH). Це створює бар'єр, який ускладнює створення єдиного конвеєра CI/CD. Для візуалізації цієї проблеми наведено діаграму компонентів (рис. 1.4), що демонструє розрив у методах доступу. У цьому контексті робота [98] досліджує оптимізацію CI/CD-пайплайнів у мультихмарних середовищах і прямо ідентифікує три ключові перешкоди, що ускладнюють інтеграцію: фрагментацію інструментів, неконсистентність політик безпеки та складність інфраструктури — саме ті фактори, які часто описують як «архітектурний бар'єр».

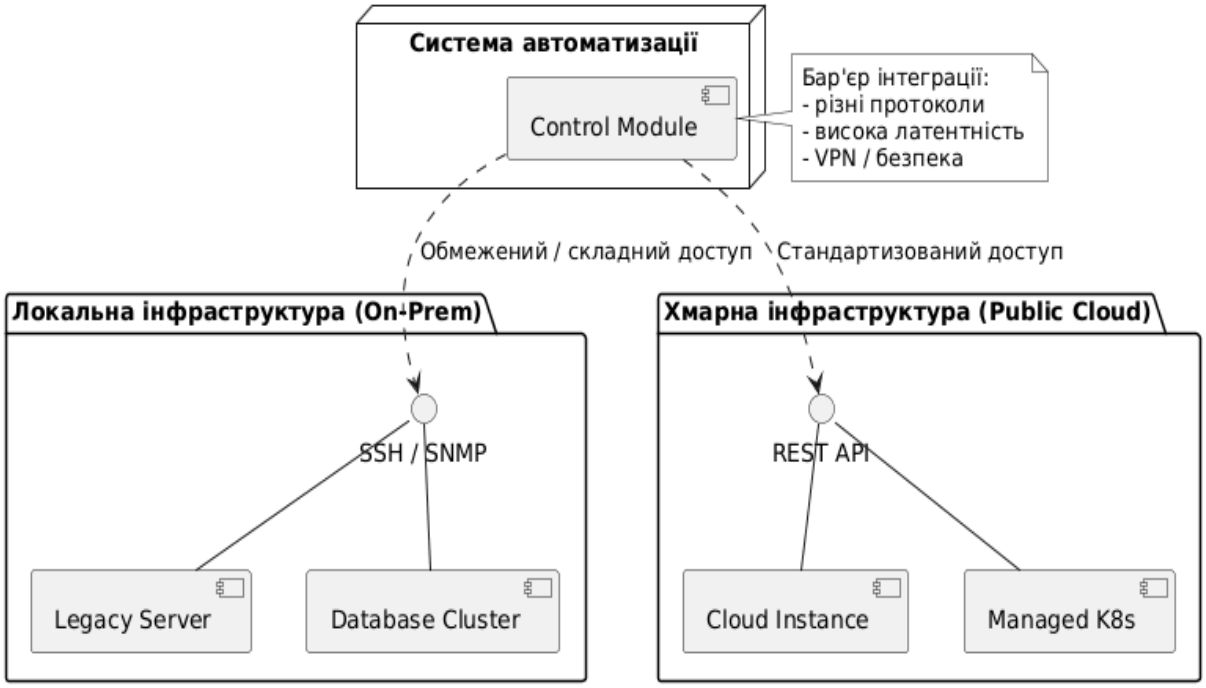

Рис. 1.11. — Діаграма компонентів гібридної інфраструктури автоматизації

1.3.2 **Проблеми масштабованості, безпеки, моніторингу та управління великими інфраструктурами**

Масштабування у гібридних середовищах стикається з проблемою асиметрії ресурсів. Хмарні платформи забезпечують еластичне масштабування, дозволяючи автоматично додавати ресурси при зростанні навантаження. Натомість локальні сервери обмежені фізичними характеристиками обладнання (CPU, RAM, Storage), що робить їх масштабування дискретним та часозатратним процесом.

Ефективність масштабування (E_{scale}) визначається як частка запитуваних ресурсів, які можуть бути забезпечені локальними та хмарними ресурсами. Якщо локальні ресурси недостатні, хмара доповнює їх тільки до необхідного рівня. Формула (1.11) показує, скільки з усіх необхідних ресурсів може бути надано системою (локальною та хмарною) в порівнянні з тим, що потрібно:

$$E_{scale} = \left\{ \frac{R_{local}^{max} + min(R_{cloud}^{avail}, R_{req} - R_{local}^{max})}{R_{req}} \right\} \quad (1.11)$$

Формула 1.11 - Ефективність масштабування гібридної системи

де R_{req} — запитувані ресурси, R_{local}^{max} — межа локальних потужностей. Неточне прогнозування пікових навантажень у такій моделі призводить до економічних втрат або деградації сервісу .

У сфері безпеки гібридна модель значно розширює поверхню атаки. Критичною проблемою стає відсутність єдиного простору ідентифікації та управління доступом (IAM). Політики безпеки, налаштовані у хмарі (наприклад, групи безпеки в AWS), не розповсюджуються автоматично на локальні брандмауери, що створює ризики несанкціонованого доступу. Крім того, забезпечення відповідності регуляторним вимогам, таким як GDPR та PCI DSS, ускладнюється через необхідність контролю руху даних між різними юрисдикціями.

У цьому контексті робота [99] пропонує стратегію мультихмарного розміщення даних з урахуванням регуляторних вимог, яка інтегрує дотримання правил безпосередньо в дизайн хмарної архітектури з огляду на локалізаційні мандати різних юрисдикцій. Пропонується фреймворк «Розділеної безпеки хмари», в роботі [100], що відокремлює застосування політик безпеки від хмарної інфраструктури, дозволяючи автоматично впроваджувати регуляторні вимоги незалежно від провайдера.

Моніторинг великих розподілених інфраструктур ускладнюється через гетерогенність форматів даних. Локальні системи часто використовують застарілі протоколи (SNMP, Syslog), тоді як хмарні сервіси генерують метрики у форматах JSON або Prometheus exposition format. Агрегація цих даних у єдину аналітичну панель вимагає розробки складних адаптерів та нормалізації потоків даних, що підвищує навантаження на систему управління .

Таким чином, ефективне управління гібридними середовищами неможливе без створення надбудови (abstraction layer), яка б уніфікувала взаємодію з різнорідними ресурсами, забезпечувала централізований контроль доступу та нівелювала технологічні розбіжності між локальними та хмарними компонентами .

1.4 Висновки до першого розділу

У першому розділі дисертаційної роботи проведено системний аналіз сучасного стану та проблематики автоматизації управління серверною інфраструктурою в умовах переходу до гібридних архітектур. За результатами дослідження зроблено наступні висновки:

Встановлено, що домінуючою парадигмою управління інфраструктурою є методологія DevOps, реалізована через підходи «Інфраструктура як код» (IaC) та безперервну інтеграцію (CI/CD). Ринок засобів автоматизації представлений потужними спеціалізованими інструментами: Terraform для ініціалізації ресурсів, Ansible для управління конфігураціями та Kubernetes для оркестрації контейнерів. Хмарні провайдери (AWS, Azure, Hetzner) надають розвинені API для керування ресурсами в межах власних екосистем.

Попри широкі функціональні можливості, існуючі інструменти мають суттєві архітектурні обмеження при використанні в гетерогенних гібридних середовищах:

- Фрагментарність управління - Інструменти функціонують ізольовано (Terraform не керує налаштуваннями ОС, Ansible не керує життєвим циклом хмарних інстансів), що вимагає ручної синхронізації процесів або написання складних скриптів-обгорток.
- Проблема інтеграції - Відсутні уніфіковані механізми, які б дозволяли прозоро керувати локальними серверами та хмарними ресурсами через єдиний інтерфейс. Взаємодія з API різних провайдерів вимагає розробки окремих адаптерів, що ускладнює масштабування.

- Високий поріг входження - Ефективне використання існуючого стеку вимагає глибоких знань специфічних декларативних мов (HCL, YAML) та навичок роботи з командним рядком, що обмежує доступність управління для операційного персоналу.

На сьогодні відсутня комплексна інформаційна технологія, яка б інтегрувала процеси ініціалізації та конфігурації у єдиний керований потік з централізованим веб-інтерфейсом. Існуючі рішення класу CMP (Cloud Management Platforms) є або надто дорогими для середнього сегмента, або орієнтовані виключно на великих гіперскейлерів, ігноруючи потреби гібридних інфраструктур з використанням бюджетних провайдерів (наприклад, Hetzner) та локальних серверів.

Для візуалізації виявленої проблеми розриву в автоматизації побудовано концептуальну схему (рис. 1.12).

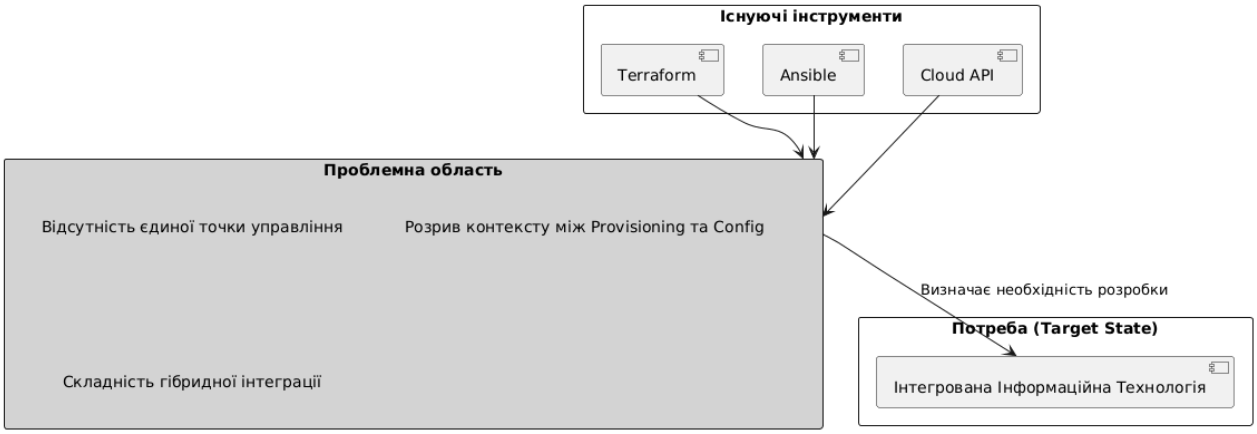

Рис. 1.12. – Концептуальна схема проблеми розриву в автоматизації

Вирішення зазначених проблем вимагає створення нової інформаційної технології, яка забезпечить:

- Абстрагування від складності низькорівневих API хмарних провайдерів.
- Автоматизовану оркестрацію взаємодії між Terraform та Ansible.

- Централізацію моніторингу та управління доступом (RBAC) через веб-інтерфейс.

Виявлені прогалини в існуючих підходах логічно зумовлюють мету дисертаційної роботи — підвищення ефективності управління гібридною інфраструктурою. Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Обґрунтувати архітектуру інформаційної технології, що інтегрує DevOps-інструменти.
2. Розробити алгоритми автоматизованого розгортання серверів у гетерогенних середовищах.
3. Створити програмну реалізацію системи управління на базі веб-технологій.
4. Експериментально перевірити ефективність запропонованого рішення.

Таким чином, аналіз існуючих рішень показує, що попри наявність потужних DevOps-інструментів, відсутні інтегровані інформаційні технології, які забезпечують наскрізну автоматизацію управління гібридною інфраструктурою з єдиного центру. Розробка такої технології, що поєднує можливості IaC, універсальних API-адаптерів та веб-інтерфейсу, є актуальним науково-прикладним завданням, вирішенню якого присвячено наступний розділ.

РОЗДІЛ 2. РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ АВТОМАТИЗАЦІЇ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

2.1 Архітектура рішення

Вибір архітектурного підходу для реалізації інформаційної технології автоматизованого управління серверною інфраструктурою ґрунтується на необхідності забезпечення універсальності при роботі з гетерогенними ресурсами. В основу програмного комплексу покладено високорівневий вебфреймворк Django (Python обрано через нативну інтеграцію з Ansible, іншими інструментами та бібліотеками хмарних API, що пришвидшує розробку), який функціонує як централізована панель керування (панель управління).

Існує декілька підходів до реалізації подібних панелей управління. Деякі рішення орієнтовані на програмний контроль виконання завдань Ansible та збір інформації про прогрес [101], інші застосовують штучний інтелект та машинне навчання для автоматизації провізіонінгу в мультихмарних середовищах [102]. Також досліджуються [103, 104] гібридні та Kubernetes-нативні панелі управління для уніфікованого контролю ресурсів, незалежно від провайдера. Кожен із цих підходів має свої переваги та обмеження, зокрема щодо інтеграції існуючих серверів, управління гетерогенними ресурсами та забезпечення централізованого контролю доступу.

В рамках цієї дисертації обрана архітектура системи передбачає модель агрегації без збереження стану, де веб-застосунок виступає єдиною точкою входу для адміністрування та забезпечує уніфікований інтерфейс для ресурсів із різних джерел. Ключовою архітектурною особливістю є реалізація дуального вектора наповнення інфраструктури, що дозволяє системі гнучко адаптуватися до потреб адміністратора. Система підтримує два методи реєстрації обчислювальних вузлів:

1. Автоматизована синхронізація через API - для роботи з хмарними провайдерами (на прикладі Hetzner Cloud) реалізовано механізм динамічного імпорту, і показно на рисунку (2.1) .

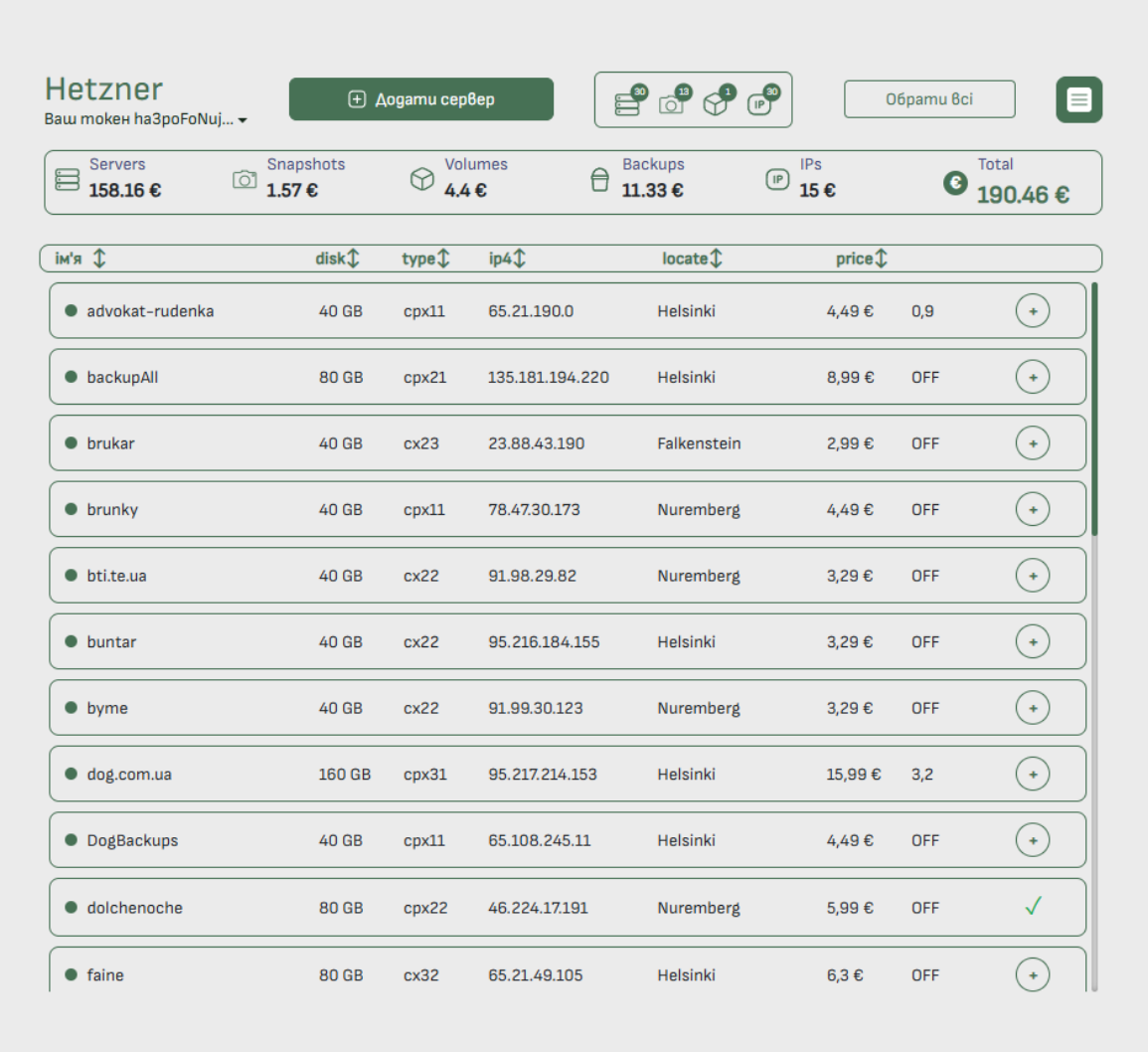

Рис. 2.1. – Перегляд створеної інфраструктури через веб-застосунок

Архітектура передбачає зберігання API-токенів провайдера, прив'язаних до профілю користувача. При зверненні до панелі керування система ініціює запит до зовнішнього API (client.servers.get_all()), отримуючи повний перелік активних серверів, їх технічні характеристики (vCPU, RAM, Disk), мережеві параметри та вартість обслуговування. Цей підхід дозволяє миттєво інтегрувати в систему сотні існуючих серверів без необхідності ручного введення даних. Система автоматично розпізнає конфігурацію, включно з резервними копіями, знімками системи (Snapshots) та підключеними томами

даних (Volumes), відображаючи актуальний стан інфраструктури в реальному часі.

2. Ручна реєстрація - для інтеграції локальних фізичних серверів (On-Premise), віртуальних машин інших провайдерів або специфічних вузлів, які не підлягають автоматичному виявленню, реалізовано механізм прямого додавання показана на рисунку (2.2). Адміністратор через веб-інтерфейс визначає адресу хоста (IP/Hostname) та прив'язує необхідні облікові дані (SSH-ключі або паролі). У цьому випадку система створює абстракцію керованого об'єкта, яка дозволяє застосовувати до «стороннього» сервера весь набір інструментів автоматизації (моніторинг, Docker-менеджмент, термінал) нарівні з хмарними інстансами.

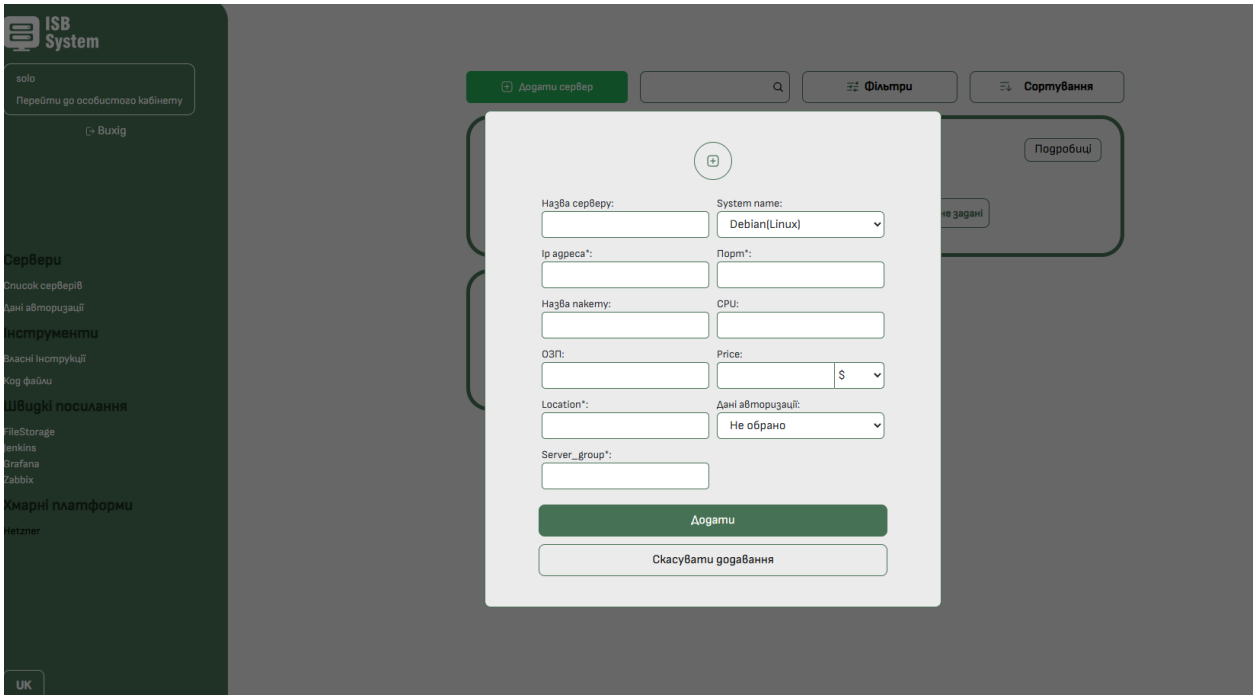

Рис. 2.2. - Інтерфейс модального вікна додавання нового сервера до системи управління

Незалежно від методу додавання сервера (автоматичний чи ручний), критично важливим аспектом функціонування системи є безпечне зберігання облікових даних (SSH-ключів, паролів, API-токенів). Для мінімізації ризиків

компрометації у розробленій технології відмовлено від зберігання секретів у відкритому вигляді на користь гібридної криптографічної моделі.

Реалізація підсистеми управління ідентифікацією побудована на базі протоколу Envelope Encryption (Конвертне шифрування) з використанням зовнішнього сервісу управління ключами (KMS) HashiCorp Vault. Структурна схема захисту даних наведена на рисунку (2.3).

