DOI: 10.58254/viti.9.2026.23.287

УДК 004.75 + 004.41 + 004.052.2

канд. техн. наук, доцент Троцько О. О. ORCID: 0000-0001-7535-5023 (ВІТІ ім. Героїв Крут)
Кондрусь А. В. ORCID: 0000-0001-8815-6517 (ВІТІ ім. Героїв Крут)
Пилипчук І. Ю. ORCID: 0000-0002-5550-8026 (ВІТІ ім. Героїв Крут)
Крамарчук В. В. ORCID: 0009-0002-2720-8373 (ВІТІ ім. Героїв Крут)

ДОСЛІДЖЕННЯ МЕТОДИКИ ВІДНОВЛЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ СПЕЦІАЛЬНОГО ПРИЗНАЧЕННЯ ШЛЯХОМ ЗАСТОСУВАННЯ ПІДХОДУ ІНФРАСТРУКТУРА ЯК КОД

У статті досліджено підхід Інфраструктура як Код (IaC) як сучасний інструмент автоматизації розгортання, масштабування та відновлення інформаційних систем спеціального призначення. Обґрунтовано концепцію розділення життєвого циклу сервісів та життєвого циклу даних як ключової архітектурної передумови реалізації IaC-орієнтованого відновлення: збій ephemeral-рівня не впливає на цілісність persistent-рівня, а відновлення системи зводиться до перестворення сервісів із репозиторію з наступним монтуванням сховища даних.

Проведено синтетичний порівняльний експеримент для трьох сценаріїв відновлення типової on-premise трирівневої вебінформаційної системи: ручного відновлення за планом відновлення після катастроф – документом, часткового IaC та повного IaC-підходу. Кожен сценарій відтворювався п'ять разів; для оцінювання достовірності результатів застосовано критерій Стьюдента з довірчим інтервалом 95 %. За результатами вимірювань отримано кількісні оцінки метрик часу відновлення Time Recovery Service, коефіцієнта автоматизації та цільової точки відновлення Recovery Point Objective. Запропоновано математичну формалізацію показників ефективності IaC з урахуванням статистичного розкиду вимірювань.

Встановлено, що повний IaC-підхід забезпечує скорочення часу відновлення сервісу до 30 хв порівняно з 315 хв при ручному відновленні – індекс ефективності 90,5 %, коефіцієнт автоматизації Ka = 87,5 %. Показано, що при правильній організації персистентного зберігання даних на рівнях 3–5 запропонованої ієрархії план відновлення після катастроф може бути функціонально замінений виконуваним IaC-репозиторієм, що гарантує детерміноване та відтворюване відновлення незалежно від наявності профільного фахівця.

Ключові слова: *інфраструктура як код, IaC, Terraform, Ansible, Docker, Kubernetes, розділення життєвих циклів, відновлення після збоїв, Time Recovery Service, Recovery Time Objective, Recovery Point Objective, надійність інформаційних систем, автоматизація розгортання.*

O. Trotsko, A. Kondrus, I. Pylypchuk, V. Kramarchuk. Research on the recovery methodology for special-purpose information systems through the application of the infrastructure as code approach

This paper investigates the Infrastructure as Code (IaC) approach as a modern tool for automating the deployment, scaling and recovery of special-purpose information systems. The concept of separating the service lifecycle from the data lifecycle is substantiated as a key architectural prerequisite for IaC-oriented recovery: a failure at the ephemeral layer does not affect the integrity of the persistent layer, and system recovery reduces to recreating services from the repository followed by mounting the data storage.

A synthetic comparative experiment was conducted for three recovery scenarios of a typical on-premise three-tier web information system: manual recovery using a Disaster Recovery Plan document, partial IaC, and full IaC approach. Each scenario was repeated five times; Student's t-criterion with a 95% confidence interval was applied to assess the reliability of the results. Based on the measurements, quantitative estimates were obtained for the Time to Restore Service, the Automation Coefficien, and the Recovery Point Objective. A mathematical formalization of IaC efficiency metrics is proposed, accounting for the statistical variance of measurements.