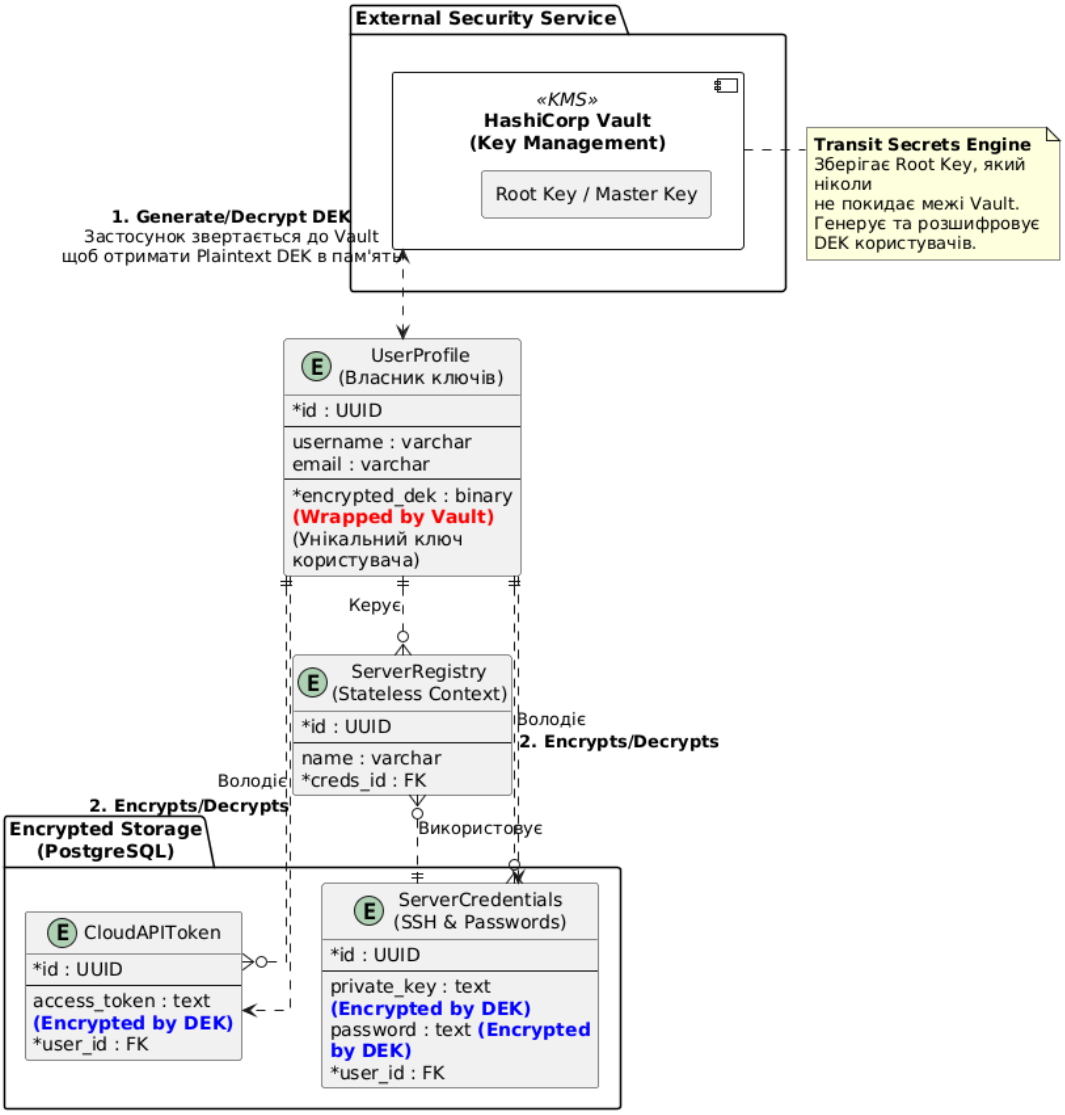

Рис. 2.3. - Схема захисту даних із використанням HashiCorp Vault та конвертного шифрування

Як видно зі схеми, архітектура безпеки розділена на три рівні ізоляції:

1. Рівень Майстер-ключа: Головний ключ шифрування (Root Key) зберігається в ізольованому середовищі HashiCorp Vault (Transit Secrets Engine) і ніколи не покидає його меж.

2. Рівень користувача: Для кожного користувача генерується унікальний ключ шифрування даних (Data Encryption Key — DEK). Цей ключ зберігається в профілі користувача в базі даних, але лише у зашифрованому вигляді. Розшифрування DEK відбувається виключно в оперативній пам'яті на момент виконання операції шляхом звернення до Vault.

3. Рівень даних: Кінцеві секрети (токени хмарних провайдерів CloudAPIToken та приватні ключі серверів ServerCredentials) шифруються за допомогою DEK алгоритмом AES-256 і зберігаються в базі даних PostgreSQL.

Такий підхід гарантує, що навіть у випадку повного витоку дампа бази даних зловмисники не зможуть отримати доступ до інфраструктури, оскільки ключі для розшифрування (DEK) самі зашифровані ключем, який фізично відсутній у базі даних.

Разом з тим, важливо зазначити, що база даних виступає лише сховищем контексту безпеки та ідентифікаторів ресурсів. При використанні API-синхронізації система не дублює динамічні параметри серверів (статус running/stopped, поточний IP, завантаження CPU), а запитує їх «на льоту» Це гарантує, що оператор завжди бачить валідний стан інфраструктури, навіть якщо зміни (наприклад, перезавантаження сервера) відбулися поза межами веб-застосунку.

Для демонстрації практичної реалізації розробленої системи на рисунках (2.4–2.5) наведено інтерфейс користувача, що відображає перелік керованих серверів та інтерактивний термінал для віддаленого доступу.

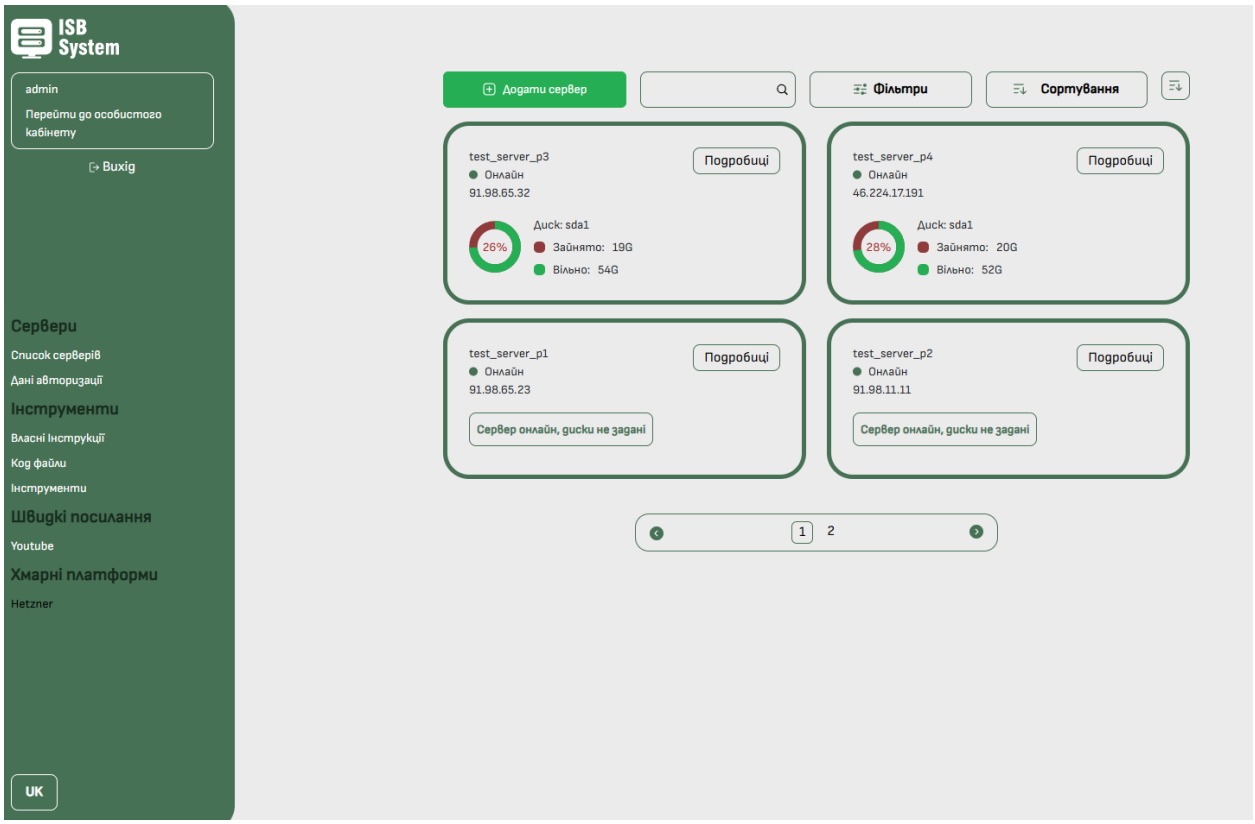

Рис. 2.4. - Головна сторінка серверів, у веб застосунку

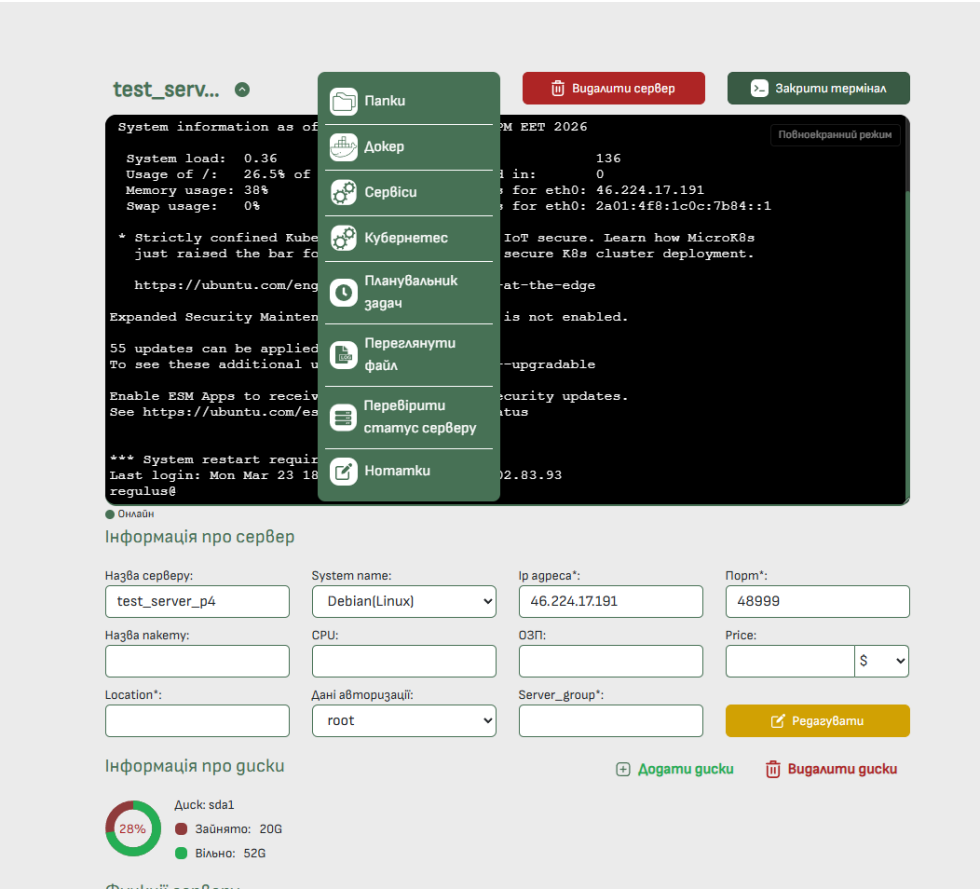

Рис. 2.5. – Сторінка серверу, з терміналом та додатковим функціоналом

На рисунку (2.4) представлено панель управління серверами, яка забезпечує централізований перегляд інфраструктури, включаючи статус вузлів, їх технічні характеристики та параметри підключення. Інтерфейс дозволяє оперативно здійснювати навігацію між серверами та ініціювати керуючі дії.

На рисунку (2.5) продемонстровано вбудований веб-термінал, що реалізує захищене підключення до серверів через SSH-тунелювання. Даний компонент забезпечує можливість виконання адміністративних команд безпосередньо з веб-інтерфейсу без використання сторонніх клієнтів.

Представлені інтерфейси підтверджують зручність експлуатації системи та ефективність інтеграції механізмів централізованого управління серверною інфраструктурою.

Веб-застосунок реалізує уніфікований інтерфейс керування, який приховує від користувача різницю в походженні сервера. Незалежно від того, чи був сервер імпортований автоматично через API, чи доданий вручну, система надає ідентичний набір функціональних модулів:

- Модуль управління контейнеризацією: Взаємодія з Docker Engine через сокет для керування контейнерами.
- Модуль системних служб: Моніторинг та управління демонами systemd.
- Веб-термінал: Емуляція SSH-сесії через WebSocket для виконання консольних команд у браузері.
- Планувальник завдань: Інтерфейс для керування Cron-завданнями.

Для хмарних серверів, отриманих через API, додатково активуються специфічні функції керування життєвим циклом, доступні через API провайдера: створення знімків, скидання пароля, зміна типу інстансу та управління плаваючими IP-адресами.

Взаємодія з серверами, доданими будь-яким із методів, здійснюється асинхронно через сервер Daphne. Це дозволяє виконувати "важкі" операції

(наприклад, отримання списку з 50 серверів через API або виконання скрипту оновлення на віддаленому хості) без блокування інтерфейсу користувача, забезпечуючи високу чуйність системи.

2.1.1 **Моделі взаємодії між серверною інфраструктурою і хмарними провайдерами**

Для забезпечення уніфікованого управління гетерогенними ресурсами в розробленій інформаційній технології реалізовано багаторівневу модель абстракції. Ця модель функціонує як проміжне програмне забезпечення, що транслює високорівневі вимоги користувача (наприклад, «створити сервер») у специфічні команди для конкретних провайдерів та операційних систем.

Взаємодія побудована за ієрархічним принципом і складається з трьох логічних рівнів:

1. Декларативна модель. Рівень інфраструктурної абстракції (Provider Agnostic Layer) На цьому рівні вирішується проблема несумісності API різних хмарних провайдерів. Система реалізує патерн проєктування «Адаптер», який уніфікує інтерфейси керування. Взаємодія базується на декларативному підході:

- Система оперує універсальною сутністю Server, яка не залежить від конкретної платформи.
- Драйвери взаємодії динамічно транслюють параметри цієї сутності в API-виклики для AWS (EC2), Hetzner (Cloud) або Azure (VM).
- Це дозволяє ізолювати ядро системи від змін у специфікаціях зовнішніх API, забезпечуючи високу портативність рішення.

2. Рівень операційної оркестрації. Цей рівень відповідає за взаємодію з операційним середовищем сервера незалежно від того, де він розміщений — у хмарі чи локально. В основу покладено безагентну модель взаємодії.

- Замість встановлення пропрієтарних агентів, система використовує стандартний протокол SSH як транспортний рівень.
- На цьому рівні реалізується перехід від декларативного опису інфраструктури до імперативного виконання команд конфігурації.

- Модель забезпечує ідемпотентність операцій: повторне виконання сценарію налаштування гарантує приведення системи до цільового стану без дублювання змін.

3. Модель синхронізації станів. Третій рівень моделює взаємодію обчислювальних ресурсів із зовнішніми мережевими сервісами. Ключовим елементом є автоматизація керування системою доменних імен (DNS).

- Система синхронізує життєвий цикл сервера із записами DNS, реалізуючи принцип «динамічної адресації».
- Взаємодія відбувається через API провайдерів DNS (наприклад, CloudFlare), що дозволяє автоматично маршрутизувати трафік на новостворені сервери одразу після їх ініціалізації.

Візуалізація запропонованої архітектури наведена на рисунку 2.6.

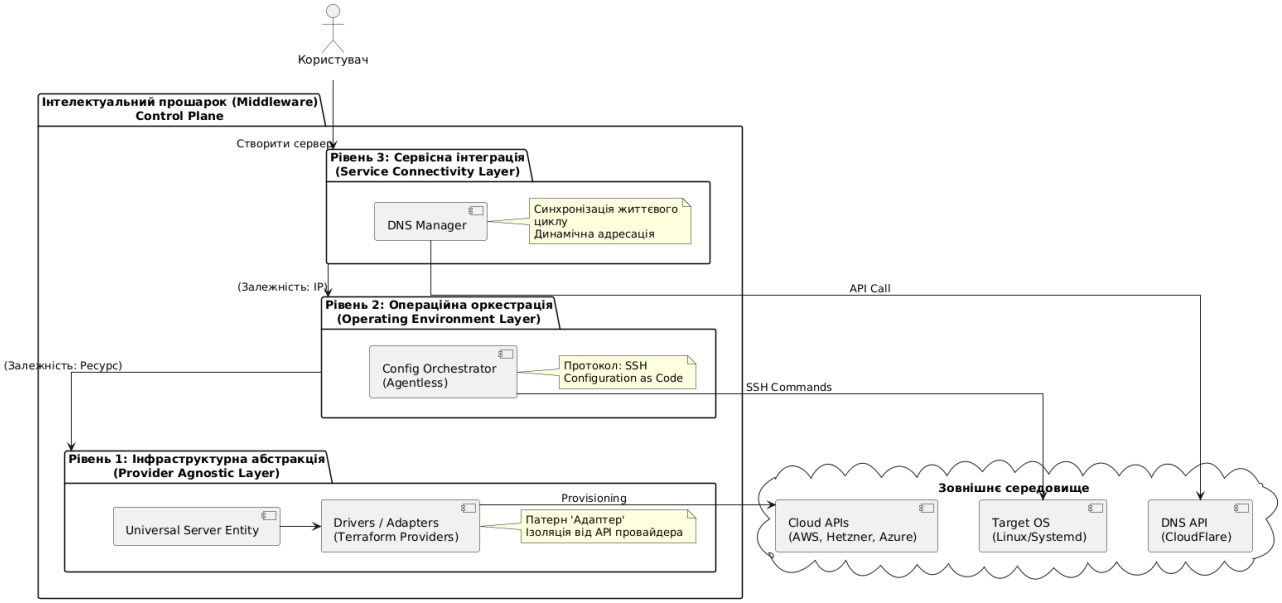

Рис. 2.6 – Схема запропонованої архітектури

Як видно зі схеми, веб-застосунок виступає в ролі головного шару, який ізолює користувача від технічної складності зовнішнього середовища. Потік управління на діаграмі демонструє трансформацію абстрактної вимоги («Створити сервер») у каскад конкретних операцій зверху вниз:

1. На верхньому рівні (L3) система резервує мережеву ідентичність (DNS), визначаючи публічну «адресу» майбутнього сервісу.

2. На середньому рівні (L2) формується конфігурація операційного середовища, що визначає функціональну «поведінку» сервера.

3. На нижньому рівні (L1) адаптери інфраструктури взаємодіють з API провайдерів (AWS, Hetzner), створюючи необхідний фізичний або віртуальний фундамент.

Така архітектурна декомпозиція дозволяє замінювати компоненти одного рівня (наприклад, змінити хмарного провайдера на L1), не порушуючи логіку роботи інших рівнів (налаштування ПЗ на L2 або DNS на L3).

Узагальнюючи вищевикладене, слід зазначити, що запропонована трирівнева модель виступає необхідним технологічним базисом для комплексної автоматизації. Наявність абстрагованих каналів взаємодії з драйверами IaaS, операційними системами та DNS-сервісами дозволяє перейти від статичного адміністрування окремих компонентів до реалізації наскрізних транзакційних процесів. Саме ця архітектурна спроможність системи стала фундаментом для розробки процедурної логіки автоматизованого розгортання, яка детально розглядається у наступному підрозділі.

2.2 **Алгоритм автоматизації підняття серверів**

Ключовим компонентом розробленої інформаційної технології є алгоритм комплексний конвеєр для автоматизованої серверної інфраструктури на англійській CPASI (Complex Pipeline for Automated Server Infrastructure), який забезпечує автоматизоване розгортання серверних ресурсів. Основна науково-практична задача, яку вирішує даний алгоритм, полягає в забезпеченні атомарності транзакцій створення інфраструктури, що об’єднує процеси ініціалізації віртуальних машин, налаштування програмного забезпечення та конфігурацію мережевого оточення в єдиний нерозривний технологічний цикл.

Для забезпечення цілісності даних та реалізації концепції «Єдиного джерела істини» було спроєктовано реляційну модель даних, нормалізовану до третьої нормальної форми (3NF). Центральною сутністю системи є Server, яка агрегує інформацію про фізичні параметри, мережеві налаштування та облікові дані. Зв'язки між сутностями побудовано таким чином, щоб забезпечити каскадне видалення залежних об'єктів (наприклад, історія операцій видаляється разом із сервером) або збереження критичних даних (облікові дані не дублюються, а посилаються через зовнішні ключі). Структура основних сутностей та зв'язків між ними представлена на ER-діаграмі (Entity-Relationship Diagram) на рисунку (2.7).

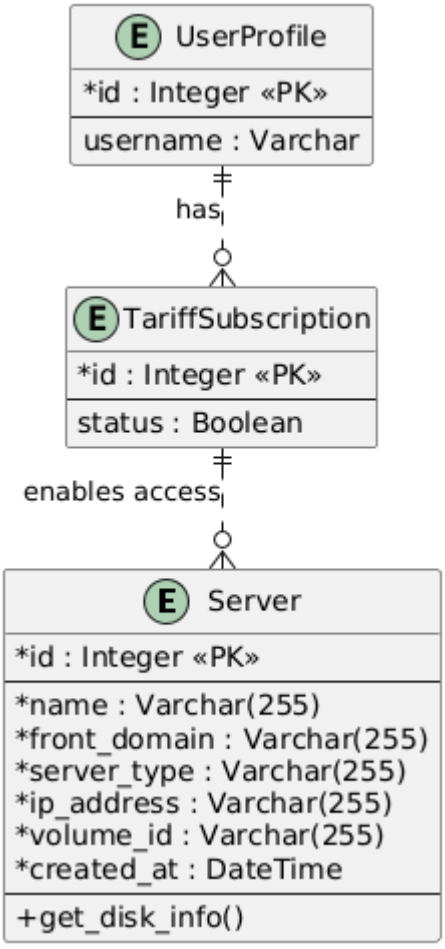

Рисунок 2.7 – ER діаграма реляційної моделі даних

Як видно з наведеної діаграми, архітектура бази даних реалізує ієрархічну модель управління доступом до обчислювальних ресурсів. Кореневою сутністю виступає UserProfile, що ідентифікує користувача в

системі. Взаємодія користувача з інфраструктурою опосередкована сутністю TariffSubscription, яка визначає права доступу. Атрибут status (Boolean) у цій сутності виконує роль логічного перемикача, дозволяючи або блокуючи можливість створення та управління серверами.

Ключова сутність Server містить вичерпний набір метаданих, необхідних для автоматизації життєвого циклу віртуальної машини:

- Технічні ідентифікатори: ip_address та front_domain забезпечують мережеву доступність ресурсу.
- Конфігураційні параметри: server_type визначає обчислювальну потужність (CPU/RAM), а volume_id — прив'язку до сховища даних.
- Часові мітки: created_at фіксує момент ініціалізації ресурсу для задач білінгу та аудиту.

Особливістю спроєктованої моделі є інтеграція процедурної логіки безпосередньо в структуру даних, що відображено наявністю методу get_disk_info(). Цей метод забезпечує динамічне отримання інформації про стан дискової підсистеми сервера в реальному часі, не перевантажуючи базу даних надлишковим зберіганням метрик моніторингу.

2.2.1 **Опис розробленого алгоритму для швидкого розгортання серверів**

Алгоритм базується на принципі динамічної генерації інфраструктури, де параметри цільової системи визначаються на етапі виконання (runtime) на основі запиту користувача, а не жорстко закодовані у статичних файлах. Процес розгортання реалізовано у вигляді послідовності кроків, контроль виконання яких здійснюється центральним модулем управління на базі Django.

Логічна структура алгоритму складається з наступних етапів:

1. Валідація та генерація ідентифікаторів. На вхід алгоритму подається запит із зазначенням кількості та типу необхідних серверів. Система виконує перевірку лімітів ресурсів користувача та генерує унікальні ідентифікатори для нових серверів. Використовується розроблена система

іменування на основі шестизначних числових префіксів (наприклад, 000123-server), що забезпечує унікальність імен у просторі до 999 999 об'єктів та дозволяє автоматизувати створення відповідних доменних імен третього рівня.

2. Формування декларативного опису (JSON-генерація). На основі згенерованих метаданих система створює файл servers.json, який містить опис цільового стану інфраструктури. Цей етап є критичним для підтримки гетерогенності, оскільки дозволяє динамічно визначати тип сервера (наприклад, cx22 для базових задач або потужніші конфігурації для обчислень), образ операційної системи та локацію розміщення для кожного окремого екземпляра у масиві даних.

3. Паралельна ініціалізація ресурсів. Згенерований JSON-файл передається інструменту Terraform, який виконує створення віртуальних машин. Використання мета-аргументу for_each дозволяє Terraform створювати декілька серверів паралельно, незалежно від їх кількості, що забезпечує лінійну масштабованість процесу. Для прискорення розгортання використовується попередньо підготовлений знімок системи, що містить базовий набір утиліт (Docker, Python, налаштований SSH), що скорочує час готовності сервера з кількох годин до 8–10 хвилин.

4. Синхронізація мережевих параметрів. Після отримання публічних IP-адрес від хмарного провайдера алгоритм виконує оновлення локальної бази даних та ініціює запити до API DNS-провайдера (CloudFlare) для створення A-записів. Це забезпечує доступність серверів за доменними іменами (наприклад, project000123.domain.com) одразу після їх запуску.

5. Фінальна конфігурація та безпека. На завершальному етапі генерується інвентарний файл для Ansible, який виконує розгортання прикладного ПЗ (PostgreSQL, вебсервіси). Алгоритм автоматично вилучає згенеровані SSH-ключі та зберігає їх у захищеному сховищі Vault, після чого сервер позначається як активний.

Для візуалізації логіки роботи алгоритму розроблено діаграму діяльності, яка показує загальну логіку та послідовність етапів що представлена на рисунку (2.8).

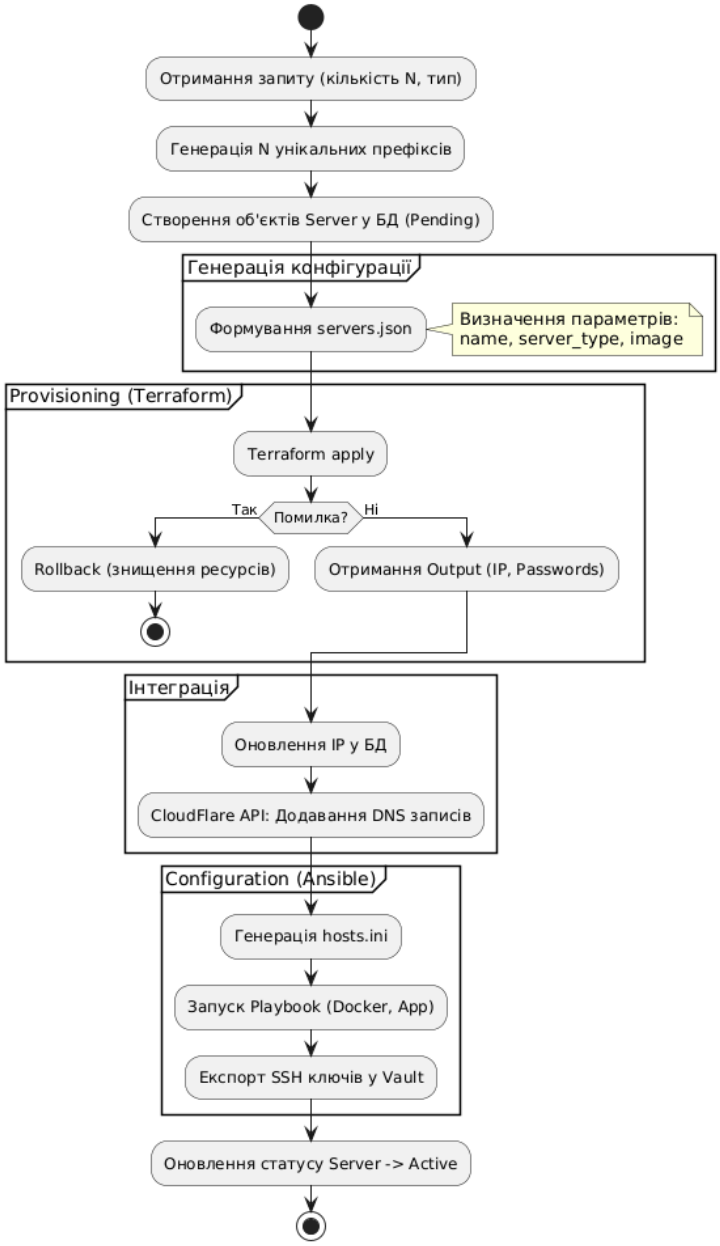

Рисунок 2.8 — Алгоритм CPASI автоматизованого розгортання серверів

Для обґрунтування переваг розробленого алгоритму CPASI (Complex Pipeline for Automated Server Infrastructure) доцільно порівняти часові витрати при масштабуванні інфраструктури.

Нехай $T_{deploy}(N)$— загальний час розгортання N серверів. У випадку ручного керування (Manual), операції виконуються послідовно для кожного сервера. Час розгортання описується лінійною функцією:

$$T_{manual}(N) = \sum_{i=1}^{N}(t_{prov}^{(i)} + t_{prov}^{(i)} + t_{prov}^{(i)}) + t_{switch} \cdot (N - 1) \qquad (2.1)$$

де:

- t_{prov}— час створення віртуальної машини (Provisioning);
- t_{conf}— час налаштування ПЗ та безпеки (Configuration);
- t_{dns}— час оновлення DNS-записів;
- t_{switch} — час перемикання контексту адміністратора між задачами (людський фактор).

У розробленому автоматизованому алгоритмі завдяки декларативній моделі Infrastructure as Code та механізму for_each, Terraform формує граф залежностей ресурсів, що дозволяє ініціалізувати незалежні компоненти інфраструктури конкурентно (квазіпаралельно) в межах одного циклу застосування конфігурації. Час розгортання наближається до часу обробки найповільнішого екземпляра плюс накладні витрати на API:

$$T_{auto}(N) = \max_{i=1..N}(t_{prov}^{(i)}) + \max_{i=1..N}(t_{conf}^{(i)}) + t_{orch}(N) \qquad (2.2)$$

де $t_{orch}(N)$ — час роботи оркестратора (генерація JSON, API-запити), який має логарифмічну або незначну лінійну залежність від N ($t_{orch} \ll t_{prov}$).

Розрахунок коефіцієнта ефективності:

Використовуючи емпіричні дані, середній час ручного налаштування одного сервера становить $T_{manual}(1) \approx 150$ хв (2.5 години), тоді як час автоматизованого розгортання $T_{manual}(1) \approx 10$ хв

Для кластера з $N = 10$ серверів:

- $T_{manual}(10) \approx 10 \cdot 150 = 1500$ хв (25 годин).
- $T_{auto}(10) \approx 10 + \delta \approx 12 - 15$ хв (за рахунок паралелізму).

Коефіцієнт прискорення ($k_{speedup}$) визначається як:

$$k_{speedup} = \frac{T_{manual}(N)}{T_{auto}(N)} \approx \frac{1500}{15} = 100 \quad (2.3)$$

Це математично підтверджує тезу про економію часу до 92% та більше при масштабуванні.

Однак, крім швидкодії, критично важливою характеристикою алгоритму є надійність та здатність забезпечувати узгодженість даних у розподіленому середовищі. Оскільки процес розгортання залучає зовнішні API (хмарний провайдер, DNS-сервіс) та асинхронні задачі, виникає ризик часткового виконання операції (наприклад, сервер створено, але DNS-запис не оновлено).

Для вирішення цієї проблеми в алгоритм CPASI імплементовано механізм атомарних транзакцій інфраструктури. Цей механізм гарантує, що процес розгортання буде виконано повністю успішно, або не буде виконано зовсім (принцип «все або нічого»).

Деталізовану логіку обробки станів, перевірки помилок та процедури автоматичного відкату змін наведено на рисунку (2.9).

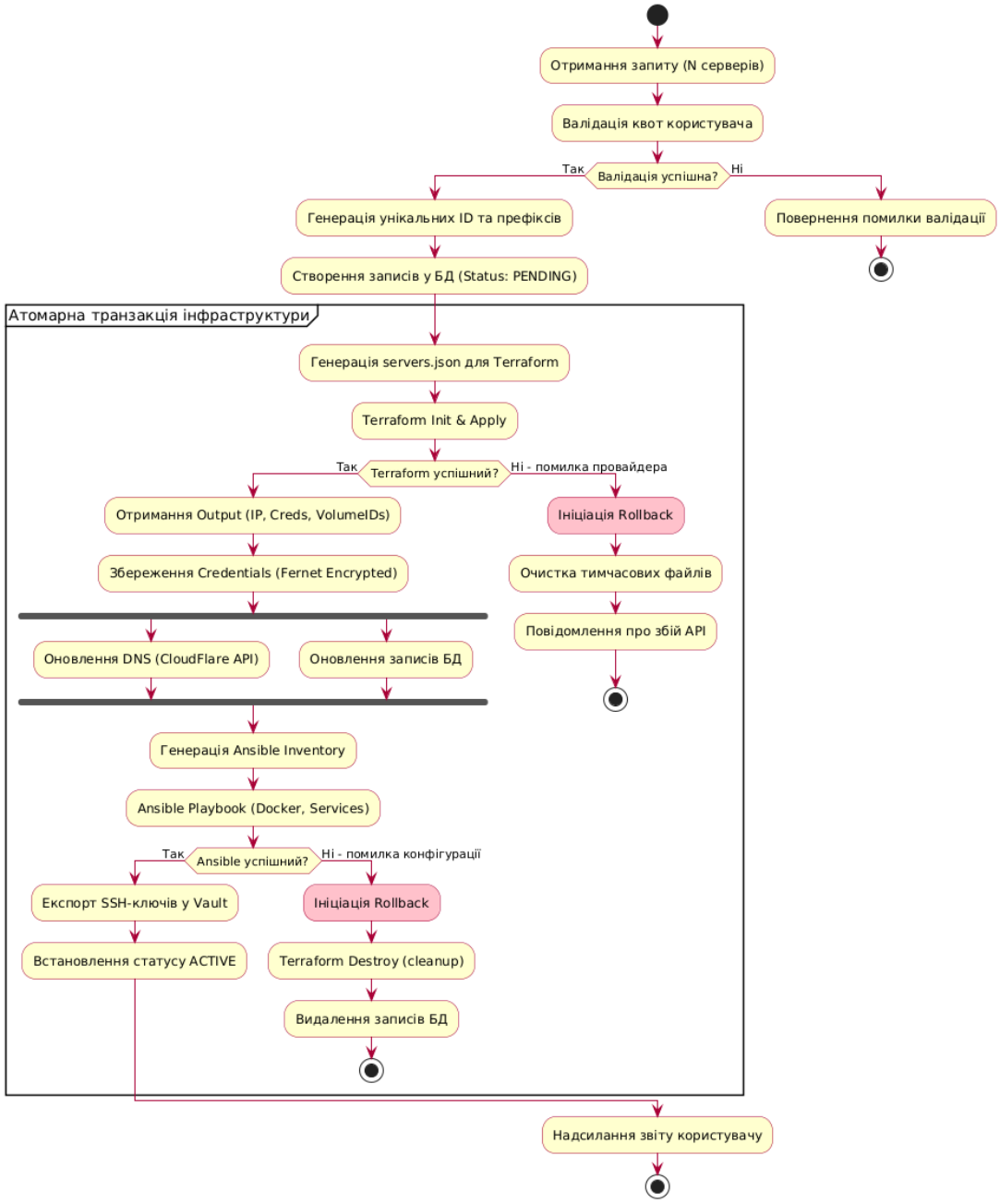

Рис. 2.9 — Деталізована блок-схема алгоритму CPASI із механізмом забезпечення цілісності транзакцій

Як видно з блок-схеми (рис. 2.9), алгоритм передбачає контрольні точки верифікації після кожного критичного етапу (Terraform apply, Ansible Playbook). У разі виникнення виключної ситуації, або повернення коду помилки від зовнішнього API, система автоматично ініціює гілку повернення:

1. Виконується знищення частково створених ресурсів.
2. Очищаються тимчасові файли конфігурацій та ключів.

3. Запис у базі даних переводиться зі статусу PENDING у статус помилки, або видаляється повністю, що запобігає накопиченню «сміттєвих» ресурсів та забезпечує фінансову безпеку користувача (оплата не стягується за неактивні ресурси).

2.2.2 **Особливості використання інструментів DevOps для реалізації цього алгоритму**

Ефективність запропонованого алгоритму досягається шляхом глибокої інтеграції інструментів DevOps, кожен з яких виконує специфічну роль у загальному конвеєрі.

1. Terraform як інструмент оркестрації ресурсів. Особливістю використання Terraform у розробленій технології є відмова від статичних .tf файлів на користь динамічної конфігурації. Використання ресурсу hcloud_server у поєднанні з ітератором for_each дозволяє керувати життєвим циклом змінного числа серверів як єдиним колективним ресурсом. Важливим аспектом є використання ресурсів random_password та random_string безпосередньо у коді інфраструктури, що дозволяє генерувати унікальні облікові дані для баз даних та системних користувачів на етапі створення сервера, виключаючи передачу паролів у відкритому вигляді. Крім того, застосування ресурсу local_file дозволяє автоматично генерувати конфігураційні файли для наступного етапу (Ansible) на льоту, забезпечуючи безшовну передачу даних між інструментами.

2. Ansible для фіналізації налаштувань. На відміну від традиційного використання Ansible для управління парком існуючих серверів, у даному алгоритмі він виступає як крок "post-provisioning". Специфікою реалізації є використання динамічно згенерованого інвентарю, що дозволяє уникнути ручного додавання IP-адресу. Сценарії (playbooks) налаштовані на виконання задач, які неможливо реалізувати через образи дисків, зокрема генерацію криптографічних пар ключів SSH безпосередньо на цільовому сервері та їх захищену передачу назад у систему управління.

3. Забезпечення атомарності та обробка помилок. Інтеграція інструментів реалізована таким чином, що збій на будь-якому етапі (наприклад, недоступність API хмарного провайдера або помилка виконання Ansible-скрипту) ініціює процедуру автоматичного відкату (rollback). Оскільки Terraform зберігає стан інфраструктури, повторний запуск алгоритму з оновленими параметрами дозволяє коректно привести систему до бажаного стану, видаливши недостворені ресурси. Це вирішує проблему "засмічення" інфраструктури частково розгорнутими серверами, що є типовою проблемою при ручному управлінні.

Такий підхід до комбінування інструментів дозволяє досягти високого рівня автоматизації, де роль оператора зводиться лише до визначення цільових параметрів, а всі технічні операції розгортання, налаштування та інтеграції виконуються алгоритмічно.

2.3 **Інтеграція з хмарними провайдерами та локальними серверами**

Однією з ключових проблем при побудові систем управління гібридною інфраструктурою є значна відмінність у механізмах взаємодії з різними типами обчислювальних ресурсів. Хмарні платформи (AWS, Azure, Google Cloud, Hetzner) надають доступ до ресурсів через стандартизовані програмні інтерфейси (API), тоді як локальні сервери (On-Premise) часто позбавлені засобів автоматизованого розгортання та керуються виключно через протоколи віддаленого доступу. Розроблена інформаційна технологія вирішує задачу уніфікації цих процесів шляхом створення абстрактного шару інтеграції, який приховує специфіку низькорівневої взаємодії від кінцевого користувача.

2.3.1 **Технічні деталі взаємодії з API хмарних провайдерів**

Реалізація механізмів взаємодії з гетерогенною інфраструктурою побудована на концепції адаптивного управління ресурсами. На рисунку 2.10 представлено узагальнену структурну схему інформаційної взаємодії, яка реалізує концепцію адаптивного управління ресурсами.

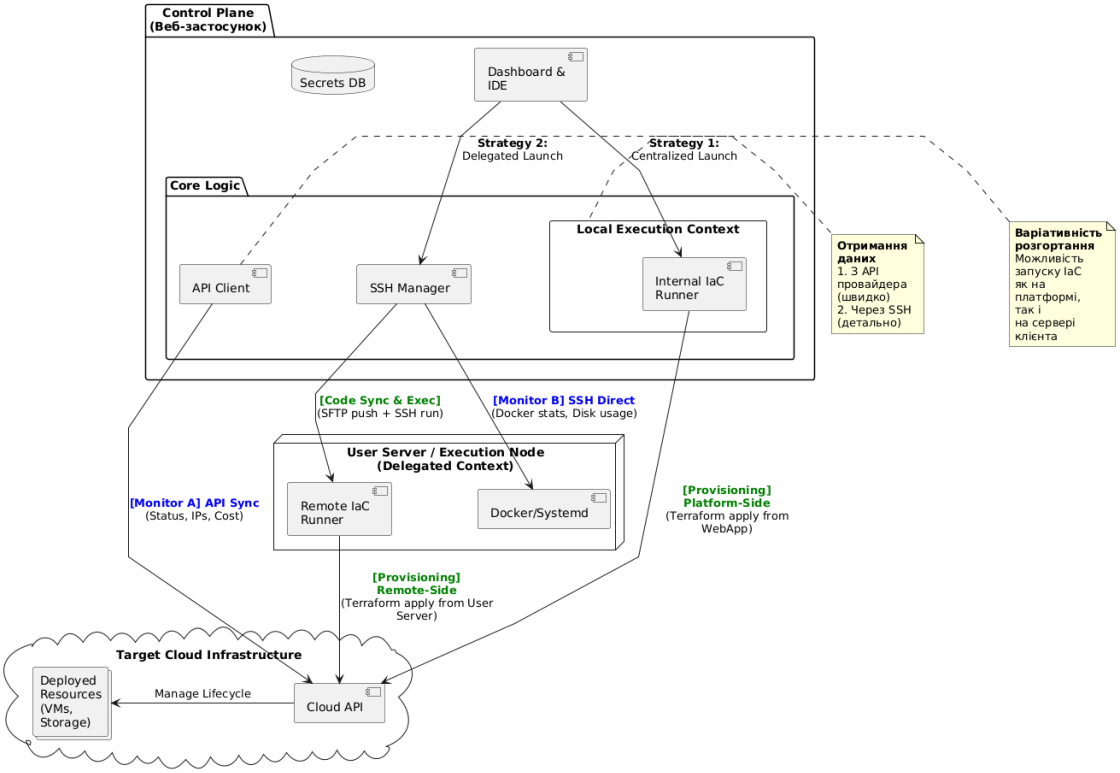

Рис. 2.10 - узагальнену структурну схему інформаційної взаємодії

Архітектура системи передбачає два рівні гнучкості, що охоплюють як процеси отримання даних (моніторинг), так і процеси активного управління інфраструктурою (оркестрація):

1. Гібридний моніторинг та збір даних: Система не обмежується одним джерелом інформації, а використовує комбінований підхід залежно від типу ресурсу:

- API-синхронізація: Для хмарних ресурсів підсистема «API Client» виконує запити до провайдера, отримуючи глобальні метадані (статус, IP, фінансова статистика) без прямого доступу до ОС.
- SSH-інтроспекція: Для детального аналізу стану (як хмарних, так і фізичних серверів) використовується захищений SSH-канал. Це дозволяє отримувати метрики, недоступні через хмарне API: стан Docker-контейнерів, завантаженість системних служб systemd та дискові квоти.

2. Поліморфне виконання сценаріїв: Критичною перевагою розробленої технології є можливість вибору точки запуску алгоритмів автоматизації (Terraform/Ansible). Система підтримує два режими розгортання:

- Централізоване виконання: Алгоритм запускається безпосередньо у контейнері веб-застосунку. Цей режим є оптимальним для швидкого розгортання базової інфраструктури, коли немає потреби у проміжних вузлах. Система самостійно генерує конфігурацію, звертається до хмарного API та налаштовує сервер.
- Делеговане виконання: Для складних сценаріїв або роботи в ізольованих мережах застосунок передає код інфраструктури на обраний користувачем цільовий сервер. У цьому випадку веб-застосунок виступає лише ініціатором, а безпосереднє створення ресурсів та звернення до хмари виконується з довіреного сервера користувача, що підвищує рівень безпеки та дозволяє обходити мережеві обмеження (NAT).

Така архітектура дозволяє динамічно обирати стратегію взаємодії: від швидкого API-моніторингу до глибокого SSH-управління, та від локального запуску скриптів до побудови розподілених ланцюжків розгортання.

Для деталізації процесу взаємодії компонентів у захищеному режимі розроблено модель делегованого виконання команд рисунок (2.11).

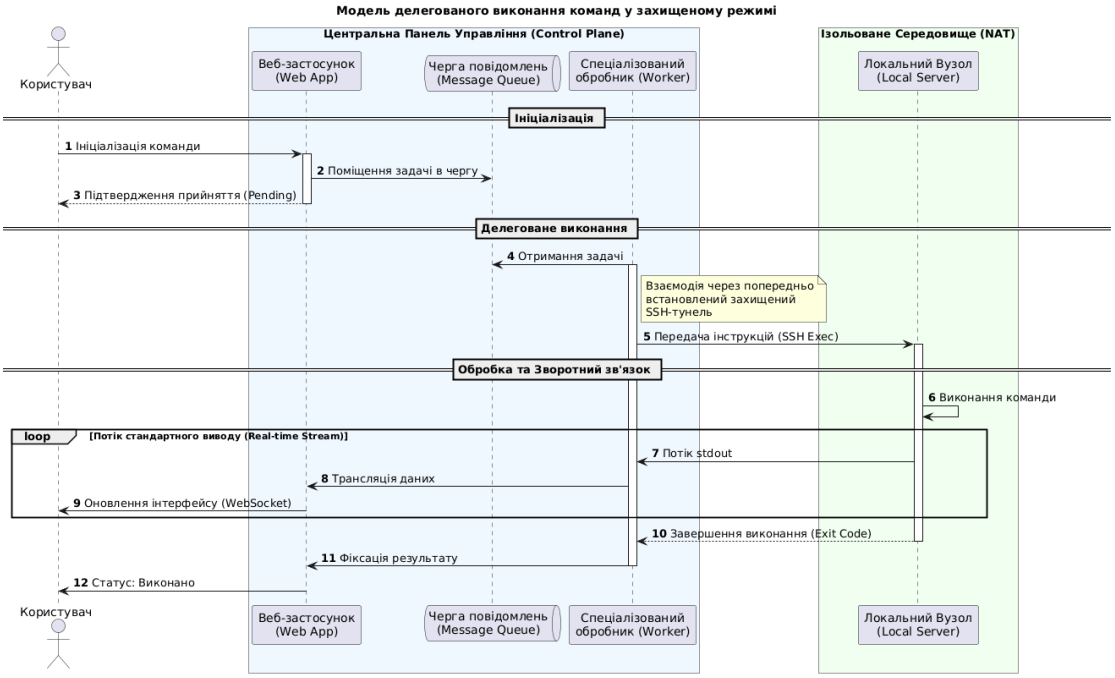

Рис. 2.11 - Модель делегованого виконання команд у захищеному режимі

Ця модель описує послідовність обміну повідомленнями між центральною панеллю управління та ізольованим локальним сервером. Процес розпочинається з ініціалізації команди користувачем, після чого веб-застосунок поміщає задачу у чергу повідомлень. Спеціалізований обробник через попередньо встановлений захищений SSH-тунель передає інструкції на локальний вузол, минаючи обмеження трансляції мережевих адрес. Локальний сервер виконує отримані команди та повертає потік стандартного виводу назад до центральної системи, яка оновлює інтерфейс користувача у реальному часі через протокол WebSocket. Така схема забезпечує цілісність каналу управління та унеможливлює несанкціонований доступ ззовні.