It is established that the full IaC approach reduces the service recovery time to 30 minutes compared to 315 minutes for manual recovery – an efficiency index of 90.5%, with an automation coefficient Ka = 87.5%. It is demonstrated that with proper organization of persistent data storage at levels 3-5 of the proposed hierarchy, the Disaster Recovery Plan can be functionally replaced by an executable IaC repository, guaranteeing deterministic and reproducible recovery regardless of the availability of a qualified specialist.

Keywords: *infrastructure as code, IaC, Terraform, Ansible, Docker, Kubernetes, lifecycle separation, disaster recovery, Time to Restore Service, Recovery Time Objective, Recovery Point Objective, information system reliability, automated deployment.*

Вступ. Сучасні інформаційні системи спеціального призначення функціонують в умовах підвищених вимог до доступності, стійкості до відмов та часу відновлення після збоїв. Забезпечення цих характеристик традиційно реалізується через розробку та підтримку детальних планів відновлення після катастроф Disaster Recovery Plan (DRP). Проте практичний досвід свідчить, що DRP-документація часто виявляється застарілою, неповною або важко відтворюваною в умовах реального інциденту [1; 2].

Підхід Інфраструктура як Код Infrastructure as Code (IaC) [9] пропонує принципово інший спосіб організації інфраструктури: весь стек розгортання описується декларативним або процедурним кодом, що зберігається в системі контролю версій. Це дозволяє відтворити будь-яке середовище автоматично, детерміновано та незалежно від суб'єктивного чинника оператора [3; 4]. Ключовою умовою ефективності такого підходу є чітке розмежування між ресурсами, що підлягають автоматичному перестворенню (сервіси), та ресурсами, що мають забезпечувати безперервне зберігання (дані) [5].

Аналіз останніх досліджень і публікацій свідчить, що проблематику застосування IaC у контексті відмовостійкості та відновлення інформаційних систем досліджували вітчизняні та зарубіжні науковці. Додонов О. Г. та ін. [1] розробили технологію забезпечення живучості територіально-розподілених інформаційних комп'ютерних систем в єдиному інформаційному просторі. Білоконь А. С. та ін. [2] провели аналіз функціонування розподілених систем обробки та зберігання даних. Мірошниченко Д. та Толстолузька О. [3] здійснили комплексне оцінювання ефективності методології IaC для хмарної інфраструктури, порівнявши інструменти Terraform, Pulumi, Ansible та CloudFormation. Копцев О. О. та ін. [4] дослідили особливості автоматичного розгортання інфраструктури як коду для хмарних сервісів. Яскевич В. Л. та Клочко О. В. [6] розглядали методи підвищення відмовостійкості інтернет-сервісів у розподілених інформаційних системах. Kapiton A. та ін. [7] обґрунтували вимоги до сучасних компонентів інформаційних систем з точки зору безпеки їх функціонування. Фролов Д. Є. [8] запропонував модель забезпечення відмовостійкості інформаційних систем на основі багаторівневих структур. Двірна О. А. та Набока С. В. [5] систематизували методи оптимізації продуктивності хмарних сервісів, включаючи Kubernetes-оркестрацію та автоматичне масштабування. Дяченко Д. О. та ін. [15] розробили підходи до моделювання ресурсовідновлення розподілених інформаційних систем. Міжнародні дослідники Morris K. [9] та Kim G. et al. [10] сформували концептуальний фундамент IaC як практики DevOps, що набула широкого практичного застосування.

Разом з тим, питання формального обґрунтування принципу розділення життєвого циклу сервісів та життєвого циклу даних, а також кількісного порівняння ефективності IaC-підходу відносно класичного DRP у контексті інформаційних систем спеціального призначення залишається недостатньо дослідженим. Це зумовлює актуальність та **наукове завдання** для цієї роботи.

Метою статті є дослідження ефективності відновлення після критичних збоїв ІССП шляхом застосування підходу Інфраструктура як код.