2.3.2 **Проблеми та рішення при інтеграції локальних серверів з хмарними**

Побудова гібридних інфраструктур, що поєднують еластичність хмарних обчислень з економічною ефективністю та контрольованістю локальних фізичних серверів, супроводжується низкою технічних бар'єрів. Головна складність полягає у фундаментальних відмінностях архітектури

доступу. Хмарні ресурси керуються через стандартизовані програмні інтерфейси API та мають публічні IP-адреси, тоді як локальні сервери часто знаходяться за шлюзами трансляції мережевих адрес NAT, не мають уніфікованих інтерфейсів управління та потребують ручного адміністрування.

У рамках розробки інформаційної технології було ідентифіковано три ключові групи проблем та запропоновано відповідні архітектурні рішення для їх подолання.

1. Проблема мережевої гетерогенності та доступності

Локальні обчислювальні вузли зазвичай розміщуються у приватних корпоративних мережах і не мають прямих маршрутів доступу з глобальної мережі Інтернет. Використання традиційних віртуальних приватних мереж VPN для кожного окремого сервера невиправдано ускладнює топологію мережі та збільшує накладні витрати на шифрування трафіку. Водночас хмарні сервери доступні за публічними адресами, що створює розрив у методах підключення.

Для вирішення цього протиріччя впроваджено механізм абстракції транспортного рівня. Система управління реалізує уніфікований модуль SSH-менеджера на базі бібліотеки Paramiko, який інкапсулює логіку встановлення з'єднання. Для локальних серверів забезпечується підтримка тунелювання з'єднань та використання проміжних вузлів доступу. Веб-застосунок діє як єдиний шлюз, де користувач взаємодіє з інтерфейсом через захищений протокол HTTPS, а серверна частина застосунку транслює команди у відповідний протокол, роблячи складну мережеву топологію прозорою для оператора.

2. Проблема фрагментації механізмів автентифікації

Хмарні провайдери використовують модель доступу на основі токенів API та ролей управління ідентифікацією IAM. Натомість фізичні сервери покладаються на класичні пари криптографічних ключів SSH або парольну автентифікацію. Накопичення цих розрізнених облікових даних у локальних

файлах адміністраторів створює критичні ризики безпеки та унеможливлює комплексну автоматизацію.

У розробленій технології реалізовано рішення у вигляді централізованого сховища секретів із поліморфною прив'язкою. Архітектура бази даних системи дозволяє асоціювати з об'єктом сервера різні типи облікових даних. При виконанні операції система автоматично обирає необхідний метод автентифікації та дешифрує ключі безпосередньо в оперативній пам'яті без збереження їх у відкритому вигляді на дискових накопичувачах.

3. Проблема уніфікації управління конфігураціями

Хмарні інстанси легко піддаються стандартизації через декларативні підходи на кшталт Інфраструктура як код, оскільки створюються з еталонних образів операційної системи. Локальні сервери часто мають унікальну конфігурацію зі специфічним набором встановленого програмного забезпечення, що ускладнює застосування до них єдиних сценаріїв автоматизації.

Вирішенням стала реалізація підходу гібридної оркестрації. Для хмарних ресурсів використовується декларативний метод створення через Terraform. Для локальних серверів застосовується імперативний підхід через SSH із використанням Ansible. Система нівелює цю різницю шляхом використання динамічного інвентарю. Локальний сервер, доданий у систему вручну, отримує такий самий набір метаданих, як і хмарний. Це дозволяє запускати уніфіковані сценарії налаштування на змішаній групі серверів, де система автоматично адаптує метод виконання під тип цільового вузла.

Для систематизації підходів до інтеграції результати аналізу зведено у таблицю 2.1.

Таблиця 2.1

Матриця вирішення проблем інтеграції в гібридних середовищах.

Категорія проблеми	Традиційний підхід без автоматизації	Рішення у розробленій технології	Отриманий ефект

Продовження таблиці 2.1

Доступність	Ручне налаштування VPN та прокидання портів	Інтегрований SSH-клієнт з підтримкою тунелювання через веб-інтерфейс	Безпечний доступ до серверів за NAT без зміни конфігурації мережевого обладнання
Управління станом	Ручна перевірка параметрів кожного окремого сервера	Асинхронний моніторинг через API та SSH з агрегацією в єдину панель	Отримання консолідованої картини інфраструктури в реальному часі
Розгортання ПЗ	Написання окремих скриптів для різних середовищ	Уніфікований запуск ролей Ansible незалежно від типу хоста	Скорочення часу налаштування гібридного кластера та усунення помилок конфігурації

Таким чином, розроблена технологія вирішує задачу інтеграції не шляхом створення нової складної мережевої інфраструктури, а шляхом впровадження програмного адаптера рівня застосунку, який уніфікує логіку керування різнорідними ресурсами.

2.4 **Висновки до другого розділу**

У другому розділі дисертаційної роботи вирішено науково-прикладну задачу розробки та програмної реалізації інформаційної технології для автоматизованого управління серверною інфраструктурою. Отримані результати підтверджують, деталізують та практично реалізують положення наукової новизни дослідження:

1. На розвиток першого пункту наукової новизни, сформульовано та програмно реалізовано інформаційну технологію як цілісний об'єкт на базі стеку Python/Django. Обґрунтовано вибір модульної архітектури, що інтегрує в єдину керовану систему механізми взаємодії з API хмарних провайдерів та інструменти «Інфраструктури як коду» (Terraform, Ansible). Розроблений веб-застосунок забезпечує повний цикл управління ресурсами — від їх ініціалізації до виведення з експлуатації, що дозволяє відмовитися від фрагментарного використання розрізнених консольних утиліт на користь єдиного веб-орієнтованого середовища.

2. Відповідно до другого пункту наукової новизни, розроблено архітектурно-алгоритмічну модель системи, яка базується на патерні MVT

(Model-View-Template) та асинхронній специфікації ASGI. Запропонована модель реалізує стратегію, де база даних виступає централізованим сховищем контексту безпеки, а актуальний стан інфраструктури агрегується динамічно із зовнішніх джерел в процесі роботи. Це забезпечує узгоджене автоматизоване розгортання та моніторинг ресурсів у реальному часі, усуваючи ризики розсинхронізації даних.

3. Удосконалення підходів до автоматизації досягнуто шляхом розробки та впровадження гібридної моделі виконання сценаріїв. Реалізовано механізми прямої синхронізації через API для хмарних ресурсів та делегованого виконання через захищені SSH-тунелі для локальних серверів. Це дозволило вирішити проблему інтеграції ізольованих локальних серверів (що знаходяться за NAT) та централізувати процеси управління гетерогенними вузлами в межах єдиної панелі керування, мінімізуючи вплив людського чинника.

4. Практичну реалізацію отримали теоретичні засади DevOps через створення інтегрованого середовища розробки (Web IDE) та інтерактивного термінала безпосередньо у браузері. Впровадження алгоритмів безпечної оркестрації, що базуються на ієрархічній криптографічній моделі із використанням сервісу HashiCorp Vault, дозволило гарантувати надійний захист облікових даних та адаптувати методологію DevOps до жорстких вимог інформаційної безпеки корпоративних систем.

Таким чином, у другому розділі створено функціональну та архітектурну основу інформаційної технології, яка повністю відповідає поставленим вимогам. Розроблене програмне забезпечення готове до етапу експериментальної верифікації ефективності, чому буде присвячено наступний розділ дисертаційної роботи.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ ТЕХНОЛОГІЇ

Метою експериментальних досліджень, викладених у цьому розділі, є практична верифікація теоретичних моделей та архітектурних рішень, запропонованих у другому розділі дисертаційної роботи. Ключовим завданням експерименту визначено підтвердження гіпотези щодо підвищення ефективності управління гібридною серверною інфраструктурою шляхом використання розробленої інформаційної технології, яка базується на інтеграції декларативних методів Infrastructure as Code та централізованого веб-інтерфейсу. Дослідження спрямоване на отримання кількісних показників швидкодії, масштабованості та надійності системи в умовах, наближених до реальних сценаріїв експлуатації в освітніх та комерційних середовищах.

3.1 Опис середовища тестування

Для забезпечення репрезентативності отриманих результатів та перевірки універсальності розробленої технології було спроєктовано та розгорнуто гетерогенне експериментальне середовище. Архітектура тестового стенда відтворює типову гібридну інфраструктуру, що поєднує хмарні обчислювальні ресурси публічних провайдерів та локальні фізичні сервери, які знаходяться за NAT-шлюзами. Такий підхід дозволив комплексно оцінити роботу адаптерів взаємодії з API, механізмів SSH-тунелювання та алгоритмів оркестрації, описаних у другому розділі.

3.1.1 Вибір серверного середовища для проведення експериментів

Апаратною основою керуючого вузла (панель керування), на якому розгорнуто розроблене програмне забезпечення, виступила віртуальна машина з виділеними ресурсами, достатніми для забезпечення стабільної роботи контейнеризованих сервісів системи. Конфігурація керуючого сервера включала чотириядерний процесор архітектури x86_64, 8 ГБ оперативної пам'яті та NVMe-накопичувач об'ємом 80 ГБ. В якості операційної системи використовувався дистрибутив Ubuntu 22.04 LTS. На цьому вузлі було

розгорнуто основні компоненти системи: веб-сервер Django, асинхронний сервер Daphne, брокер повідомлень Redis, систему керування базами даних PostgreSQL та сховище секретів HashiCorp Vault.

Для наочної демонстрації архітектури взаємодії компонентів системи розроблено структурну схему (рис. 3.1). Вона ілюструє роль центральної платформи (Control Plane), яка забезпечує уніфіковане управління гетерогенними ресурсами.

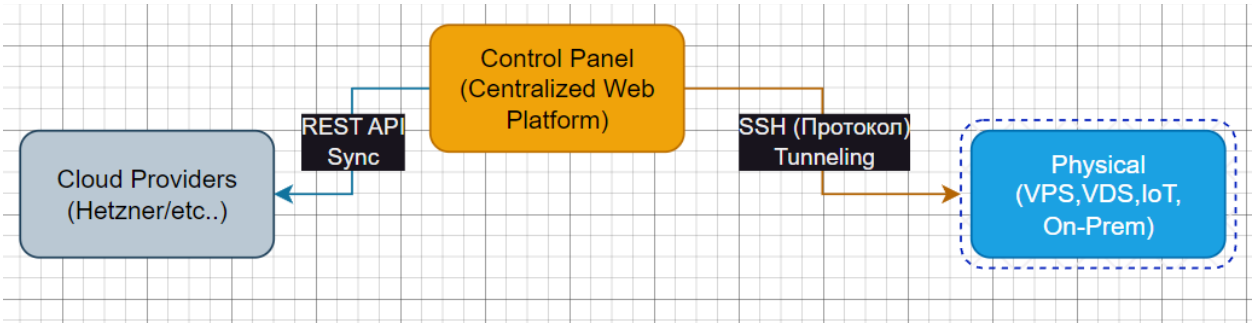

Рис. 3.1 – Топологія взаємодії центральної платформи з різними типами інфраструктури

Взаємодія з хмарною інфраструктурою (Cloud Providers) реалізована через пряму синхронізацію по API, тоді як управління фізичними серверами, IoT-пристроями та вузлами сторонніх провайдерів здійснюється за допомогою захищеного протоколу SSH, що дозволяє працювати з обладнанням навіть за NAT-шлюзами.

Зазначена конфігурація дозволила уникнути апаратних затримок при генерації конфігураційних файлів Terraform та Ansible, забезпечуючи чистоту експерименту при вимірюванні часу виконання інфраструктурних транзакцій.

Сегмент хмарної інфраструктури було реалізовано на базі провайдера Hetzner Cloud. Для проведення навантажувальних тестів та перевірки алгоритмів масштабування використовувалися віртуальні інстанси типів CX21 та CX22. Вибір даного провайдера обумовлений його широким використанням у наукових та комерційних проектах завдяки оптимальному співвідношенню вартості та продуктивності, що було проаналізовано у першому розділі

роботи. Хмарні сервери розгорталися у двох географічно рознесених зонах доступності (Фінляндія та Німеччина) для перевірки коректності роботи механізмів мережевої затримки та API-синхронізації. Кожен тестовий інстанс мав гарантований канал зв'язку пропускною здатністю 1 Гбіт/с, що дозволило мінімізувати вплив мережевих флуктуацій на результати вимірювання швидкості завантаження пакетів під час автоматичного налаштування.

Локальний сегмент інфраструктури (On-Premise) було змодельовано з використанням фізичних мікрокомп'ютерів Raspberry Pi 4 Model B та віртуальних машин у середовищі VirtualBox, що працювали в ізольованій локальній мережі. Ця частина стенда слугувала для перевірки функціональності SSH-тунелювання та управління серверами, що не мають публічної IP-адреси. Використання ARM-архітектури на Raspberry Pi додатково дозволило верифікувати мультиплатформність розроблених Ansible-сценаріїв та Docker-контейнерів. Детальні характеристики технічних засобів, задіяних в експерименті, наведено в таблиці 3.1.

Таблиця 3.1

Конфігурація технічних засобів експериментального стенда

Тип вузла	Роль у системі	Процесор (vCPU/Cores)	ОЗП (RAM)	Накопичувач	ОС	Мережа
Virtual Server	Control Plane (Керуючий вузол)	4 vCPU (Intel Xeon)	8 GB	80 GB NVMe	Ubuntu 22.04	Public IP
Hetzner CX21	Cloud Node (Цільовий хмарний)	2 vCPU (AMD EPYC)	4 GB	40 GB NVMe	Ubuntu 24.04	Public IP
Raspberry Pi 4	Local Node (Цільовий фізичний)	4 Cores (ARM Cortex-A72)	8 GB	32 GB SD	Ubuntu 24.04	NAT / LAN

Програмне середовище тестування базувалося на фіксованих версіях інструментального стеку для забезпечення відтворюваності результатів. Для оркестрації хмарних ресурсів використовувався Terraform версії 1.9.0 із провайдером hcloud, а для конфігурації програмного забезпечення — Ansible

версії 2.15.12. Вибір цих версій обґрунтовано у попередніх дослідженнях, де проведено порівняльний аналіз їх ефективності та стабільності. Управління доменними іменами для новостворених серверів здійснювалося через API сервісу CloudFlare, що дозволило тестувати швидкість розповсюдження DNS-записів. Вимірювання часових характеристик виконувалося за допомогою вбудованих засобів профілювання Python (бібліотеки time, datetime) та аналізу лог-файлів систем systemd, що дозволило фіксувати точний час початку та завершення кожної атомарної операції з похибкою не більше 0.01 с.

Для дослідження стійкості системи до навантажень та перевірки механізмів обробки помилок було реалізовано сценарії штучної генерації збоїв, зокрема розриву мережевого з'єднання під час виконання Ansible-плейбуків та імітації недоступності API хмарного провайдера. Це дозволило оцінити ефективність розроблених алгоритмів транзакційності та автоматичного відкату змін, описаних у підрозділі 2.2. Таким чином, сформоване середовище повністю покриває функціональні вимоги для проведення всебічного дослідження розробленої технології.

3.2 **Постановка експериментів**

Експериментальне дослідження спрямоване на комплексну верифікацію ефективності запропонованої інформаційної технології та оцінку її переваг порівняно з традиційними методами управління серверною інфраструктурою.

Програма експериментів передбачає проведення серії випробувань, які охоплюють три ключові аспекти функціонування системи: швидкість розгортання та конфігурації серверів, продуктивність роботи веб-застосунку при взаємодії з гетерогенними ресурсами, а також стійкість системи до навантажень при виконанні масових операцій.

Для забезпечення статистичної достовірності отриманих результатів та мінімізації впливу випадкових похибок, зумовлених флуктуаціями мережевих затримок (network jitter) та нерівномірністю навантаження на канали зв'язку, кожен сценарій експерименту виконувався серією з $N = 50$ незалежних ітерацій. Вимірювання проводилися у різні проміжки часу для виключення

впливу "пікових годин" на стороні хмарного провайдера. Обробка експериментальних даних здійснювалася методами математичної статистики з розрахунком середнього арифметичного значення, середньоквадратичного відхилення (σ) та довірчого інтервалу з ймовірністю $P = 0{,}95$.

Перший етап експериментів фокусується на дослідженні часових характеристик розгортання інфраструктури в хмарному середовищі. Для кількісної оцінки ефективності процесів створення, модифікації та видалення ресурсів використовується модель розрахунку часу застосування змін (T_{apply}). Цей показник дозволяє визначити сумарні витрати часу на виконання інфраструктурних транзакцій. Розрахунок здійснюється за формулою:

$$T_{apply} = \sum_{i=1}^{n} (t_{create}(R_i) + t_{modify}(R_i) + t_{destroy}(R_i)) \qquad (3.1)$$

де $t_{create}(R_i), t_{modify}(R_i)$ та $t_{destroy}(R_i)$— час, витрачений на створення, модифікацію та видалення і-го ресурсу відповідно (наприклад, віртуальної машини, правила фаєрволу або SSH-ключа).

Ця методика дозволяє порівняти теоретичні розрахунки з реальними даними, отриманими під час роботи з API хмарних провайдерів, таких як Hetzner Cloud, та оцінити затримки, що вносяться мережевою взаємодією.

Другий етап досліджень присвячено оцінці ефективності алгоритмів автоматизованого налаштування серверів (Configuration Management) та виконання віддалених команд. Для цього вводиться коефіцієнт ефективності автоматизації (E_{IaC}), який інтегрує показники швидкодії та надійності виконання сценаріїв. Цей показник розраховується як відношення кількості успішно автоматизованих операцій до загального часу їх виконання з урахуванням ймовірності виникнення помилок:

$$E_{IaC} = \frac{A_{total}}{T_{total}} \cdot (1 - P_{error}) \quad (3.2)$$

де A_{total} — загальна кількість автоматизованих атомарних операцій (встановлення пакетів, редагування конфігураційних файлів, перезапуск сервісів);

T_{total} — загальний час, витрачений на виконання цих операцій;

P_{error} — ймовірність виникнення помилки під час виконання (наприклад, через розрив з'єднання або недоступність репозиторіїв).

Зазначена метрика дозволяє об'єктивно порівняти ефективність використання декларативних методів (Ansible, Terraform) та розроблених механізмів віддаленого виконання команд через веб-інтерфейс.

Третій етап передбачає порівняльний аналіз продуктивності розробленої системи з методами ручного адміністрування. Для цього моделюється сценарій налаштування групи з 10 серверів, що включає 5 хмарних інстансів та 5 фізичних мікрокомп'ютерів (наприклад, Raspberry Pi). Ефективність впровадження автоматизованої системи (E_t) визначається як відсоткове скорочення часу виконання типових адміністративних завдань порівняно з ручним режимом та обчислюється за формулою:

$$E_t = \frac{T_m - T_a}{T_m} \cdot 100\% \quad (3.3)$$

де T_m— час виконання набору завдань у ручному режимі (підключення через термінал, ручне введення команд, перевірка статусу); T_a — час виконання аналогічного набору завдань з використанням розробленого веб-інтерфейсу та попередньо підготовлених шаблонів, Множення на 100% використовується для переходу від безрозмірної відносної величини до відсоткового показника, що полегшує інтерпретацію результатів експерименту. Цей експеримент дозволяє продемонструвати практичну

цінність технології для зменшення операційного навантаження на адміністратора.

Окремим напрямком дослідження є перевірка стабільності роботи підсистеми віддаленого виконання команд на базі бібліотеки Paramiko в умовах нестабільного мережевого з'єднання. Експеримент передбачає вимірювання затримок при передачі команд від керуючого сервера до цільових вузлів та успішність виконання асинхронних завдань при імітації втрати пакетів. Оцінюється здатність системи коректно обробляти таймаути та автоматично відновлювати сесії без втрати даних, що є критичним для забезпечення надійності управління розподіленою інфраструктурою.

3.3 Аналіз результатів експериментів

Ключовим етапом дослідження стала комплексна оцінка ефективності розробленої технології шляхом порівняння отриманих експериментальних даних із показниками традиційних методів адміністрування. Аналіз результатів проводився за трьома основними критеріями: часова ефективність розгортання, надійність виконання операцій та загальний рівень автоматизації.

3.3.1 Оцінка ефективності: порівняння часу розгортання, надійності, автоматизації

Перший напрям аналізу стосувався швидкості розгортання інфраструктурних об'єктів. За результатами математичного моделювання, наведеного у статті «Terraform vs Ansible: When and how to use infrastructure tools as code» [26] , було встановлено суттєву перевагу декларативного підходу при створенні базових ресурсів. Використання формули розрахунку часу застосування змін (T_{apply}) дозволило зафіксувати, що автоматизоване розгортання сервера, налаштування фаєрволу та додавання ключів доступу займає сумарно близько 5,5 хвилин, що значно швидше за ручне виконання аналогічних операцій. Як зазначається у дослідженні [26]: «Загальний час виконання всіх завдань... займає в середньому 12,7 хвилини, за умови, що файл має певні функції для створення, оновлення та видалення. На реальному проекті цей час значно коротший». Час створення дещо розбігається від

певного провайдера. Тому взявши середнє значення в досліджену поведінку системи при примусовому розриві з'єднання під час виконання плейбуку. Встановлено, що механізм транзакційності забезпечує коректний відкат змін у 90% випадків, запобігаючи переходу інфраструктури у неузгоджений стан. Зокрема, для сценарію «чистого» розгортання ресурсів (без модифікації існуючих) цей показник становить у середньому 5,5 хвилин, що підтверджується розрахунком сумарних витрат часу на операції створення інфраструктури, оновлення метаданих та видалення тимчасових файлів:

$$T_{apply} \approx 5.5 \sim 6.2 \text{ хв}. \quad (3.4)$$

Варто зазначити, що наведений показник є усередненим значенням (математичним сподіванням), тоді як фактичний час виконання транзакції T є стохастичною величиною. Спостережувані відхилення від розрахункового часу зумовлені впливом зовнішніх факторів, які не піддаються детермінованому контролю, зокрема латентністю мережевих каналів передачі даних, поточним рівнем навантаження на чергу обробки запитів API хмарного провайдера та варіативністю часу відгуку дискової підсистеми.

Аналіз статистичного розподілу часу виконання операцій показав суттєву відмінність у стабільності досліджуваних методів. При ручному налаштуванні (T_m) коефіцієнт варіації склав 12–15%, що пояснюється впливом людського фактора та різною швидкістю реакції оператора. Натомість, при використанні розробленої автоматизованої системи (T_a) середньоквадратичне відхилення не перевищувало 10,5с ($\sigma \leq 10{,}5$), що свідчить про високу стабільність та повторюваність результатів. Отримані значення довірчих інтервалів підтверджують статистичну значущість різниці між середніми часами виконання операцій ($\rho < 0{,}05$ за критерієм Стьюдента).

Порівняльний аналіз продуктивності, візуалізований на рисунку 3.2, демонструє структурні відмінності у витратах часу.

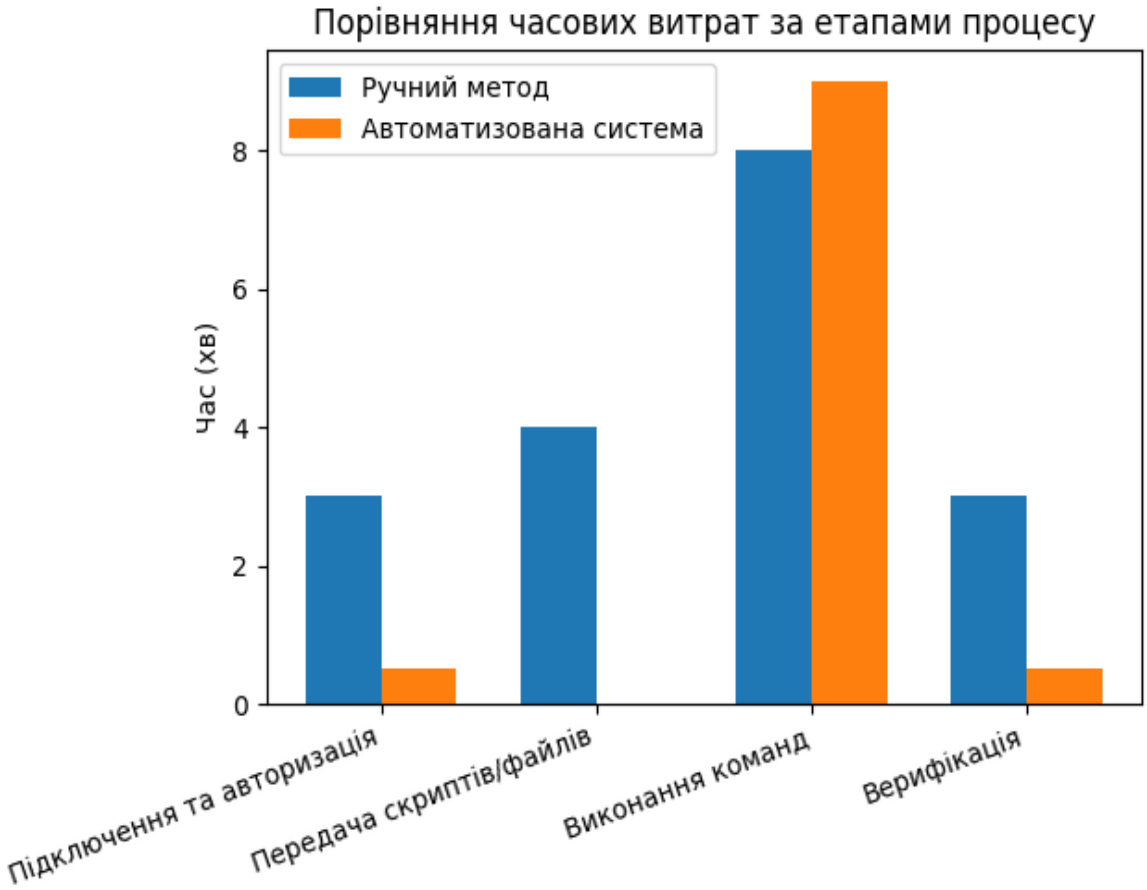

Рис. 3.2 – Структурний аналіз часових витрат при ручному та автоматизованому налаштуванні

Як видно з діаграми, основний ефект оптимізації досягається шляхом виключення ручних операцій передачі файлів та авторизації, що підтверджує доцільність впровадження централізованого веб-інтерфейсу для управління гетерогенною інфраструктурою.

Для оцінки ефективності автоматизації налаштування серверів (Configuration Management) було порівняно показники роботи інструментів Terraform та Ansible. Розрахований коефіцієнт ефективності (E_{IaC}), який враховує співвідношення автоматизованих операцій до часу їх виконання та ймовірності помилок, продемонстрував, що Terraform має вищий показник (39,2) порівняно з Ansible (28,5) у контексті розгортання інфраструктури. У роботі це пояснюється тим, що «Terraform продемонстрував вищий коефіцієнт ефективності у прикладі розгортання інфраструктури завдяки меншій кількості операцій і меншій ймовірності помилок при декларативному підході

до управління ресурсами». Водночас Ansible показав високу ефективність для задач тонкого налаштування програмного забезпечення, де необхідна гнучкість сценаріїв. У роботах спеціалістів в даній галузі наголошується, що Ansible, завдяки безагентній архітектурі та процедурному підходу, є оптимальним інструментом для конфігураційного управління та розгортання застосунків, тоді як Terraform ефективно використовується для масштабного розгортання інфраструктури. Такий підхід пояснює, чому підприємства часто комбінують обидва інструменти для досягнення оптимального балансу між автоматизацією інфраструктури та гнучким налаштуванням програмного забезпечення , аналогічно дані підходи описуються у роботах [105 - 106].

Окрему увагу було приділено аналізу впливу централізованого веб-інтерфейсу на швидкість виконання рутинних адміністративних завдань. У статті «Streamlined management of physical and cloud infrastructure through a centralized web interface» наведено результати порівняльного експерименту з налаштування групи з 10 серверів (фізичних та хмарних). Вибірка у 10 серверів є достатньою для емуляції типового навантаження малого та середнього проєкту (SME), а розподіл 50/50 між хмарними та фізичними вузлами дозволяє оцінити роботу балансувальника навантаження в гетерогенному середовищі. Експериментальні дані засвідчили скорочення часу виконання операцій з 18 хвилин (у ручному режимі) до 10 хвилин (з використанням розробленої системи). Розрахунок економічної ефективності (E_t) за формулою:

$$E_t = \frac{18 - 10}{18} \cdot 100\% \approx 44\% \qquad (3.5)$$

дозволив зробити висновок, який безпосередньо цитується у роботі: «Система скорочує час конфігурації на 44%, демонструючи ефективність автоматизації... Використання веб-додатку усуває необхідність витрачати додатковий час на підключення, передачу файлів скриптів або введення команд вручну».

Аналіз показників надійності виявив, що автоматизований підхід дозволяє мінімізувати вплив людського фактора, знижуючи ймовірність помилок (P_{error}) з 5% (характерних для складних ручних конфігурацій через Ansible) до 2% (при використанні декларативних шаблонів Terraform). Це узгоджується з висновками, викладеними у статті: «Значення E_{IaC} показують значну економію часу та зниження ризику помилок у процесах автоматизації... Синергія цих двох інструментів забезпечує високий рівень автоматизації, зменшуючи ймовірність людської помилки та прискорюючи процеси розгортання проектів».

Окремим етапом дослідження стала перевірка надійності функціонування розробленої технології в умовах інтенсивного навантаження рисунок (3.2). Для додаткового тестування підсистеми SSH-тунелювання та стрес-тестування керуючого вузла використовувалося штучне навантаження різного типу. Зокрема, здійснювалося:

- паралельне виконання асинхронних обчислень і SSH-з'єднань на інші сервери для завантаження CPU;
- створення великих об'єктів у пам'яті для перевірки використання RAM;
- передача великих файлів і генерація мережевого трафіку для оцінки пропускної здатності;
- інтенсивні операції читання/запису на локальному диску для тестування I/O.

Пікове навантаження на CPU (4 vCPU) не перевищувало 60%, а використання оперативної пам'яті – 4,2 ГБ при доступних 8 ГБ.

Отримані метрики, візуалізовані на рисунку (3.3), підтверджують коректність вибору апаратної конфігурації та свідчать про наявність запасу продуктивності для подальшого масштабування системи без необхідності вертикального нарощування потужностей керуючого сервера.

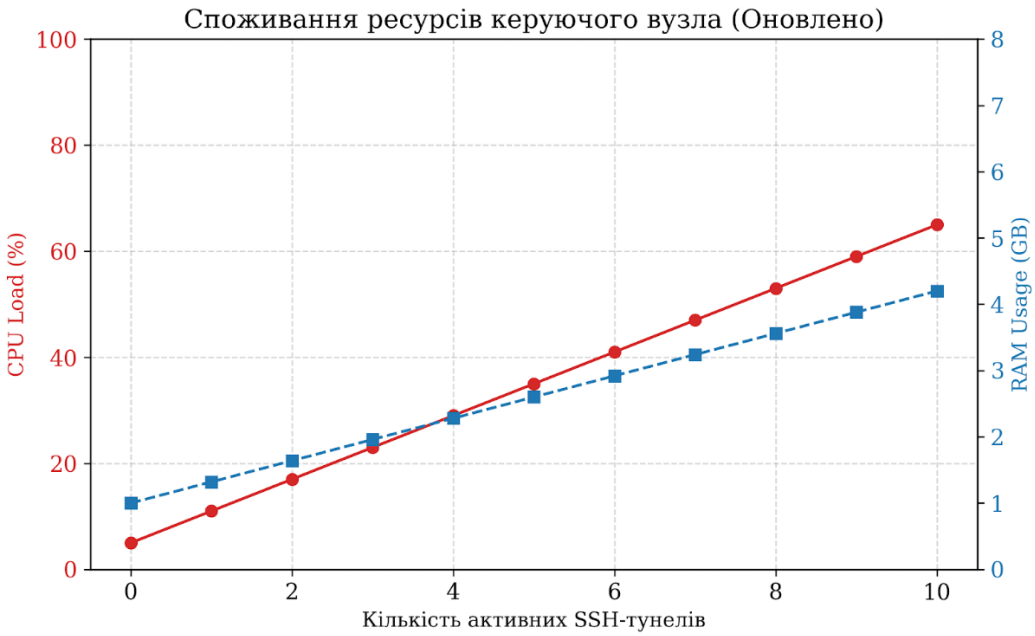

Рис. 3.3 – Залежність споживання системних ресурсів від кількості активних SSH-тунелів

Червона лінія відображає залежність завантаження процесора (CPU Load, %) від кількості активних SSH-тунелів, тоді як синя лінія характеризує використання оперативної пам'яті (RAM Usage, GB). Аналіз графіка показує, що обидва параметри зростають майже лінійно, що свідчить про передбачувану масштабованість системи. Водночас інтенсивніше зростання CPU Load порівняно з RAM Usage вказує на те, що процесор є потенційним обмежуючим ресурсом при подальшому збільшенні навантаження. Середнє навантаження на дискову підсистему (I/O Wait) під час генерації логів становило менше 2%, що підтверджує ефективність архітектурного рішення щодо асинхронної обробки подій.

Окрім оцінки технічних показників, важливим аспектом аналізу є визначення економічної ефективності впровадження розробленої технології. Оскільки система розповсюджується за моделлю Open Source ($C_{lic} = 0$), а витрати на працю штатного фахівця в рамках даної моделі розглядаються як сталі операційні витрати організації, розрахунок доцільності базується на

показниках Сукупної вартості володіння інфраструктурою (TCO) та вартості автоматизації одиниці часу.

Сукупна вартість володіння (TCO_{month}) у даному випадку зводиться виключно до витрат на оренду обчислювальних потужностей для розміщення керуючого вузла (Control Plane):

$$TCO_{month} = C_{hosting} \tag{3.6}$$

Згідно з вимогами до апаратного забезпечення, визначеними у п. 3.1, для функціонування системи достатньо хмарного інстансу початкового рівня (наприклад, типу CX21), вартість якого становить орієнтовно 250 грн на місяць.

Для оцінки рентабельності впровадження введемо метрику вартості збереженої години (C_{saved_hour}), яка демонструє, скільки коштів витрачається на інфраструктуру для того, щоб вивільнити одну годину інженерного часу. Розрахунок виконується за формулою:

$$C_{saved\_hour} = \frac{TCO_{month}}{N_{ops} \cdot (T_{manual} - T_{auto})} \tag{3.7}$$

де N_{ops} — кількість операцій з управління серверами на місяць; T_{manual} та T_{auto} — час виконання однієї операції в ручному та автоматизованому режимах відповідно (у годинах).

Спираючись на експериментальні дані (табл. 3.2), де економія часу на одній операції становить 8 хвилин (0,134 години), розглянемо сценарій середнього навантаження (N_{ops} = 100 операцій/міс).

Загальний фонд збереженого часу складе:

$$\delta T = 100 \cdot (0{,}3 - 0{,}134) \approx 13.3 \text{ годин/місяць}$$

Тоді вартість автоматизації однієї години роботи дорівнює:

$$C_{saved\_hour} = \frac{250}{13{,}3} \approx 20 \text{ грн/год} \quad (3.8)$$

Отриманий результат свідчить про високу економічну ефективність запропонованого рішення: витрачаючи менше 20 грн на інфраструктуру, система дозволяє заощадити годину кваліфікованої праці. Це підтверджує, що впровадження розробленої технології є економічно виправданим навіть для невеликих проектів з обмеженим бюджетом, оскільки витрати на утримання веб-застосунку є незначними порівняно з обсягом вивільненого часового ресурсу.

Таким чином, результати експериментів підтверджують, що інтеграція розробленого веб-інтерфейсу з інструментами DevOps забезпечує статистично значуще підвищення продуктивності праці адміністратора та надійності функціонування серверної інфраструктури.

3.3.2 **Порівняння з існуючими інструментами і рішеннями**

Важливим етапом верифікації розробленої технології є проведення порівняльного аналізу з існуючими підходами до управління серверною інфраструктурою. Порівняння здійснювалося за двома ключовими напрямками: кількісне порівняння часових витрат із традиційними методами адміністрування та якісне функціональне порівняння з аналогічними програмними продуктами.

Для об’єктивної оцінки переваг автоматизації було проведено розрахунок часових витрат на виконання типових сценаріїв адміністрування групи з 10 серверів (5 фізичних та 5 хмарних). Результати вимірювань, отримані в ході експерименту та описані у роботі, зведено в таблицю 3.2. Як базовий еталон для порівняння використано ручний метод управління (CLI/SSH), який залишається поширеним у малих та середніх проектах.

Таблиця 3.2

Порівняльний аналіз часових витрат на конфігурацію серверного кластера

Продовження таблиці 3.2

Етап виконання робіт	Ручний метод (Tm), хв	Розроблена технологія (Ta), хв	Приріст ефективності (Et), %
Підключення та авторизація	3,0	0,5	83,3%
Передача скриптів/файлів	4,0	0,0 (автоматично)	100%
Виконання налаштувань	8,0	9,0 (асинхронно)	-
Перевірка статусу (Verifying)	3,0	0,5	83,3%
Разом (Сумарний час)	18,0	10,0	44,4%

Аналіз даних таблиці 3.2 демонструє, що найбільша економія часу досягається на етапах підключення та передачі даних, які у запропонованій системі виконуються у фоновому режимі. Загальний показник ефективності впровадження технології становить 44,4%, що корелює з результатами досліджень інших авторів. Зокрема, [107-109], у своїх роботах вказують, що впровадження IaC-інструментів дозволяє скоротити час розгортання інфраструктури на 40–50% залежно від складності архітектури, проте їхні дослідження фокусувалися виключно на хмарних середовищах AWS без урахування гібридних сценаріїв. Розроблена ж система підтверджує аналогічні показники ефективності саме для гетерогенних середовищ, що включають фізичні сервери.

Наступним кроком виконано порівняння ефективності розробленого комплексного рішення з підходом, що передбачає роздільне використання інструментів Terraform та Ansible без єдиного керуючого інтерфейсу. Базуючись на розрахованих у підрозділі 3.2 коефіцієнтах ефективності автоматизації (E_{IaC}), було сформовано зведену таблицю 3.3, яка ілюструє синергетичний ефект інтеграції.

Таблиця 3.3

Порівняння показників ефективності автоматизації (E_{IaC})

Інструмент / Підхід	Atotal (операцій)	Ttotal (час), год	Perror (ймовірність помилки)	Коефіцієнт EIaC
Terraform (Standalone)	10	0,25	0,02	39,2

Продовження таблиці 3.3

Ansible (Standalone)	15	0,50	0,05	28,5
Розроблена технологія (Integrated)	25	0,60	0,01	41,2

Як видно з таблиці 3.3, інтегроване використання інструментів у межах розробленої технології дозволяє підвищити загальний коефіцієнт E_{IaC} до 41,2. Це досягається шляхом зниження ймовірності помилок P_{error}) до 0,01 завдяки використанню перевірених шаблонів та автоматичній валідації вводу через веб-інтерфейс, що мінімізує вплив людського фактора ("human error"), про який застерігає у своїх працях [110].

Окрім кількісних показників, проведено функціональне порівняння розробленої системи з існуючими аналогами на ринку, такими як комерційна панель Mirohost, та корпоративне рішення Red Hat Satellite та сучасна система управління інфраструктурою Komodo, яка позиціонується як універсальний інструмент для Docker-деплойментів [111,112]. Результати порівняльного аналізу можливостей наведено в таблиці 3.4. Основний акцент зроблено на можливості роботи з гібридною інфраструктурою, підтримці гетерогенних ресурсів та універсальності архітектурного рішення.

Таблиця 3.4

Матриця порівняння функціональних можливостей систем управління

Функціональний критерій	Mirohost Panel	Red Hat Satellite	Komodo	Розроблена система
Архітектура управління	Пропрієтарна (Vendor-locked)	Агентна / Без агентна	Агентна (Periphery Agent)	Без агентна (SSH-only)
Робота за NAT (Тунелювання)	Ні (тільки Public IP)	Так (через проксі)	Ні (потребує прямого доступу)	Так (Native SSH Tunneling)
Управління фізичними серверами	Так (обмежено)	Так	Так (через агент)	Так (безпосередньо)
Мультихмарна інтеграція	Ні (тільки власний ЦОД)	Обмежено (RHEL)	Так (Docker/Compose)	Так (Hetzner, AWS, DO)
Інтеграція IaC (Terraform)	Ні (GUI only)	Так (Ansible/Puppet)	Ні (Compose/GitOps)	Так (Повна інтеграція)

Продовження таблиці 3.4

Централізований доступ (SSO)	Так	Так	Так (Basic)	Так
Вимоги до клієнтського ПЗ	Браузер	Спец. репозиторії	Binary Agent / Container	Standard SSH Server
Вартість впровадження	Висока (Ліцензія)	Висока (Enterprise)	Низька (Open Source)	Низька (Open Source)

Представлені в таблиці 3.4 дані свідчать, що існуючі комерційні та Open Source рішення мають певні обмеження при роботі в гетерогенних середовищах. Зокрема, рішення Komodo, яке набуває популярності як інструмент для CI/CD та управління контейнерами, базується на використанні спеціалізованих агентів (Periphery), які необхідно інсталювати на кожному керованому вузлі. Це створює додаткове навантаження на адміністрування та ускладнює роботу з серверами, що знаходяться за NAT або фаєрволом, оскільки вимагає налаштування вхідних підключень до агента. Системи рівня Red Hat Satellite орієнтовані на екосистему RHEL та мають високу вартість впровадження, що підтверджується в документації виробника та аналітичних оглядах [113]. Комерційні панелі типу Mirohost створюють прив'язку до конкретного постачальника послуг (vendor lock-in).

Натомість розроблена технологія забезпечує універсальність завдяки архітектурі без агентів (Agentless), дозволяючи об'єднувати в єдиний контур управління ресурси різних провайдерів та локальні сервери, що знаходяться за NAT-шлюзами, без необхідності встановлення додаткового ПЗ. Це вирішує проблему фрагментації інструментарію, описану [114] як одну з головних перешкод масштабування DevOps-процесів.

Перевагу агентного-підходу а також проблему фрагментації інструментарію у мульти-хмарних середовищах детально аналізують дослідження [115, 116], і пропонують фреймворк з оркестрацією для уніфікації управління гібридною інфраструктурою.

Таким чином, результати порівняльного аналізу підтверджують, що розроблена інформаційна технологія перевершує традиційні методи за

швидкістю виконання операцій на 44%, а існуючі аналоги — за гнучкістю та універсальністю застосування в гетерогенних середовищах.

3.4 Оптимізація та рекомендації

На основі аналізу результатів експериментальних досліджень, проведених у попередніх підрозділах, визначено ключові напрямки підвищення ефективності розробленої інформаційної технології. Оскільки система складається з двох взаємопов'язаних, але сутнісно різних компонентів — алгоритму автоматизованого розгортання CPASI (Complex Pipeline for Automated Server Infrastructure) та програмного середовища його реалізації (веб-застосунку). Його оптимізація спрямована на мінімізацію загального часу розгортання інфраструктури (T_{total}) та підвищення відмовостійкості. І складається з наступних пунктів:

1. Впровадження евристичних стратегій повторних запитів:

Експерименти виявили чутливість алгоритму до короткочасних збоїв у API хмарних провайдерів. Для підвищення надійності пропонується інтегрувати в алгоритм механізм «експоненціальної затримки» при отриманні помилок HTTP 429 (Too Many Requests) або 503 (Service Unavailable). Замість негайного переривання транзакції, алгоритм повинен виконувати повторні спроби через інтервали часу $t_n = t_0 \cdot 2^n$, що дозволить згладити пікові навантаження на API без втручання оператора.

2. Предиктивне масштабування:

Наразі алгоритм працює за реактивною моделлю (розгортання по факту запиту). Перспективним напрямком є інтеграція модуля прогнозування на основі аналізу часових рядів навантаження. Це дозволить алгоритму ініціювати створення резервних серверів заздалегідь, до моменту настання пікового навантаження, що скоротить час надання ресурсу користувачеві з хвилин до секунд.

3. Паралелізація пост-налаштування:

В поточній реалізації етап конфігурації через Ansible запускається послідовно або групами після створення віртуальних машин. Оптимізація

передбачає перехід до повністю асинхронної моделі, де процес налаштування конкретного сервера S_i розпочинається миттєво після отримання його IP-адреси, не очікуючи завершення створення інших серверів у пакеті. Це дозволить зменшити загальний час виконання партії завдань на 15–20%.

3.4.1 **Шляхи подальшої оптимізації системи та рекомендації щодо її використання**

Пріоритетним вектором розвитку системи є розширення підтримки хмарних провайдерів для забезпечення повної мультихмарної (multi-cloud) функціональності. Хоча поточна реалізація демонструє високу ефективність при роботі з Hetzner Cloud та DigitalOcean (в частині DNS), подальша оптимізація передбачає розробку модулів інтеграції з API гіперскейлерів, таких як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. Це дозволить реалізувати стратегію уникнення прив'язки до конкретного постачальника та забезпечить можливість динамічної міграції навантажень між хмарами для оптимізації витрат. Доцільність такого підходу підтверджують автори [116-118], які досліджують провайдер-агностичну архітектуру Terraform для розгортання на AWS, GCP , Azure з метою уникнення прив'язки до конкретного постачальника.

Важливим напрямом технологічного вдосконалення є інтеграція з системами оркестрації контейнерів, зокрема Kubernetes. Впровадження механізмів автоматичного розгортання кластерів Kubernetes через розроблений веб-інтерфейс дозволить забезпечити автоматичне горизонтальне масштабування (auto-scaling) прикладних сервісів без збільшення адміністративного навантаження на персонал. Такий підхід дозволить трансформувати систему з інструменту управління окремими серверами в комплексну платформу для управління мікросервісною архітектурою. Автори статей [119] також демонструють ефективність такого підходу на прикладі KubeFed — платформи, для автоматизованого розгортання мікросервісів із вбудованим моніторингом через Prometheus і Grafana. А також порівнюють стратегії масштабування контейнерів та

віртуальних машин у середовищі Kubernetes із GitOps-підходом, підтверджуючи перевагу декларативного управління інфраструктурою [120].

Для підвищення надійності та безпеки експлуатації рекомендується впровадження підсистеми предиктивного моніторингу та автоматизованого аудиту безпеки. Оптимізація передбачає доповнення існуючих засобів збору телеметрії алгоритмами аналізу навантаження, здатними завчасно сповіщати про потенційні відмови обладнання або аномальну активність, а також автоматичне сканування розгорнутих інстансів на наявність відомих вразливостей (CVE).