Виклад основного матеріалу. На основі синтетичного порівняльного експерименту кількісно оцінити ефективність трьох сценаріїв відновлення типової on-premise інформаційної системи (ручного, часткового IaC та повного IaC) за метриками Time to Restore Service (TRS), Automation Coefficient (Ka) та Recovery Point Objective (RPO), формалізувати систему показників оцінювання IaC-ефективності та обґрунтувати умови застосування IaC-репозиторію як виконуваної альтернативи плану відновлення після катастроф (DRP) з урахуванням ієрархії рівнів персистентності даних.

Концепція розділення життєвих циклів сервісів і даних передбачає наступні теоретичні засади розділення ресурсів, а саме будь-яка інформаційна система може бути декомпонована на два принципово різних класи ресурсів:

Ephemeral-ресурси *(тимчасові)* – програмні компоненти, мікросервіси, контейнери, конфігурації операційних систем, мережеві правила. Ці ресурси не мають власної тривалої цінності поза контекстом їхнього коду та конфігурації. За наявності IaC-репозиторію вони можуть бути відтворені у будь-який момент часу.

Persistent-ресурси *(постійні)* – бази даних, файлові сховища, сертифікати, облікові записи користувачів, транзакційні журнали. Ці ресурси мають незамінну бізнес-цінність і не можуть бути відтворені програмно – лише збережені та захищені.

Принцип розділення полягає в тому, що архітектура інформаційної системи має бути спроєктована так, щоб два класи ресурсів – ephemeral та persistent – мали повністю незалежні шляхи відмов і відновлення. Це означає, що збій або повне знищення ephemeral-рівня (контейнера, віртуальної машини, вузла кластера) не повинен жодним чином впливати на цілісність, доступність чи консистентність persistent-рівня.

На практиці реалізація цього принципу передбачає виконання кількох архітектурних вимог. По-перше, жоден сервіс не повинен зберігати критичні дані всередині власного контейнера або на локальному диску вузла – всі дані, що мають цінність або критичність, виносяться на зовнішні рівні персистентності (томи, об’єктне сховище, керована база даних). По-друге, сервіси мають проєктуватися як stateless або з мінімально можливим локальним станом: будь-який екземпляр сервісу повинен бути функціонально рівноцінним будь-якому іншому екземпляру, запущеному з того самого образу. По-третє, параметри підключення до сховища даних (рядки з’єднання, облікові дані, адреси кінцевих точок) передаються через механізми управління секретами (Vault, Kubernetes Secrets, змінні середовища), а не вбудовуються в код або конфігурацію образу.

За умови дотримання цих вимог відновлення системи після будь-якого збою ephemeral-рівня зводиться до двох послідовних операцій: перестворення всіх ephemeral-компонентів з IaC-репозиторію та монтування persistent-сховища до відновлених сервісів. При цьому дані залишаються недоторканними протягом усього часу відновлення — вони фізично не беруть участі в процесі перерозгортання інфраструктури. Саме ця властивість перетворює IaC-репозиторій на повноцінний операційний замінник традиційного плану відновлення після катастроф: замість покрокової інструкції для оператора система має виконуваний код, що детерміновано відтворює весь ephemeral-рівень у будь-який момент часу.

На рисунку 1 зображено порівняльну архітектурну модель традиційного підходу, де обидва класи ресурсів тісно зв’язані в межах одного середовища, та IaC-підходу, де реалізовано чітке розділення.