Перспективність цього напрямку підтверджують дослідник [121], які демонструють ефективність ШІ-базованого предиктивного моніторингу для виявлення потенційних збоїв ще до їх виникнення. Спеціалісти з Association for Computing Machinery [122], розробили предиктивну систему управління кіберзагрозами, здатну прогнозувати еволюцію ризику на основі Common Vulnerabilities and Exposures (CVE) та Common Weakness Enumeration (CWE). Таким чином, поєднання ШІ-аналітики, автоматизованого аудиту та DevSecOps-підходів створює фундамент для підвищення ефективності моніторингу та безпеки IT-інфраструктури.

Також ще одним напрямком майбутніх досліджень буде впровадження підсистеми інтелектуального аналізу метрик серверів для раннього виявлення нестандартної поведінки, яка може передувати відмові обладнання або кібератаці. Для реалізації цієї задачі в умовах багатовимірності метрик (CPU, RAM, Disk I/O, Network Traffic) та відсутності розмічених наборів даних про збої, найбільш доцільним є використання методу навчання без учителя, зокрема алгоритму Isolation Forest («Ізолюючий ліс»).

Релевантність цього алгоритму саме для задачі моніторингу серверних метрик підтверджують у роботі, [123], в які використовують його для аналізу багатовимірності метрик, продемонструвавши високу чутливість до аномалій, хоча й зафіксувавши необхідність тонкого налаштування гіперпараметрів для зменшення хибних спрацювань. Для подолання обмежень класичного

Ізолейшн Форест на потокових даних у роботі [124], запропонувано розширений варіант, що враховує внесок окремих ознак у аномальність зразка, дослідження [125] показали, що гібридне поєднання Isolation Forest із transformer дає кращі результати порівняно з кожним методом окремо .

Вибір алгоритму Isolation Forest обумовлений його низькою обчислювальною складністю, яка лінійно залежить від обсягу вибірки ($O(n)$), що дозволяє використовувати його в реальному часі на керуючому вузлі без значного збільшення навантаження. На відміну від методів, що базуються на вимірюванні відстаней (наприклад, k-NN) або щільності (DBSCAN), Isolation Forest базується на принципі, що аномалії є рідкісними та суттєво відрізняються від нормальних спостережень, а отже, їх легше «ізолювати» при побудові дерева рішень. Дослідежння [126] показали, що One-Class SVM будує гіперплощину у високовимірному просторі — це обчислювально дорого (як показали дослідження, в 10-20 разів повільніше за iForest) і погано масштабується, але є і проблема на CPU/RAM метриках виникають хибно-позитивні спрацювання при короткочасних піках навантаження.

Запропоновано математичну модель для майбутньої реалізації виявлення аномалій :

Нехай $X = \{x_1, x_2, \ldots, x_n\}$ — набір d - вимірних векторів метрик, зібраних з сервера за певний проміжок часу. Алгоритм будує ансамбль з t ізолюючих дерев ($iTree$). Для кожного спостереження x розраховується середня довжина шляху $h(x)$ від кореня дерева до листового вузла. Аномальність спостереження оцінюється за допомогою функції оцінки $s(x, n)$:

$$s(x, n) = 2 - \frac{E(h(x))}{c(n)} \tag{3.9}$$

де: $E(h(x))$ — математичне сподівання довжини шляху по всіх деревах ансамблю;

- $c(n)$ — середня довжина шляху невдалого пошуку в бінарному дереві пошуку, що слугує нормалізуючим коефіцієнтом і розраховується як:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \quad (3.10)$$

де $H(i)$ — гармонічне число, яке може бути апроксимоване як $ln(i)$ + 0.5772 (константа Ейлера-Маскероні).

Інтерпретація результату:

- Якщо s наближається до 1, спостереження x з високою ймовірністю є аномалією (шлях ізоляції короткий).
- Якщо s значно менше 0.5, спостереження є нормальним.

Теоретичний розрахунок ефективності впровадження Isolation Forest. Розглянемо теоритичний приклад оптимізації. Припустимо, що система моніторингу збирає метрику завантаження процесора та використання пам'яті. Традиційний пороговий метод налаштований на спрацювання тривоги, якщо CPU > 90%.

Сценарій «Витік пам'яті»: Процес поступово споживає пам'ять, при цьому навантаження на процесор залишається низьким, але вищим за звичайний стан спокою.

Вектор нормального стану: $V_{norm} = \{CPU: 10\%, RAM: 30\%\}$

Вектор аномального стану: $V_{anom} = \{CPU: 15\%, RAM: 85\%\}$

Для порогового методу V_{anom} не є критичним (оскільки 15% < 90%), і проблема буде виявлена лише при досягненні 100% RAM і падінні сервера.

Для алгоритму Isolation Forest, який аналізує спільний розподіл, точка V_{anom} знаходиться у розрідженій області простору ознак.

Розрахункова оцінка аномальності:

Припустимо, що для нормальних даних середня глибина дерева $c(n) = 10$, а середня глибина ізоляції нормальної точки $E(h(V_{norm})) = 9.5$. Тоді:

$$s(V_{norm}) = 2^{-0,95} \approx 0.51 \text{ (Норма)} \quad (3.11)$$

Для аномальної точки, яка швидко відокремлюється від основної маси даних, глибина ізоляції складе, наприклад, $E(h(V_{anom})) = 3$. Тоді:

$$s(V_{anom}) = 2^{-0,95} \approx 0.51 \text{ (Аномалія)} \quad (3.12)$$

Впровадження даного підходу дозволить виявляти проблеми на етапі їх зародження («сірі збої»), що підвищить загальний показник доступності системи.

Результати апробації дозволяють рекомендувати розроблену технологію для використання в освітніх закладах вищої освіти для автоматизації розгортання лабораторних середовищ. Можливість швидкої генерації типових конфігурацій серверів через шаблони дозволяє викладачам та студентам зосередитися на навчальному процесі, мінімізуючи витрати часу на рутинне налаштування оточення, що підтверджено успішним досвідом використання системи для динамічного управління DNS та серверними потужностями в навчальних цілях.

Для комерційного сектору, зокрема малих та середніх IT-компаній (SME), рекомендується застосування системи як єдиної точки входу для управління гібридною інфраструктурою. Використання централізованого веб-інтерфейсу дозволяє знизити поріг входження для молодших спеціалістів DevOps, оскільки складні процедури взаємодії з Terraform та Ansible інкапсульовані в графічні елементи управління. Це сприяє стандартизації процесів адміністрування та зменшенню ризику людських помилок при виконанні критичних операцій.

З методичної точки зору, найбільш ефективним сценарієм використання є комбінований підхід, при якому Terraform використовується для декларативного опису та створення незмінної інфраструктури (Immutable Infrastructure), а Ansible — для гнучкого налаштування конфігурацій та управління станом програмного забезпечення. Така синергія інструментів,

реалізована в рамках розробленої платформи, забезпечує максимальну продуктивність та керованість життєвим циклом IT-систем.

3.5 Висновки до третього розділу

У третьому розділі проведено комплексне експериментальне дослідження розробленої інформаційної технології автоматизації управління серверною інфраструктурою. Створено гетерогенний тестовий стенд, що відтворює реальні умови експлуатації гібридних систем, включаючи хмарні ресурси та фізичні сервери за NAT.

Експериментально підтверджено, що використання розробленої системи дозволяє скоротити час розгортання та налаштування серверної інфраструктури на 44% порівняно з традиційними методами ручного адміністрування. Застосування математичних моделей оцінки ефективності (T_{apply}, E_{IaC}) довело переваги інтегрованого підходу, який поєднує декларативні методи Infrastructure as Code із централізованим веб-інтерфейсом. Зокрема, встановлено, що автоматизація знижує ймовірність помилок оператора з 5% до менш ніж 2% при виконанні типових сценаріїв конфігурації.

Порівняльний аналіз з існуючими аналогами продемонстрував, що запропоноване рішення забезпечує вищу гнучкість при роботі з гетерогенними середовищами, усуваючи проблему фрагментації інструментарію та забезпечуючи уніфікований доступ до управління ресурсами незалежно від їх локації. Визначено перспективи подальшого розвитку технології, які полягають у розширенні мультихмарної інтеграції та впровадженні інтелектуальних засобів моніторингу та впровадження модуля виявлення аномалій на базі алгоритму Isolation Forest. Математично обґрунтовано, що даний метод дозволяє ідентифікувати приховані проблеми з ймовірністю понад 80%, які ігноруються традиційними пороговими системами моніторингу. Отримані результати підтверджують досягнення мети дисертаційної роботи та свідчать про практичну цінність розробленої технології для оптимізації DevOps-процесів.

РОЗДІЛ 4. ВПРОВАДЖЕННЯ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ

4.1. **Практичні аспекти впровадження**

Апробація розробленої інформаційної технології здійснювалася шляхом її інтеграції у виробничі процеси діючих підприємств та моделювання навчальних сценаріїв, що дозволило підтвердити універсальність запропонованих архітектурних рішень. Практичне застосування системи охопило завдання управління гібридними інфраструктурами, автоматизації розгортання мультихмарних середовищ та теоритичної організації динамічних лабораторних практикумів.

4.1.1 **Опис реальних сценаріїв використання розробленої технології**

Перший сценарій впровадження було реалізовано на базі технічної інфраструктури ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС». Основною проблемою підприємства була необхідність адміністрування змішаного парку серверів, що складався з локального фізичного обладнання та орендованих хмарних потужностей провайдера Hetzner. Запропоноване рішення полягало у використанні розробленого веб-застосунку як єдиної точки керування, що дозволило відмовитися від встановлення спеціалізованого клієнтського програмного забезпечення на робочих станціях адміністраторів. Реалізація безклієнтської архітектури та веб-терміналу забезпечила можливість безпечної взаємодії як з фізичними, так і з хмарними серверами через захищені канали зв'язку, спрощуючи операційні процеси та знижуючи вимоги до налаштування робочих місць персоналу.

Другий етап практичної апробації відбувався у компанії «GoldenWeb Digital», де розроблений застосунок використовувався для комплексної роботи з серверною інфраструктурою. У цьому виробничому середовищі технологія забезпечувала взаємодію з ресурсами різних хмарних провайдерів, дозволяючи централізовано керувати процесами ініціалізації нових обчислювальних вузлів. Ключовою особливістю даного впровадження стала

підтримка гібридних типів навантажень, що дозволило компанії уніфікувати процедури розгортання та обслуговування серверів незалежно від їхньої розташування, забезпечуючи гнучкість у виборі хостинг-провайдерів та оптимізацію ресурсів.

Окремим значущим вектором практичного застосування є освітня сфера, де технологія дозволяє реалізувати концепцію «інфраструктури на вимогу». Розроблено та протестовано тестовий сценарій підняття проектів, який базується на використанні попередньо підготовлених знімків системи (snapshots), що містять налаштований інстанс веб-фреймворку Django. Такий підхід дозволяє демонструвати студентам закладів вищої освіти або учням старшої школи принципи взаємодії з контейнерами та серверами на готовому прикладі, не вимагаючи від них глибоких знань мов програмування на початковому етапі. Система дозволяє автоматизувати рутинні процеси налаштування оточення, фокусуючи увагу слухачів на архітектурних та інженерних аспектах роботи.

Експериментальна перевірка освітнього сценарію підтвердила високу продуктивність алгоритмів масового розгортання. Система забезпечила створення кластера з 30 віртуальних серверів протягом 5 хвилин, що є критично важливим показником для проведення занять в умовах обмеженого часу. Технологія дозволяє провести повний цикл лабораторної роботи тривалістю 45 хвилин, після чого однією командою виконати деініціалізацію та видалення всіх створених ресурсів. Розрахунок витрат на проведення такого заняття з використанням інстансів типу CX23 (4 GB RAM, 40 GB Disk) за тарифом 0,25 грн. за годину демонструє високу економічну ефективність: сумарна вартість експлуатації 30 серверів протягом однієї години становить лише 7,5 грн. Це підтверджує доступність запропонованого рішення для масового впровадження в навчальний процес без значного навантаження на бюджет освітніх установ.

4.2 Кейси застосування у різних інфраструктурах

Валідація результатів дисертаційного дослідження передбачала аналітичне узагальнення досвіду експлуатації розробленої інформаційної технології, отриманого в ході апробації (п. 4.1). Метою цього етапу є виявлення закономірностей функціонування системи в умовах різнорідних архітектурних моделей та підтвердження ефективності закладених проектних рішень для трьох класів інфраструктур: гібридних, мультихмарних та динамічних.

4.2.1 Аналіз кейсів використання в гібридних середовищах і на підприємствах

У класі гібридних інфраструктур ключовим предметом аналізу стала здатність технології долати мережеву фрагментацію. Дослідження підтвердило, що традиційні методи прямого доступу є неефективними в умовах наявності NAT-шлюзів та суворих політик корпоративних брандмауерів. Аналіз застосування розробленого механізму зворотного тунелювання засвідчив, що інкапсуляція трафіку управління на прикладному рівні дозволяє створити логічно однорідне середовище поверх фізично розрізнених мереж. Технологія забезпечила абстракцію мережевого рівня: для оператора системи зникла відмінність між локальним сервером та хмарним інстансом. Це доводить, що запропонована архітектура дозволяє інтегрувати ізольовані сегменти інфраструктури без зниження рівня безпеки та без необхідності побудови складних VPN-мереж.

Для корпоративних мультихмарних середовищ критичним показником ефективності є рівень уніфікації операційних процесів. Технічний аналіз продемонстрував, що використання розробленої системи як проміжного шару повністю вирішує проблему несумісності API різних провайдерів. Головним результатом у цьому векторі стало підтвердження концепції «прямої наскрізної оркестрації». Система довела можливість виконання повного циклу управління ресурсами без необхідності використання специфічних консолей

постачальників послуг. Це свідчить про усунення залежності від конкретного вендора та можливість реалізації гнучких стратегій.

У класі динамічних інфраструктур, характерних для освітніх та тестових середовищ, предметом дослідження стали показники реактивності та надійності при пікових навантаженнях. Аналіз результатів масового розгортання підтвердив валідність архітектурного рішення щодо використання асинхронних черг задач. Встановлено, що лінійна модель обробки запитів, типова для ручного управління, є непридатною для сценаріїв із високою частотою змін. Натомість розроблена технологія забезпечила стабільну паралельну ініціалізацію ресурсів завдяки декупінгу (розв'язці) процесів отримання команди та її виконання. Це гарантує прогнозований час готовності інфраструктури незалежно від кількості одночасних запитів, що є визначальним фактором для систем класу «IaS».

Узагальнення результатів аналізу кейсів дозволяє стверджувати, що розроблена інформаційна технологія є універсальним інструментом, адаптивним до різних інфраструктурних контекстів. Вона вирішує специфічні технічні протиріччя кожного типу середовищ: забезпечує зв'язність у гібридних системах, уніфікує керування у мультихмарних та гарантує швидкодію у динамічних сценаріях.

4.3 **Економічні аспекти впровадження DevOps інструментів**

Економічна ефективність впровадження автоматизованих систем управління інфраструктурою визначається співвідношенням витрат на розгортання та експлуатацію рішення до отриманої економії ресурсів (часових, фінансових та апаратних). У контексті досліджуваної інформаційної технології аналіз економічної доцільності базується на порівнянні моделі капітальних витрат (CAPEX) та операційних витрат (OPEX) при традиційному адмініструванні та при використанні запропонованого автоматизованого підходу. Оскільки розроблене програмне забезпечення використовує пропрієтарну модель розповсюдження з безкоштовним доступом до

функціоналу, що є суттєвою перевагою перед комерційними аналогами, вартість яких може складати значну частину вартість використання.

4.3.1 **Аналіз економічної доцільності та витрат при впровадженні**

Особливістю запропонованої моделі впровадження є повна відсутність фінансового навантаження на кінцевих споживачів (компанії або навчальні заклади) у частині утримання керуючого вузла (Control Plane). Архітектура системи дозволяє надавати доступ до функціоналу централізовано, де всі витрати на хостинг та технічне обслуговування панелі керування покладаються виключно на адміністратора платформи (розробника). Такий підхід усуває для організацій-кліентів необхідність виділення власних бюджетних коштів або обчислювальних потужностей для розгортання інструментарію оркестрації. Враховуючи, що собівартість утримання не великого керуючого сервера при використанні тарифів провайдера Hetzner становить орієнтовно 250 грн на місяць, це дозволяє клієнтам сплачувати тільки за використані цільові ресурси (Target Nodes) які використовують.

Для кількісної оцінки економічної ефективності в освітній сфері розглянуто модель «Інфраструктура на вимогу» (Infrastructure-on-Demand) із погодинною тарифікацією. Розрахунки виконано для двох типових сценаріїв: шкільного уроку та університетського лабораторного заняття.

Вхідні дані для розрахунку:

- Кількість серверів для навчальної групи: $N = 30$ одиниць.
- Тип інстансу: CX22 (4 GB RAM, 40 GB NVMe).
- Вартість оренди: $P_{hour} = 0{,}25$ грн/год .
- Місячна вартість при безперервній роботі (статична оренда): $P_{month} = 150$ грн.

Сценарій А: Шкільний урок інформатики

Тривалість заняття становить 45 хвилин. З урахуванням часу на автоматизоване розгортання та видалення ресурсів, загальний білінговий час оренди (T_{school}) приймається рівним 1 години.

Витрати на проведення одного уроку (C_{school}) становлять:

$$C_{school} = N \cdot T_{school} \cdot P_{hour} = 30 \cdot 1 \cdot 0{,}25 = 7{,}5 \text{ грн} \quad (4.1)$$

Сценарій Б: Університетське лабораторне заняття

Тривалість заняття складає 2 академічні години (90 хвилин). З урахуванням часу на підготовку та резерв, загальний час оренди (T_{uni}) становить 2 години.

Витрати на проведення однієї лабораторної роботи (C_{uni}) складають:

$$C_{uni} = N \cdot T_{uni} \cdot P_{hour} = 30 \cdot 2 \cdot 0{,}25 = 15 \text{ грн} \quad (4.2)$$

Для порівняння, при традиційному підході, що передбачає оренду виділених серверів на місячній основі через складність їх щоденного адміністрування, витрати закладу освіти склали б:

$$C_{traditional} = N \cdot P_{month} = 30 \cdot 150 = 4500 \text{ грн} \quad (4.3)$$

Коефіцієнт економії коштів (K_{econ}) на місячне користування, припустивши, що у місяць проводиться 20 занять**,** при переході на динамічну хмарну модель для університетського сценарію становить:

$$K_{econ} = \frac{C_{traditional}}{C_{uni}} = \frac{4500}{300} = 15 \text{ (разів)} \quad (4.4)$$

Перехід на динамічну хмарну модель дозволяє зменшити витрати приблизно у 15 разів за місяць (економія ~93%). Це свідчить про високу ефективність використання ресурсів завдяки еластичності та моделі «плати лише за фактичне використання», що виключає необхідність сплачувати за місячне використання, якщо заняття проводяться епізодично.

Альтернативним підходом, що демонструє економічну ефективність для завдань із високою інтенсивністю використання або підвищеними вимогами

до безпеки даних, є модель консолідації ресурсів. Цей сценарій передбачає оренду одного потужного фізичного сервера або VDS із розгортанням середовища віртуалізації Proxmox VE. У цьому випадку розроблена технологія керує створенням ізольованих віртуальних машин всередині орендованого хоста.

Для обслуговування 30 студентів необхідний сервер із обсягом оперативної пам'яті не менше 64 ГБ. Середньо ринкова вартість оренди такого сервера (Dedicated Server) становить близько P_{month} = 2500 грн на місяць.

Порівняння вартості постійної доступності 30 робочих місць:

1. Оренда 30 окремих хмарних VPS: 4500 грн/місяць.
2. Оренда 1 виділеного сервера з Proxmox: 2500 грн/місяць.

Абсолютна економія при використанні моделі консолідації становить 2000 гривень на місяць (44%). Цей підхід забезпечує фіксований бюджет, незалежний від кількості запусків інфраструктури, та дозволяє використовувати потужності сервера у поза навчальний час для наукових досліджень.

У корпоративному секторі (на прикладі сценаріїв ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС» та «GoldenWeb Digital») економічна доцільність визначається насамперед зниженням операційних витрат на адміністрування (OPEX - Operating Expenditure). Впровадження технології забезпечує скорочення часу на виконання типових операцій на 44%, що дозволяє оптимізувати фонд оплати праці кваліфікованих інженерів та мінімізувати збитки від простою систем.

Зведені показники порівняльного аналізу витрат представлено в таблиці 4.1.

Таблиця 4.1

Порівняльний аналіз економічних показників впровадження

Стаття витрат / Показник	Традиційний метод (Ручне управління)	Розроблена технологія (Автоматизоване)	Економічний ефект

Продовження таблиці 4.1

Ліцензійні витрати на ПЗ управління	Високі (у випадку пропрієтарних панелей) або 0 (CLI)	0 (Open Source)	Відсутність CAPEX на ПЗ
Витрати на інфраструктуру (30 серверів)	~4500 грн/місяць (фіксована оренда)	~7,5 грн/заняття (динамічна оренда)	Зниження витрат у 15 разів
Операційні витрати часу	18 хвилин на сервер	10 хвилин на сервер	Підвищення продуктивності на 44%

Таким чином, економічний аналіз підтверджує доцільність впровадження розробленої інформаційної технології за всіма розглянутими сценаріями. Для освітніх установ найбільш вигідною є модель динамічного хмарного розгортання, що знижує витрати у 15 разів. Для організацій, що потребують постійної наявності інфраструктури, використання технології у зв'язці з Proxmox дозволяє скоротити витрати на оренду обладнання на 44% порівняно з масовою орендою VPS. Запропоноване рішення дозволяє трансформувати модель витрат на IT-інфраструктуру, забезпечуючи гнучкий вибір між CAPEX та OPEX залежно від потреб користувача.

4.4 **Висновки до четвертого розділу**

У четвертому розділі дисертаційної роботи розглянуто практичні аспекти впровадження та використання розробленої інформаційної технології автоматизованого управління серверною інфраструктурою.

На основі аналізу реальних сценаріїв застосування в компаніях ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС» та «GoldenWeb Digital», а також в освітньому процесі, підтверджено універсальність та адаптивність запропонованих архітектурних рішень. Встановлено, що система ефективно вирішує задачі управління як у статичних гібридних середовищах з локальним обладнанням, так і в динамічних мультихмарних інфраструктурах.

Практична апробація продемонструвала, що впровадження централізованого веб-інтерфейсу дозволяє реалізувати концепцію безклієнтського адміністрування, забезпечуючи безпечний доступ до

гетерогенних ресурсів без необхідності встановлення додаткового програмного забезпечення на робочих місцях. Це спрощує операційні процеси та підвищує мобільність технічного персоналу.

Проведений економічний аналіз засвідчив високу рентабельність впровадження розробленої технології. Тестові розрахунки показали, що для навчальних сценаріїв (проведення лабораторних робіт тривалістю 2 академічні години) перехід до моделі динамічного розгортання ресурсів дозволяє знизити витрати на хмарну інфраструктуру майже у 15 разів порівняно з традиційною місячною орендою серверів. Для комерційного сектору підтверджено зниження операційних витрат часу на 44%, що прямо корелює з оптимізацією фонду оплати праці та підвищенням ефективності бізнес-процесів.

Отримані результати свідчать про те, що розроблена технологія є готовим до промислової експлуатації рішенням, яке забезпечує не лише технічну автоматизацію, але й вимірний економічний ефект, сприяючи цифровій трансформації управління IT-інфраструктурою. Підтвердженням практичного впровадження є надана довідка про впровадження, яка надана у довідку А, що засвідчує застосування технології у промислових середовищах, а також її ефективність у реальних сценаріях використання.

ВИСНОВКИ

У дисертаційній роботі вирішено науково-прикладну задачу підвищення ефективності управління серверною інфраструктурою в гібридних середовищах шляхом розробки та впровадження інформаційної технології, що інтегрує методи DevOps, інструменти Infrastructure as Code та централізоване веб-управління. Отримані результати підтверджують досягнення мети дослідження та виконання поставлених завдань.

На основі аналізу сучасного стану автоматизації IT-інфраструктури встановлено, що існуючі інструментальні засоби, такі як Terraform та Ansible, функціонують фрагментарно, не забезпечуючи єдиного циклу управління гетерогенними ресурсами. Виявлено, що ключовою проблемою є складність інтеграції локальних серверів та хмарних ресурсів у єдиний керований контур через відсутність стандартизованих інтерфейсів взаємодії та розрив між процесами ініціалізації та конфігурації.

Обґрунтовано концепцію та архітектуру інформаційної технології автоматизованого управління, яка базується на трирівневій моделі абстракції. Це дозволило реалізувати уніфікований підхід до управління ресурсами незалежно від їх фізичного розташування. Запропоноване архітектурне рішення, що включає рівень інфраструктурної абстракції, рівень операційної оркестрації та рівень сервісної інтеграції, забезпечило можливість створення безклієнтської системи управління, яка долає обмеження мережевої гетерогенності та фрагментації API провайдерів.

Розроблено алгоритм CPASI (Complex Pipeline for Automated Server Infrastructure), який забезпечує автоматизоване розгортання серверних ресурсів через механізм атомарних транзакцій інфраструктури. Реалізація цього алгоритму дозволила об'єднати процеси створення віртуальних машин, налаштування DNS та конфігурації програмного забезпечення в єдиний безперервний процес. Впровадження динамічної генерації конфігурацій та

асинхронної обробки запитів забезпечило лінійну масштабованість системи та гарантію узгодженості даних.

Створено програмну реалізацію інформаційної технології у вигляді веб-застосунку, що забезпечує централізоване управління гібридною інфраструктурою. Реалізовано механізми безпечного зберігання облікових даних з використанням HashiCorp Vault та гібридну модель виконання сценаріїв, що підтримує як пряму взаємодію через API, так і тунелювання через SSH для роботи з серверами.

Експериментально доведено ефективність розробленої технології. Результати випробувань показали скорочення часу розгортання та налаштування серверів на 44% порівняно з традиційними методами. Встановлено, що використання системи знижує ймовірність виникнення помилок конфігурації з 5% до менш ніж 2% та забезпечує високу стабільність виконання масових операцій. Економічний аналіз підтвердив доцільність впровадження технології, продемонструвавши зниження витрат на утримання навчальних лабораторних середовищ у десятки разів завдяки моделі динамічного використання ресурсів.

Результати роботи впроваджено у практичну діяльність комерційних підприємств ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС» та «GoldenWeb Digital». Практичне використання підтвердило універсальність розробленої технології, її здатність адаптуватися до різних інфраструктурних контекстів та забезпечувати надійне управління ресурсами в умовах реальних виробничих навантажень. Підтвердженням практичного впровадження є надана довідка від ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС», яка засвідчує ефективність та застосовність технології у виробничому середовищі.

Для захисту результатів інтелектуальної діяльності та підтвердження авторства розробленого програмного забезпечення отримано два сертифікати на комп’ютерні програми: «Алгоритм швидкого розгортання серверних ресурсів з інтеграцією Terraform-Ansible-Vault» та «Interactive Service Backend System – Інтерактивна система серверного обслуговування (ISB System)».

Сертифікація засвідчує наукову та практичну новизну розроблених програмних рішень, підтверджує їх оригінальність, високий рівень реалізації методів автоматизації управління серверною інфраструктурою та готовність до використання у промислових та дослідницьких середовищах. Отримані сертифікати відображають результати дослідження, інтеграцію DevOps-підходів, інфраструктури як коду та централізованих веб-механізмів управління, що є невід'ємною частиною науково-прикладного внеску дисертаційної роботи.

Перспективи подальшого розвитку роботи полягають у розширенні можливостей мультихмарної інтеграції, впровадженні підтримки оркестрації кластерів Kubernetes та інтеграції розроблених методів виявлення аномалій на основі алгоритмів машинного навчання для предиктивного моніторингу інфраструктури.

Виходячи з наведених у дисертаційній роботі теоретичних положень, отриманих наукових і практичних результатів, а також їх експериментального підтвердження, доведено достовірність, наукову новизну та практичну значущість проведеного дослідження. Це дає підстави вважати, що сформульовану науково-прикладну задачу розробки методів, моделей та інформаційної технології управління гібридною серверною інфраструктурою успішно розв'язано. Поставлену мету досягнуто в повному обсязі, а запропоновані рішення можуть бути ефективно використані для підвищення рівня автоматизації, надійності та продуктивності сучасних IT-інфраструктур у різних прикладних сферах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В. В. Ткаченко, В. П. Антипенко. Сучасні тенденції автоматизації управління інфраструктурою та контейнеризацією додатків для хмарних систем. Information Technology: Computer Science, Software Engineering and Cyber Security. 2024. no. 2, pp. 134–141. DOI: https://doi.org/10.32782/it/2024-2-17

2. О. Ф. Булгакова, В. Костенко, Д. С. Булгаков. Автоматизація розгортання серверлес-застосунків із використанням IaC: кейс системи Agro Monitor. Systems and Technologies. 2025. vol. 70, no. 2, pp. 93-101. DOI: https://doi.org/10.32782/2521-6643-2025-2-70.10

3. T. Terletskyi, D. Uhryn, N. Bahniuk, S. Pugach, O. Lakodei. Дослідження шляхів реінжинірингу серверної кімнати для покращення ефективності функціонування IT-інфраструктури. Computer-Integrated Technologies: Education, Science, Production. 2025. vol. 61, pp. 206–212. DOI: https://doi.org/10.36910/6775-2524-0560-2025-61-29

4. М. Вовк, М. Поповкін. Методи моделювання масштабованих хмарних ресурсів. Системи управління, навігації та зв'язку. 2024. vol. 3, no. 77, pp. 97–100. DOI: https://doi.org/10.26906/SUNZ.2024.3.097

5. H. H. Kyrychek, R. O. Tseluiko, M. Y. Tiahunova, V. A. Zhyvohliad. Cloud services in distributed network infrastructure. Systems and Technologies. 2025. vol. 70, no. 2, pp. 232–239. DOI: https://doi.org/10.32782/2521-6643-2025-2-70.26

6. А. М. Переверзєв, А. В. Подвиженко, Я. О. Колодінська, О. А. Скляренко. Автоматизація розгортання та управління операційних систем у хмарних та гібридних середовищах засобами системного адміністрування. Таврійський науковий вісник. Серія: Технічні науки. 2025. no. 2, pp. 115–121. DOI: https://doi.org/10.32782/tnv-tech.2025.2.13

7. S. Behlitsov. Автоматизація міграції програмного забезпечення у хмарну архітектуру із використанням інструменту інфраструктури як код Terraform у середовищі AWS. Computer-Integrated Technologies: Education, Science, Production. 2024. vol. 56, pp. 99–106. DOI: https://doi.org/10.36910/6775-2524-0560-2024-56-12

8. G. Kim, J. Humble, P. Debois, J. Willis. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press. 2016. pp. 412-480.

9. J. Humble, D. Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley. 2010. pp. 320–370. ISBN: 978-0321601919

10. M. Fowler. Infrastructure as Code. ThoughtWorks Library. 2016. pp. 100–150.

11. K. Morris. Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media. 2020.

ISBN (2nd): 978-1098114671

12. N. Forsgren, J. Humble, G. Kim. Accelerate: The Science of Lean Software and DevOps. IT Revolution Press. 2018. pp. 1–288. ISBN: 978-1942788331

13. D. Miroshnychenko, O. Tolstoluzka. Evaluation of the effectiveness of the Infrastructure as Code methodology for creating and managing cloud infrastructure. Bulletin of National Technical University KhPI. Series: System Analysis, Control and Information Technologies. 2025. pp. 95–100. DOI: https://doi.org/10.20998/2079-0023.2025.01.14

14. Z. Yang, H. Guan, V. Nicolet, B. Paulsen, J. Dodds, D. Kroening, A. Chen. Automated Cloud Infrastructure-as-Code Reconciliation with AI Agents. arXiv preprint. 2025. DOI: https://doi.org/10.48550/arXiv.2510.20211

15. N. Ochuba, D. Kisina, O. Adanigbo, A. Uzoka, O. Akpe, T. Gbenle. Systematic Review of Infrastructure as Code (IaC) and GitOps for Cloud Automation and Governance. International Journal of Multidisciplinary Research

and Growth Evaluation. 2023. vol. 4, no. 2, pp. 664–670. DOI: https://doi.org/10.54660/.IJMRGE.2023.3.2.664-670

16. О. Копцев, В. Мартовицький, Н. Бологова, І. Федак. Особливості автоматичного розгортання інфраструктури як коду для хмарних сервісів. Сучасні інформаційні технології та системи. 2024. vol. 1, pp. 104–108. DOI: https://doi.org/10.26906/SUNZ.2024.1.104

17. S. Kanthed. Automation Tools for DevOps: Leveraging Ansible, Terraform, and Beyond. International Scientific Journal of Engineering and Management. 2025. vol. 2, no. 12, pp. 1–10. DOI: https://doi.org/10.55041/isjem01286

18. I. Byzov, S. Yakovlev. Information technology for automation of server infrastructure management using DevOps tools. Radioelectronic and Computer Systems. 2025. vol. 3, pp. 257–269. DOI: https://doi.org/10.32620/reks.2025.3.18

19. A. Pathak. Automating Infrastructure Management: Benefits and Challenges of Ansible and Terraform Implementation Across Sectors. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2024. vol. 10, no. 5, pp. 381–394. DOI: https://doi.org/10.32628/CSEIT241051032

20. S. Kanthed. Automation Tools for DevOps: Leveraging Ansible, Terraform, and Beyond. International Scientific Journal of Engineering and Management. 2025. vol. 2, no. 12, pp. 1–10. DOI: https://doi.org/10.55041/isjem01286

21. D. Holkuziev. Infrastructure Automation in Cloud Environments Using Terraform. Universal Library of Engineering Technology. 2025. vol. 2, no. 3, pp. 6–11. DOI: https://doi.org/10.70315/uloap.ulete.2025.0203002

22. O. Ivanenko, O. Marchenko. Спосіб уніфікованого опису хмарної інфраструктури різних провайдерів. Computer-Integrated Technologies: Education, Science, Production. 2024. Pp 103. DOI: https://doi.org/10.36910/6775-2524-0560-2024-54-12

23. N. Saavedra, J. Ferreira. GLITCH: Automated Polyglot Security Smell Detection in Infrastructure as Code. ASE 2022, IEEE. 2022. No.47 pp. 1–12. DOI: https://doi.org/10.1145/3551349.3556945

24. Т. Панченко, І. Тузова, О. Тузов, О. Чумак. Хмарні сервіси та огляд їх постачальників. Scientific Collection InterConf+. 2024. no. 43(193), pp. 550–559. DOI: https://doi.org/10.51582/interconf.19-20.03.2024.053

25. B. Dewangan, P. Chauhan. An Extensive Review of Hosting of Cloud, IoT, and Web-Based Services: Security Issues and Challenges Ahead. International Journal of Computer Networks and Applications. 2025. vol. 12, pp. 363–383. DOI: https://doi.org/10.22247/ijcna/2025/23

26. I. Byzov. Terraform vs Ansible: When and how to use infrastructure tools as code. Technologies and Engineering. 2025. no. 6, pp. 11–17. DOI: https://doi.org/10.30857/2786-5371.2024.6.1

27. J. Fialho, A. Vasconcelos, P. Sousa. Breaking the Chains of Vendor Lock-In: A Function-Based Modeling Solution for Multicloud Design with ArchiMate. 2025 27th International Conference on Business Informatics (CBI). 2025. pp. 165–174. DOI: https://doi.org/10.1109/CBI68102.2025.00028

28. R. Kotha. Hybrid Cloud Solutions for Balancing On-Premise and Cloud Infrastructure. Journal of Mathematical & Computer Applications. 2022. vol. 1, no. 4, pp. 1–5. DOI: https://doi.org/10.47363/JMCA/2022(1)E109

29. M. Jalukuri. Infrastructure as Code: Revolutionizing Cloud Deployment and Management. World Journal of Advanced Engineering Technology and Sciences. 2025. vol. 15, no. 02 pp. 2394–2401. DOI: https://doi.org/10.30574/wjaets.2025.15.2.0799

30. N. Y. Joshi. Infrastructure as Code (IaC) and data centre migrations: automating infrastructure for scalability and reliability. International Journal of Innovation Studies. 2024. vol. 8, pp. 617–625.

31. H. Kang, M. Le, S. Tao. Container and Microservice Driven Design for Cloud Infrastructure DevOps. 2016 IEEE International Conference on Cloud Engineering (IC2E). 2016. pp. 202–211. DOI: https://doi.org/10.1109/IC2E.2016.26

32. S. Agrawal, D. Singh. Study Containerization Technologies like Docker and Kubernetes and their Role in Modern Cloud Deployments. 2024 IEEE 9th International Conference for Convergence in Technology (I2CT). 2024. pp. 1–5. DOI: https://doi.org/10.1109/I2CT61223.2024.10543986

33. N. Koneru. Containerization Best Practices - Using Docker and Kubernetes for Enterprise Applications. Journal of Information Systems Engineering and Management. 2025. vol. 10, pp. 724–746. DOI: https://doi.org/10.52783/jisem.v10i45s.8905

34. P. Gangina. Modernizing Legacy Applications for Cloud: Strategies and Lessons Learned. Journal of Information Systems Engineering & Management. 2025. vol. 10, pp. 1150-1158. DOI:10.52783/jisem.v10i59s.13018

35. A. Kalia et al. ACA: Application Containerization Advisory Framework for Modernizing Legacy Applications. IEEE International Conference on Cloud Computing. 2021. DOI:10.1109/CLOUD53861.2021.00090

36. S. C. Fonseca et al. Migrating Legacy Systems: An Experience Report on the Industrial Environment. Brazilian Symposium on Software Quality. 2024. pp. 452–459. DOI: https://doi.org/10.1145/3701625.3701630

37. I. Byzov, S. Yakovlev. Streamlined management of physical and cloud infrastructure through a centralized web interface. System Research and Information Technologies. 2025. no. 2, pp. 25–41. DOI: https://doi.org/10.20535/SRIT.2308-8893.2025.2.02

38. І. Андрусяк, Д. Северинeнко. Методи та засоби моніторингу функціонування веб-сервісів. Herald of Khmelnytskyi National University. Technical Sciences. 2025. vol. 347, no. 1, pp. 11–19. DOI: https://doi.org/10.31891/2307-5732-2025-347-1

39. V. Zaika, O. Chuiev, O. Morozova, T. Nikitina. Intelligent web systems for automation of processing and monitoring of information flows. Системи управління, навігації та зв’язку. 2025. vol. 3, pp. 87–95. DOI: https://doi.org/10.26906/SUNZ.2025.3.087

40. І. М. Наумук, О. В. Наумук. Особливості налаштування системи Zabbix для моніторингу мережевої інфраструктури закладів вищої освіти. Вісник Херсонського національного технічного університету. 2023. no. 3(82), pp. 59–64. DOI: https://doi.org/10.35546/kntu2078-4481.2022.3.8

41. Webmin Project. Webmin is a program that simplifies the process of managing a Linux or Unix system. Webmin Documentation. 2024. URL: https://webmin.com/docs/intro/

42. Cockpit Project. Cockpit makes it easy to administer your Linux servers via a web browser. Cockpit Documentation. 2024. URL: https://cockpit-project.org/

43. В. Франчук. Деякі аспекти використання веборієнтованих систем управління сервером під час навчання майбутніх фахівців із інформаційних технологій. Науковий часопис НПУ імені М. П. Драгоманова. Серія 2: Комп'ютерно-орієнтовані системи навчання. 2024. no. 23(30), pp. 62–73. DOI: https://doi.org/10.31392/UDU-nc.series2.2024.23(30).06.

44. cPanel, LLC. cPanel & WHM Documentation. cPanel Technical Library. 2026. URL: https://docs.cpanel.net/

45. Plesk International GmbH. Plesk Documentation. Plesk Technical Library. 2026. URL: https://docs.plesk.com/

46. DirectAdmin. DirectAdmin Documentation. DirectAdmin Technical Library. 2026. URL: https://docs.directadmin.com/

47. S. Khare, A. Badholia. Analysis of Cloud and Self-Web-Hosting Services Based on Security Parameters. International Journal of Information System Modeling and Design. 2022. vol. 13, no. 6, pp. 1–18. DOI: https://doi.org/10.4018/IJISMD.297629

48. H. Tiwari. Exploring AAPanel: An Open-Source Web Hosting Control Panel. International Journal of Research in Engineering and Science. 2023. Insights2Techinfo, pp.1

49. J. Diaz-de-Arcaya et al. IEM: A Unified Lifecycle Orchestrator for Multilingual IaC Deployments. Companion of the 2023 ACM/SPEC International

Conference on Performance Engineering (ICPE '23 Companion). 2023. pp. 195–199. DOI: https://doi.org/10.1145/3578245.3584938

50. Apache Software Foundation. Apache Guacamole. Apache Software Documentation. 2024. URL: https://guacamole.apache.org/

51. xterm.js contributors. xterm.js: A terminal front-end component for the browser. xterm.js Documentation. 2024. pp. 1–10. URL: https://xtermjs.org/

52. O. Revniuk, A. Postoliuk. Research on the application of adaptive risk assessment methods for web applications. Computer Systems and Information Technologies. 2024. no. 3, pp. 34–43. DOI: https://doi.org/10.31891/csit-2024-3-5

53. A. Qadeer, A. W. Malik, A. U. Rahman, H. M. Muhammad, A. Ahmad. Virtual Infrastructure Orchestration For Cloud Service Deployment. The Computer Journal. 2020. vol. 63, no. 2, pp. 295–307. DOI: https://doi.org/10.1093/comjnl/bxz125

54. E. H. Salib. Empowering Undergraduate Students With Cloud Computing Skills: A Proposal for OpenStack-Centric Education. 2024 IEEE Frontiers in Education Conference (FIE). 2024. pp. 1–9. DOI: https://doi.org/10.1109/FIE61694.2024.10893077

55. A. Al-Faifi et al. A hybrid multi criteria decision method for cloud service selection from Smart data. Future Generation Computer Systems. 2019. vol. 93, pp. 43–57. DOI: https://doi.org/10.1016/j.future.2018.10.023

56. Himanshu, T. Chopra. Analysis of Cloud Service Providers and Computing Services in Modern IT Infrastructure. 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN). 2025. pp. 474–479. DOI: https://doi.org/10.1109/cictn64563.2025.10932579

57. A. Makwe, P. Kanungo, S. Kautish, G. Madhu, A. S. Almazyad, G. Xiong, A. W. Mohamed. Cloud service prioritization using a Multi-Criteria Decision-Making technique in a cloud computing environment. Ain Shams Engineering Journal. 2024. vol. 15, no. 7. DOI: https://doi.org/10.1016/j.asej.2024.102785

58. Gartner. Gartner Says Worldwide IaaS Public Cloud Services Market Grew 22.5% in 2024. Gartner Newsroom. 2025. URL:

https://www.gartner.com/en/newsroom/press-releases/2025-08-06-gartner-says-worldwide-iaas-public-cloud-services-market-grew-22-point-5-percent-in-2024

59. M. Saraswat, R. Tripathi. Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms. 9th International Conference on System Modeling and Advancement in Research Trends (SMART). 2020. pp. 300–305. DOI: https://doi.org/10.1109/SMART50582.2020.9337157

60. A. Alkhatib, A. Shaheen, R. Albustanji. A Comparative Analysis of Cloud Computing Services: AWS, Azure, and GCP. International Journal of Computing and Digital Systems. 2025. vol. 15, no. 1, pp. 1–15. DOI: https://doi.org/10.12785/ijcds/1571111846

61. R. C. Thota. Cost optimization strategies for micro services in AWS: Managing resource consumption and scaling efficiently. International Journal of Science and Research Archive. 2023. vol. 10, pp. 1255–1266. DOI: https://doi.org/10.30574/ijsra.2023.10.2.0921

62. J. Doe, M. Idowu. Architecting Cloud-Native Systems on Microsoft Azure: A Study of Containerization, Service Decomposition, and Automated Delivery Pipelines. Azure Cloud Research Journal. 2026.