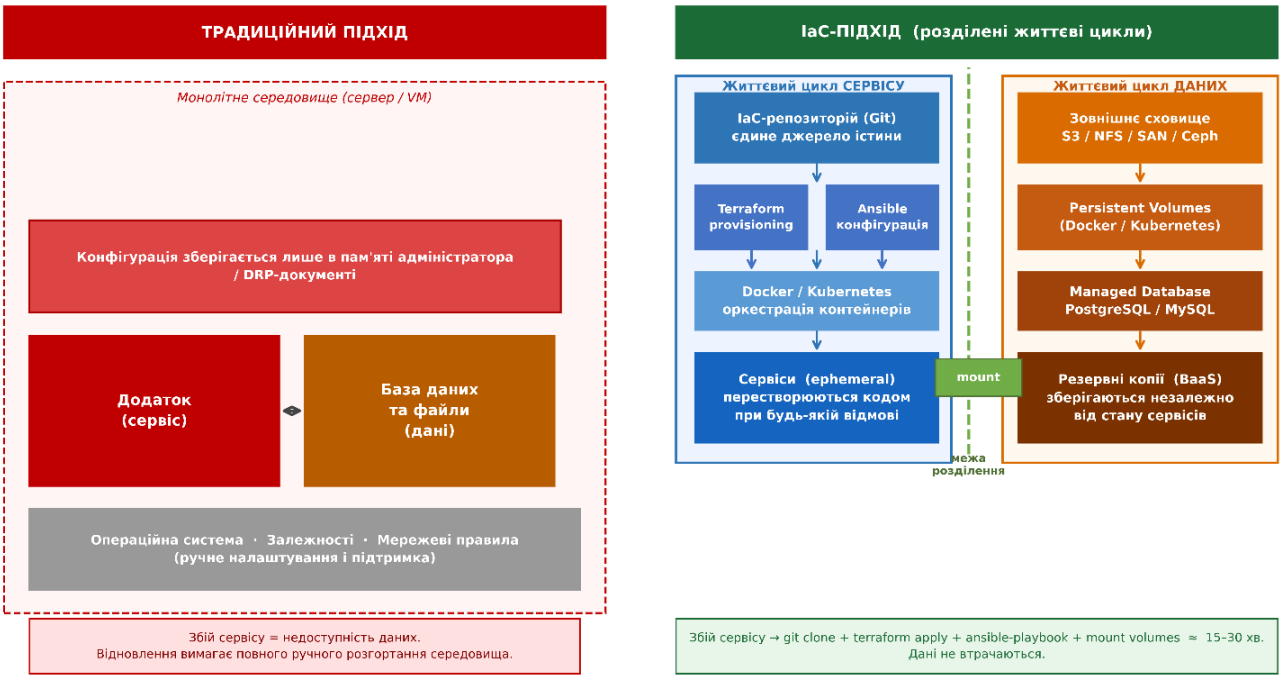

Рис. 1. Порівняльна архітектурна модель: традиційний підхід та IaC-підхід до організації інформаційних систем

Методологія IaC: класифікація та інструментарій. Методологія IaC реалізується через два основних підходи: декларативний та процедурний (імперативний). У декларативному підході оператор описує бажаний кінцевий стан системи, а інструмент самостійно визначає послідовність дій для його досягнення. У процедурному підході описується точна послідовність операцій [9].

Провідні IaC-інструменти можна класифікувати за рівнем абстракції та функціональним призначенням:

Terraform (HashiCorp) – декларативний інструмент виділення ресурсів інфраструктури. Управляє ресурсами хмарних провайдерів, мережевою інфраструктурою, сховищами через HCL-синтаксис. Підтримує state-файли для відстеження поточного стану інфраструктури [12].

Ansible (Red Hat) – гібридний (декларативний/процедурний) інструмент конфігураційного управління. Виконує налаштування ОС, встановлення пакетів, розгортання сервісів через SSH без агента на вузлах [13].

Pulumi – декларативний інструмент керування ресурсами інфраструктури, що використовує загальновживані мови програмування (Python, TypeScript, Go). Особливо доцільний при наявності складної логіки генерації конфігурацій [12].

Docker Compose / Kubernetes Helm – інструменти оркестрації рівня контейнерних додатків. Описують склад сервісів, мережеву взаємодію та монтування томів у декларативних YAML-маніфестах [5; 14].

IaC як функціональна альтернатива плану відновлення після катастроф. Класичний DRP являє собою документ, що описує послідовність ручних дій з відновлення системи. Його фундаментальним недоліком є залежність від актуальності документації та кваліфікації персоналу, що виконує відновлення [1; 6]. На противагу цьому, IaC-репозиторій є виконуваним артефактом – «живим DRP», що автоматично відображає поточний стан інфраструктури за умови дотримання практики «інфраструктура в коді» (все, що розгорнуто, має відповідний код у репозиторії).

Умовою коректного застосування IaC як DRP є обов'язкове виконання принципу розділення життєвого циклу: якщо дані зберігаються незалежно від ephemeral-рівня, то

відновлення системи зводиться до трьох операцій: (1) клонування IaC-репозиторію, (2) виконання коду підготовки ресурсів інфраструктури, (3) монтування persistent-сховища. Такий підхід формалізовано у вигляді процесу відновлення на рисунку 2.

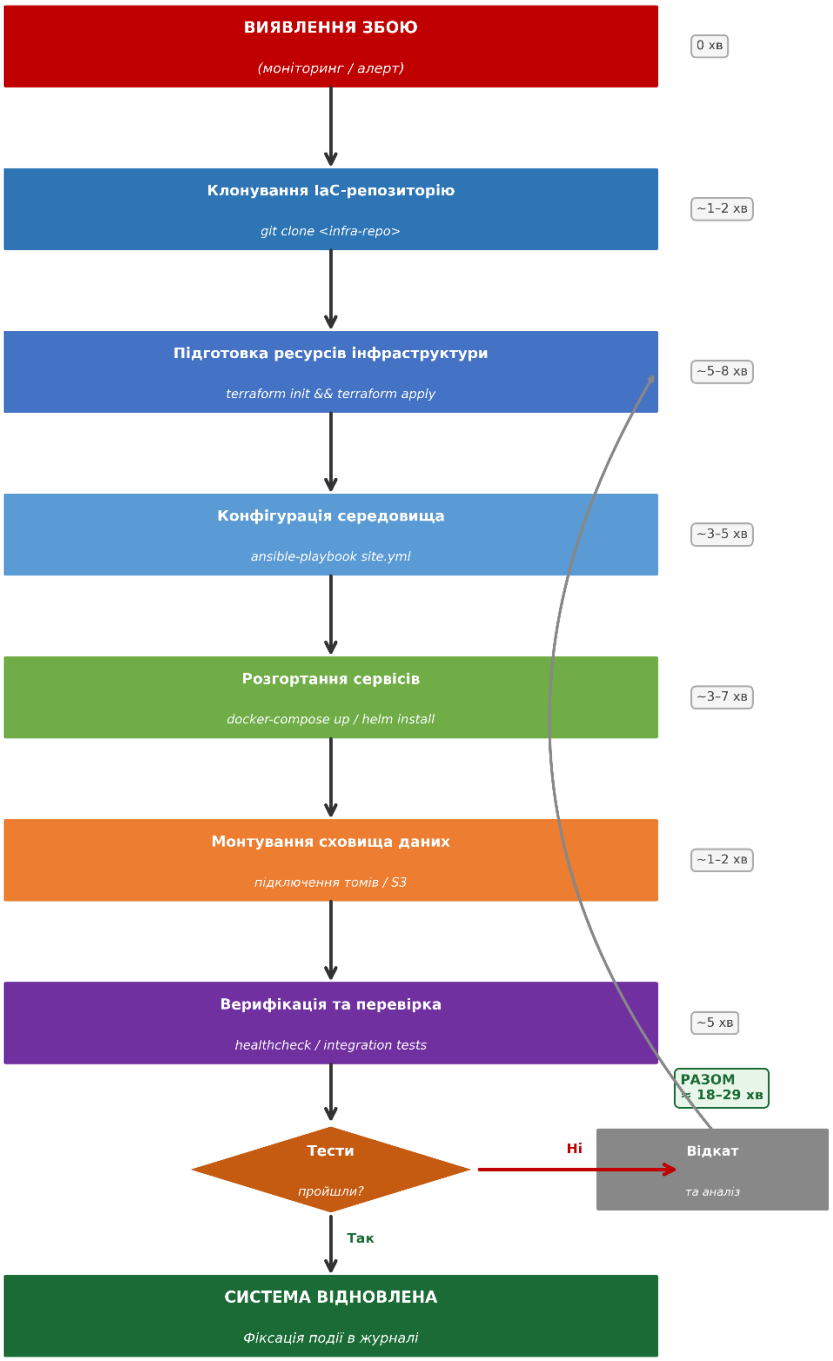

Рис. 2. Процес відновлення інформаційної системи за IaC-методологією

Згідно з рисунком 2, загальний час відновлення за повного IaC-підходу (сценарій С) складає 18–29 хв. Для порівняння: у синтетичному експерименті відновлення тієї самої інформаційної системи вручну (сценарій А) зайняло в середньому 315 хв за п'ятьма вимірами. Об'єктом відновлення у всіх трьох сценаріях слугувала типова трирівнева on-premise вебінформаційна система, що складається з чотирьох компонентів: Nginx (зворотний проксі та вебсервер), PHP-FPM (середовище виконання застосунку), PostgreSQL 15 (реляційна СУБД,

база даних обсягом 2,5 ГБ) та Redis (кеш-рівень). База даних у всіх сценаріях зберігалась на зовнішньому томі поза контейнером і не підлягала відновленню – відновлювався виключно програмно-конфігураційний (ephemeral) стек. Вхідними даними експерименту слугували: чистий хостовий сервер Ubuntu Server 22.04 LTS (8 vCPU, 16 ГБ RAM, 200 ГБ SSD) без попередньо встановленого програмного забезпечення, DRP-документ для сценарію A, Ansible-playbook для сценарію B та повний IaC-репозиторій (Terraform + Ansible + Docker Compose, 26 файлів) для сценарію C.