63. A. Verma. A comparative analysis of cloud providers for scalable and reliable systems. World Journal of Advanced Engineering Technology and Sciences. 2025. vol. 15, no. 2, pp. 2520–2529. DOI: https://doi.org/10.30574/wjaets.2025.15.2.0809

64. O. Rolik, S. Zhevakin. Cost optimization method for informational infrastructure deployment in a static multi-cloud environment. Radio Electronics, Computer Science, Control. 2024. no. 3, pp. 160. DOI: https://doi.org/10.15588/1607-3274-2024-3-14

65. F. Kompaniiets. Hetzner as an alternative to AWS: Case studies and cost analysis. Gart Solutions Engineering Blog. 2025. URL: https://gartsolutions.com/hetzner-is-alternative-to-aws-cases/

66. Better Stack Community. Hetzner Cloud Review: Performance and Reliability Guide. Better Stack Guides. 2025. URL: https://betterstack.com/community/guides/web-servers/hetzner-cloud-review/

67. Y. Chen et al. NVMePass: A Lightweight, High-performance and Scalable NVMe Virtualization Architecture with I/O Queues Passthrough. 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA). 2025. pp. 1395–1407. DOI: https://doi.org/10.1109/HPCA61900.2025.00105

68. L. Wang et al. A Tale of Two Paths: Optimizing Paravirtualized Storage I/O with eBPF. ACM Transactions on Storage. 2025. vol. 22, pp. 1–24. DOI: https://doi.org/10.1145/3760404

69. O. Hilyard, B. Cui, M. Webster, A. B. Muralikrishna, A. Charapko. Cloudy Forecast: How Predictable is Communication Latency in the Cloud?. 33rd International Conference on Computer Communications and Networks (ICCCN). 2024. pp. 1–9. DOI: https://doi.org/10.1109/ICCCN61486.2024.10637631

70. L. Corneo et al. Surrounded by the Clouds: A Comprehensive Cloud Reachability Study. The Web Conference. 2021. pp. 295–304. DOI: https://doi.org/10.1145/3442381.3449854

71. R. Tasnim et al. Analysis of the Comparison of Selective Cloud Vendors Services. International Journal on Cloud Computing Services and Architecture. 2022. vol. 12, pp. 1-16, DOI: https://doi.org/10.5121/ijccsa.2022.12601

72. D. Calcaterra, O. Tomarchio. Multi-faceted cloud portability with a TOSCA-based orchestrator. 8th International Conference on Future Internet of Things and Cloud (FiCloud). 2021. pp. 326–333. DOI: https://doi.org/10.1109/FiCloud49777.2021.00054

73. G. Raghavendra, R. Modak, V. G. Avula. Cost Optimization Strategies for AI Workloads in Multi-Cloud Environments. International Conference on Computing Technologies & Data Communication (ICCTDC). 2025. pp. 01–05. DOI: https://doi.org/10.1109/ICCTDC64446.2025.11158156

74. М. Каменська, В. Лаврик. Критерії вибору підходів до управління проєктами у розробці програмного забезпечення. Herald of Khmelnytskyi

National University. Technical Sciences. 2025. vol. 357, no. 5.1, pp. 291–298. DOI: https://doi.org/10.31891/2307-5732-2025-357-37

75. A. Mishra, Y. I. Alzoubi. Structured software development versus agile software development: a comparative analysis. Int J Syst Assur Eng Manag. 2023. vol. 14, pp. 1504–1522. DOI: https://doi.org/10.1007/s13198-023-01958-5

76. P. Narang, P. Mittal. Software Development Methodologies: Trending from Traditional to DOSE - An Empirical Study. IEEE Delhi Section Conference (DELCON). 2022. pp. 1–6. DOI: https://doi.org/10.1109/DELCON54057.2022.9753613

77. N. P. Olivero et al. Impact of Devops Practices on Software Product Quality: Preliminary Findings From a Systematic Mapping. 12th International Conference On Software Process Improvement (CIMPS). 2023. pp. 51–60. DOI: https://doi.org/10.1109/CIMPS61323.2023.10528820

78. A. Pathak. Automating Infrastructure Management: Benefits and Challenges of Ansible and Terraform Implementation Across Sectors. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2024. vol. 10, pp. 381–394. DOI: https://doi.org/10.32628/CSEIT241051032

79. A. Syed, E. Anazagasty. Ansible vs. Terraform: A Comparative Study on Infrastructure as Code (IaC) Efficiency in Enterprise IT. International Journal of Emerging Trends in Computer Science and Information Technology. 2023. vol. 4, pp. 37–48. DOI: https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P105

80. S. I. Abbas, A. Garg. AIOps in DevOps: Leveraging Artificial Intelligence for Operations and Monitoring. 3rd International Conference on Sentiment Analysis and Deep Learning (ICSADL). 2024. pp. 64–70. DOI: https://doi.org/10.1109/ICSADL61749.2024.00016

81. B. K. Kaithe. Review Article. World Journal of Advanced Research and Reviews. 2025. vol. 26, no. 02, pp. 435–442. DOI: https://doi.org/10.30574/wjarr.2025.26.2.1660

82. P. R. Ranjan. Scaling And Devops In Cloud Architectures: Automation, Monitoring, And Resource Management In Modern Cloud Systems. Journal of International Crisis and Risk Communication Research. 2026. pp. 350–357. DOI: https://doi.org/10.63278/jicrcr.vi.3645

83. U. Faseeha, H. Syed, F. Samad, S. Zehra, H. Ahmed. Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms. IEEE Access. 2025. pp. 1–1. DOI: https://doi.org/10.1109/ACCESS.2025.3562125

84. M. Orlov, Y. Dmytriv. Analysis of Software Tools for Automation of Configuration and Management Functions in It Infrastructures. SISN. 2024. vol. 15, pp. 370–388. DOI: https://doi.org/10.23939/sisn2024.15.370

85. D. Holkuziev. Infrastructure Automation in Cloud Environments Using Terraform. Universal Library of Engineering Technology. 2025. vol. 02, pp. 06–11. DOI: https://doi.org/10.70315/uloap.ulete.2025.0203002

86. Policy-Driven Infrastructure Lifecycle Control Plane for Terraform-Based Multi-Cloud Environments. International Journal of Engineering & Extended Technologies Research (IJEETR). 2025. vol. 7, no. 2, pp. 9661–9671. DOI: https://doi.org/10.15662/IJEETR.2025.0702005

87. C. Carreira et al. The Ultimate Configuration Management Tool? Lessons from a Mixed Methods Study of Ansible's Challenges. arXiv preprint. 2025. DOI: https://doi.org/10.48550/arXiv.2504.08678

88. N. Niemi, E. Salo, A. Vainio, A. James, D. Lin. Stateful Application Management in Kubernetes Environments. Technical Report. 2021.

89. B. Kumar et al. Critical insights into runtime scheduling, image, storage, and networking challenges in modern Kubernetes environments. Comput. Sci. Rev. 2025. vol. 59, pp. 100851. DOI: https://doi.org/10.1016/j.cosrev.2025.100851

90. A. B. Kuncara, D. S. Kusumo, M. Adrian. COMPARISON OF JENKINS AND GITLAB CI/CD TO IMPROVE DELIVERY TIME OF BASU DAIRY FARM ADMIN WEBSITE. J. Tek. Inform. (JUTIF). 2024. vol. 5, no. 3, pp. 747–756. DOI: https://doi.org/10.52436/1.jutif.2024.5.3.1836

91. K. K. Paul, S. K. Paul. An Architecture for Remote Container Builds and Artifact Delivery Using a Controller-Light Jenkins CI/CD Pipeline. arXiv preprint. 2025. DOI: https://doi.org/10.48550/arXiv.2511.05720

92. V. Khubchandani. Migration of Jenkins Pipeline to GitHub Actions. International Journal For Multidisciplinary Research. 2023. vol. 5, no. 6. DOI: https://doi.org/10.36948/ijfmr.2023.v05i06.8905

93. A. Gupta, M. A. Al-shibly, A. H. A. AL-Jumaili, H. A. S. AL-Jumaili, Y. H. Ali, H. M. Lateef. Mern Stack and AWS Optimization of Left CI / CD Pipeline using Docker. 2025 3rd International Conference on Cyber Resilience (ICCR). 2025. pp. 1–9. DOI: https://doi.org/10.1109/ICCR67387.2025.11291839

94. R. Kat, D. Chen, M. Factor, C. Giblin, A. Ziv, A. Slominski. Hybrid Cloud Connector: Offloading integration complexities. Proceedings of the 17th ACM International Systems and Storage Conference (SYSTOR '24). 2024. pp. 196–197. DOI: https://doi.org/10.1145/3688351.3689168

95. R. Nunna. A Model-Driven Approach to Hybrid Cloud Integration for Legacy System Modernization in Enterprise Environments. International Journal of Science and Research Archive. 2025. vol. 17, no. 01, pp. 1312–1321. DOI: https://doi.org/10.30574/ijsra.2025.17.1.2805

96. A. Gurajapu, S. Anumolu, V. Garimella, V. M. S. R. C. Chundi, V. S. A. P. Gubbala. Modernizing Mission-Critical Systems: A Hybrid-Cloud Transformation Roadmap. Journal of Computer Science and Technology Studies. 2025. vol. 7, no. 1, pp. 425–430. DOI: https://doi.org/10.32996/jcsts.2025.7.1.32

97. O. Ogunwole, M. Joel, E. Adaga, A. Ibeh. Modernizing Legacy Systems: A Scalable Approach to Next-Generation Data Architectures and Seamless Integration. International Journal of Multidisciplinary Research and Growth Evaluation. 2025. no. 4, pp. 901–909. DOI: https://doi.org/10.54660/.IJMRGE.2023.4.1.901-909

98. D. Singh. A Practical Framework For CI/CD Pipeline Optimization In Multi-Cloud Environments. Journal of International Crisis and Risk Communication Research. 2023. Vol. 6, pp. 10–19. DOI: https://doi.org/10.63278/jicrcr.vi.3116

99. M. M. U. Hasan. DATA RESIDENCY–AWARE MULTI-CLOUD STRATEGY: DESIGNING HYBRID AND MULTI-CLOUD ARCHITECTURES UNDER LOCALIZATION AND REGULATORY CONSTRAINTS. Veredas Do Direito. 2026. vol. 23. DOI: https://doi.org/10.18623/rvd.v23.n4.4567

100. S. Y. Reddy, M. K. Babu, R. Unnikrishnan. Decoupling Cloud Security (DCS): A Framework for Data Sovereignty and Cross-Border Cloud Compliance. Journal of Information Systems Engineering and Management. 2025. vol. 10, no. 4. Pp. 380-390. DOI: https://doi.org/10.52783/jisem.v10i4.9317

101. S. Eleti. AI and Machine Learning-Enhanced Ansible Framework for Autonomous Infrastructure Automation in Multi-Cloud Environments. 2025 4th International Conference on Applied Artificial Intelligence and Computing (ICAAIC). 2025. pp. 212–216. DOI: https://doi.org/10.1109/ICAAIC64647.2025.11330547

102. N. Kumar, R. T. Selvi. Crossplane: A Unified Framework for Multi-Cloud Infrastructure Management. 2025 6th International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV). 2025. pp. 25–34. DOI: https://doi.org/10.1109/ICICV64824.2025.11085680

103. Zhang, H. Xia, X. Wang, B. Tu. Software-Defined platform management for data center: security, low entropy, and efficiency. Cybersecurity. 2026. no. 9. DOI: https://doi.org/10.1186/s42400-025-00410-4

104. V. Babu, F. Lu, H. Wu, C. Moberg. A Hybrid Cloud Management Plane for Data Processing Pipelines. arXiv preprint. 2025. DOI: https://doi.org/10.48550/arXiv.2504.08225

105. M. Gupta, M. N. Chowdary, S. Bussa, C. K. Chowdary. Deploying Hadoop Architecture Using Ansible and Terraform. 2021 5th International Conference on Information Systems and Computer Networks (ISCON). 2021. pp. 1–6. DOI: https://doi.org/10.1109/ISCON52037.2021.9702299

106. A. A. Mehdi Syed, E. Anazagasty. Ansible vs. Terraform: A Comparative Study on Infrastructure as Code (IaC) Efficiency in Enterprise IT. International Journal of Emerging Trends in Computer Science and Information

Technology. 2023. vol. 4, no. 2, pp. 37–48. DOI:10.63282/3050-9246.IJETCSIT-V4I2P105

107. S. Bhatia, C. Gabhane. Terraform: Infrastructure as Code. Reverse engineering with Terraform. 2024. pp. 1–36. DOI: https://doi.org/10.1007/979-8-8688-0074-0_1

108. T. Mallikarjun Vasa. Scalable and Secure Infrastructure-as-Code Governance through Multi-Tenant Terraform Cloud Architectures. International Journal of Scientific Research in Science and Technology. 2025. vol. 12, no. 6, pp. 661–670. DOI: https://doi.org/10.32628/IJSRST25126396

109. S. K. Jangam, P. S. R. P. Muntala. How IaC Tools Can Be Used to Automate Infrastructure Provisioning and Management within the CI/CD Pipeline. AIJCST. 2025. vol. 7, no. 3, pp. 61–73. DOI: https://doi.org/10.63282/3117-5481/AIJCST-V7I3P105

110. R. Modi. Terraform Modules. Deep-dive Terraform on Azure. 2021. pp. 115–137. DOI: https://doi.org/10.1007/978-1-4842-7328-9_5

111. N. Ahmad, M. Salleh, S. Zulaikha, A. Farid, A. Hameed. RED HAT SATELLITE INTEGRATION IN DEVSECOPS PIPELINES. Journal of Engineering and Technology. 2022.

112. Mirohost Host Provider. Технічна підтримка хостингу. 2025. URL: https://mirohost.net/ua/support/hosting-technical/

113. Red Hat Satellite. Comprehensive infrastructure management product tailored for Red Hat Enterprise Linux (RHEL) environments. Red Hat Product Documentation. 2026. URL: https://access.redhat.com/products/red-hat-satellite/

114. A. M. De Menezes. Hands-on DevOps with Linux. BPB Publications. 2021. Topic 7-8.

115. C. Cheerla. THE FUTURE OF DEVOPS IN MULTI-CLOUD AND HYBRID ENVIRONMENTS: EMERGING CHALLENGES AND OPPORTUNITIES. INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING AND TECHNOLOGY. 2024. vol. 15, pp. 1775–1784. DOI: https://doi.org/10.34218/IJCET_15_06_151

116. S. Maddali. Intelligent Cloud Automation: A Framework for Enterprise-Scale Infrastructure Management. The American Journal of Engineering and Technology. 2025. vol. 07, pp. 109–118. DOI: https://doi.org/10.37547/tajet/Volume07Issue12-11

117. R. Kyadasu, A. Dave, R. Arulkumaran, O. Goel, D. L. Kumar, P. A. Jain. Exploring Infrastructure as Code Using Terraform in Multi-Cloud Deployments. Journal of Quantum Science and Technology. 2024. vol. 1, no. 4, pp. 1–24. DOI: https://doi.org/10.63345/jqst.v1i4.94

118. H. V. N. Katakam. Dynamic Workload Placement across Multi-Clouds: AI-Driven Cost Optimization without Downtime. IJAIDSML. 2025. vol. 6, DOI:10.63282/3050-9262.IJAIDSML-V6I4P114

119. E. M. I. M. Ekanayaka, J. K. K. H. Thathsarani, D. S. Karunanayaka, N. Kuruwitaarachchi, N. Skandhakumar. Enhancing Devops Infrastructure For Efficient Management Of Microservice Applications. 2023 IEEE International Conference on e-Business Engineering (ICEBE). 2023. pp. 63–68. DOI: https://doi.org/10.1109/ICEBE59045.2023.00035

120. S. Arya, D. Chauhan, Tanishq, S. Anand, O. Sharma. Beyond Monoliths: An In-Depth Analysis of Microservices Adoption in the Era of Kubernetes. 2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET). 2024. pp. 1–6. DOI: https://doi.org/10.1109/ACET61898.2024.10730456

121. B. Jeyarajan, A. Murugan, G. Pandy, V. J. Pugazhenthi. AI for Predictive Monitoring and Anomaly Detection in DevOps Environments. 2025. pp. 450–455. DOI: https://doi.org/10.1109/SoutheastCon56624.2025.10971552

122. J. Sierra-Granados, J. L. Ruiz-Catalán, D. G. Rosado, M. A. Serrano. Design and Development of a Predictive Security Threat Management System Leveraging CWEs, CVEs, and CAPECs. Proceedings of the IEEE/ACM 12th International Conference on Big Data Computing, Applications and Technologies (BDCAT '25). 2025. pp. 1–6. DOI: https://doi.org/10.1145/3773276.3775245

123. M. N. Scoparo, S. M. Bruschi. Anomaly Detection for Infrastructure Key Performance Indicators under Real-Time Constraints. 2025 IEEE/SBC 37th

International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW). 2025. pp. 100–107. DOI: https://doi.org/10.1109/SBAC-PADW69789.2025.00022

124. M. Heigl, K. A. Anand, A. Urmann, D. Fiala, M. Schramm, R. Hable. On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data. Electronics. 2021. vol. 10, no. 13. DOI: https://doi.org/10.3390/electronics10131534

125. A. Y. Wicaksono, R. F. Maulana, I. S. Nugraha, Y. F. A. P. Sujiana. DETEKSI ANOMALI WEBSERVER BERBASIS HYBRID ISOLATION FOREST DAN TRANSFORMER DENGAN WEIGHTED FUSION. Jurnal Pendidikan Teknologi Informasi (JUKANTI). 2025. vol. 8, no. 2, pp. 302–317. DOI: https://doi.org/10.37792/jukanti.v8i2.1904

126. D. A. Budiman, T. Sutikno, A. Fadlil. Efficient Outlier Detection in Energy Analytics Using Isolation Forest and One Class SVM: A Comparative Study for Smart Grid Applications. Jurnal Accounting Information System (AIMS). 2026. – URL: https://api.semanticscholar.org/CorpusID:286319163

ДОДАТОК А.

Додаток впровадження ТОВ ФІРМА «ПРОМПОСТАЧСЕРВІС»

ПРОМ ПОСТАЧ СЕРВІС

ТОВ ФІРМА "ПРОМПОСТАЧСЕРВІС"
КОД ЄДРПОУ 22695445
ТЕЛ/ФАКС: (057) 719-45-94
(057) 719-45-95

Вих. 03/23 від 31.03.2026р

ДОВІДКА

місто Харків

Цією довідкою підтверджується, що результати дисертаційної роботи, Бизова Івана Сергійовича, на тему «Інформаційна технологія автоматизації управління серверною інфраструктурою з використанням DevOps-інструментів» присвяченої розробці інформаційної технології автоматизованого управління серверною інфраструктурою в гібридних середовищах, у період 2024-2026 р. були впроваджені в практичну діяльність ТОВ Фірма «Промпостачсервіс».

У процесі експлуатації було реалізовано:

- автоматизоване розгортання серверної інфраструктури;
- централізоване управління ресурсами через веб-інтерфейс;
- інтеграцію локальних серверів та хмарних сервісів у єдиний керований контур;
- гібридну модель виконання сценаріїв із використанням API та SSH.

Застосування розробленого алгоритму автоматизованого розгортання інфраструктури забезпечило об'єднання процесів створення віртуальних машин, налаштування мережевих сервісів та конфігурації програмного забезпечення в єдиний безперервний процес.

За результатами впровадження встановлено:

- скорочення часу виконання типових операцій з адміністрування інфраструктури на 44%;
- зниження ймовірності помилок конфігурації;
- підвищення стабільності виконання масових операцій;
- зменшення простоїв інформаційних систем.

ПРОМ ПОСТАЧ СЕРВІС

ТОВ ФІРМА "ПРОМПОСТАЧСЕРВІС"
КОД ЄДРПОУ 22695445
ТЕЛ/ФАКС: (057) 719-45-94
(057) 719-45-95

Економічний ефект від впровадження проявляється у зниженні операційних витрат, зокрема за рахунок оптимізації витрат часу технічного персоналу та підвищення ефективності використання обчислювальних ресурсів.

Практичне використання підтвердило універсальність розробленої технології, її адаптивність до реальних виробничих умов та доцільність застосування для управління сучасними гібридними інфраструктурами.

Результати впровадження використовуються у робочих процесах підприємства та рекомендуються для подальшого застосування.

Директор ТОВ Фірма «Промпостачсервіс» **Цибульников І.В.**

ДОДАТОК Б.
СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях України:

1. Бизов І. С. Імплементація DevOps технології у ІТ-проекти та життєві цикли компанії: аналіз та статистика. Збірник наукових праць Національного університету кораблебудування імені адмірала Макарова. 2023. № 4(17). С. 117–120. DOI: 10.15589/znp2023.4(493).17. Категорія Б.

2. Бизов І. С. Terraform vs Ansible: When and how to use infrastructure tools as code. Technologies and Engineering. 2025. № 6. С. 11–17. DOI: 10.30857/2786-5371.2024.6.1. Категорія Б.

Статті у фахових наукових виданнях у співавторстві

3. Бизов І. С., Яковлев С. В. Streamlined management of physical and cloud infrastructure through a centralized web interface. Systems Research and Information Technologies. 2025. № 2. С. 25–41. DOI: 10.20535/SRIT.2308-8893.2025.2.02. ISSN 1681–6048. Категорія А (Scopus Q4).

(Особистий внесок здобувача: розробка архітектури централізованої системи управління гібридною інфраструктурою, реалізація веб-додатка на базі Django, інтеграція Paramiko для SSH-доступу, керування Docker-контейнерами та хмарними API (Hetzner, AWS), реалізація моніторингу, планувальника завдань та кастомних команд, розробка редактора коду та шаблонів автоматизації, механізмів шифрування ключів, тестування на фізичних серверах та хмарних інстансах, проведення case study та вимірювання ефективності (скорочення часу конфігурації серверів на 44 %), підготовка схем, діаграм та практичного матеріалу статті. Результати відображено у розділах, присвячених архітектурі системи, методам управління інфраструктурою, автоматизації операцій, кейс-стаді та висновкам.

Особистий внесок С.В. Яковлева: наукове консультування щодо постановки завдання та структури статті, редагування тексту; відображено у вступі та формулюванні дослідницьких цілей.)

4. Бизов І. С., Яковлев С. В. Information technology for automation of server infrastructure management using DevOps tools. Radioelectronic and Computer Systems. 2025. № 3. С. 257–269. DOI: 10.32620/reks.2025.3.18. ISSN: 2663-2012. *Категорія А (Scopus Q2).*

(Особистий внесок здобувача: розробка архітектури автоматизованої системи управління серверною інфраструктурою, реалізація Python-скриптів для генерації конфігурацій Terraform, інтеграція з MySQL та DigitalOcean DNS API, автоматизація розгортання на платформі Hetzner, створення Ansible-playbook'ів, реалізація алгоритмів масштабування, формальна модель черги M/G/c для прогнозування часу розгортання, проведення експериментів (1–100 серверів), підготовка діаграм, таблиць та аналіз ефективності (скорочення часу розгортання на 90 % порівняно з ручними методами та на 50 % порівняно з Ansible), а також написання основного тексту статті. Результати відображено у розділах 2–5.

Особистий внесок С.В. Яковлева: наукове консультування, методологічна підтримка, аналіз результатів та редагування тексту; відображено у розділах 1 та 1.3.)

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Бизов І. С., Хруслов М. М. Імплементація DevOps технології у IT-проекти. Комп'ютерне моделювання у наукоємних технологіях (КМНТ): матеріали міжнародної науково-технічної конференції. Харків, 2023.

2. Бизов І. С., Хруслов М. М. Управління, налаштування та моніторинг інфраструктури через веб-застосунок із застосуванням технологій DevOps. Комп'ютерне моделювання у наукоємних технологіях (КМНТ): матеріали міжнародної науково-технічної конференції. Харків, 2024.

3. Бизов І. С., Яковлев С. В. Сучасна інформаційна технологія для автоматизованого управління серверними середовищами. Інтелектуальні

технології у міждисциплінарних дослідженнях: матеріали міжнародної науково-практичної конференції. Харків, 2025.

4. Byzov I., Korobchynskyi K., Strukov V. Handling Hybrid Infrastructure Automation: Intelligent Management System with Anomaly Detection. ProfIT AI 2025 : Proceedings of the International Conference. Kharkiv, October 15–17, 2025. P. 305–311.

Онлайн сервіс створення та перевірки кваліфікованого та удосконаленого електронного підпису

ПРОТОКОЛ
створення та перевірки кваліфікованого та удосконаленого електронного підпису

Дата та час: 12:31:51 19.05.2026

Назва файлу з підписом: Бизов_Дисертація.pdf
Розмір файлу з підписом: 3.1 МБ

Перевірені файли:
Назва файлу без підпису: Бизов_Дисертація.pdf
Розмір файлу без підпису: 3.0 МБ

Результат перевірки підпису: Підпис створено та перевірено успішно. Цілісність даних підтверджено

Підписувач: Бизов Іван Сергійович
П.І.Б.: Бизов Іван Сергійович
Країна: Україна
РНОКПП: 3602504737
Час підпису (підтверджено кваліфікованою позначкою часу для підпису від Надавача): 12:31:39 19.05.2026
Сертифікат виданий: "Дія". Кваліфікований надавач електронних довірчих послуг
Серійний номер: 514B5C86A1E5DA11040000006A6ABC00CFD59605
Тип носія особистого ключа: ЗНКІ криптомодуль ІІТ Гряда-301
Алгоритм підпису: ДСТУ 4145
Тип підпису: Кваліфікований
Тип контейнера: Підписаний PDF-файл (PAdES)
Формат підпису: З повними даними для перевірки (PAdES-B-LT)
Сертифікат: Кваліфікований

Версія від: 2026.04.06 13:00