Математична формалізація метрик ефективності IaC. Для кількісного оцінювання ефективності IaC-підходу адаптовано та систематизовано відомі метрики надійності інформаційних систем [6; 8] стосовно специфіки IaC-сценаріїв відновлення. Обрані метрики (TRS, Ka, RPO) є підмножиною більш широкого спектра показників DevOps/SRE-практик, зокрема Change Failure Rate (CFR), Rollback Frequency та Configuration Drift Rate. Зазначені показники виходять за межі цього дослідження, оскільки потребують тривалого моніторингу продуктивної системи, тоді як у межах синтетичного експерименту вони не можуть бути виміряні коректно. Перспективи їх включення до системи оцінювання розглянуто в підрозділі «Висновки».

Час відновлення сервісу (TRS). Загальний час відновлення сервісу TRS визначається як сума часу виконання кожного етапу процесу відновлення згідно з виразом (1):

$$\mathrm{TRS} = T_0 + T_1 + T_2 + T_3 + T_4, \tag{1}$$

де T_0 – час виявлення збою; T_1 – час підготовки ресурсів інфраструктури (Terraform); T_2 – час конфігурації середовища (Ansible); T_3 – час розгортання сервісів (Docker/Helm); T_4 – час монтування сховища та верифікації.

Коефіцієнт автоматизації відновлення (Ка) характеризує частку операцій, що виконуються без участі людини-оператора. Значення Ka = 80 % прийнято як цільовий рівень автоматизації відповідно до рекомендацій DORA (DevOps Research and Assessment) [10], згідно з якими організації з високим рівнем DevOps-зрілості автоматизують не менше 80 % процесів розгортання та відновлення. Цей поріг також корелює з практикою SRE (Site Reliability Engineering) [11], де ручні операції понад 20 % від загального часу роботи інженера (так зване «toil») вважаються неприйнятними для систем з вимогами до доступності 99,9 % і вище:

$$\mathrm{K_a} = (N_{\mathrm{au}}{}^{\mathrm{t}}{}_0 / N_{\mathrm{t0tal}}) \times 100\ \%, \tag{2}$$

де $N_{au}{}^{t}{}_{0}$ – кількість автоматизованих операцій відновлення;

N_{t0tal} – загальна кількість операцій відновлення.

Індекс ефективності IaC-підходу ($\mathrm{E^{I}_{a}{}^{c}}$) відносно базового сценарію ручного відновлення визначається за формулою (3):

$$\mathrm{E^{I}_{a}{}^{c}} = (TRS^{\mathrm{MaNUAL}} - TRS^{\mathrm{I}}_{\mathrm{a}}{}^{\mathrm{c}}) / TRS^{\mathrm{MaNUAL}} \times 100\ \%, \tag{3}$$

де TRS^{MaNUAL} – час відновлення при ручному процесі;

$TRS^{I}_{a}{}^{c}$ – час відновлення при IaC-підході.

Цільова точка відновлення для IaC-підходу ($\mathrm{RPO^{I}_{a}{}^{c}}$). При IaC-підході RPO визначається не тільки частотою резервного копіювання даних, а й частотою фіксації змін в IaC-репозиторії:

$$\mathrm{RPO^{I}_{a}{}^{c}} = \max(T^{\mathrm{b}}{}_{\mathrm{a}}{}^{\mathrm{c}\prime\mathrm{u}}{}_{\mathrm{u}}{}^{\mathrm{p}}, T^{\mathrm{gITcommIT}}), \tag{4}$$

де T^{backup} – інтервал між резервними копіями даних;
$T^{gITcommIT}$ – інтервал між git-комітами IaC-коду.

Методика проведення дослідження. З метою кількісного обґрунтування ефективності IaC-підходу проведено синтетичний порівняльний експеримент. Термін «синтетичний» означає, що умови відновлення системи відтворювались у контрольованому ізольованому тестовому середовищі, без впливу сторонніх виробничих факторів (мережевого навантаження, паралельних процесів, втручання користувачів), але з дотриманням реалістичних часових і ресурсних характеристик кожного сценарію [8; 15]. Такий підхід забезпечує відтворюваність результатів та дозволяє виключити випадкові зовнішні чинники.

Тестове середовище відтворювало типову on-premise конфігурацію виділеного сервера – клас інфраструктури, широко застосовуваний в інформаційних системах спеціального призначення, де використання публічних хмарних платформ може бути обмежене вимогами безпеки. Обрана конфігурація: хостовий сервер з ОС Ubuntu Server 22.04 LTS (8 vCPU, 16 ГБ RAM, 200 ГБ SSD NVMe); стек розгортання – Docker Engine 24.0, Docker Compose 2.x, Terraform 1.6, Ansible 2.15; тип ІС – типовий трирівневий вебзастосунок (Nginx як зворотний проксі + PHP-FPM як середовище виконання + PostgreSQL 15 як СУБД + Redis як кеш-рівень); обсяг бази даних – 2,5 ГБ, збережена на зовнішньому томі (bind mount за межами контейнера); IaC-репозиторій – Git-репозиторій (26 файлів) з Terraform-модулями (мережа, firewall, томи) та Ansible-ролями (nginx, php-fpm, docker, security). Саме така архітектура обрана тому, що вона охоплює всі ключові компоненти більшості реальних ІС: вебсервер, середовище виконання, СУБД і кеш – і дозволяє виміряти час відновлення на рівні кожного компонента окремо. Ресурсні витрати на підготовку IaC-репозиторію склали орієнтовно 40–60 людино-годин для повного опису інфраструктури, що є одноразовою інвестицією, яка окупається вже за першого випадку відновлення.

Визначено три сценарії відновлення:

Сценарій А (Ручне відновлення) – відновлення системи відповідно до DRP-документа без використання IaC. Оператор виконує всі дії вручну: встановлення ОС, залежностей, конфігурація Nginx/PHP-FPM, підключення БД, налаштування мережевих правил.

Сценарій В (Частковий IaC) – наявний Ansible-playbook для конфігурації середовища, але підготовка ресурсів інфраструктури та оркестрація сервісів виконуються частково вручну.

Сценарій С (Повний IaC) – повністю автоматизований процес: terraform apply → ansible-playbook → docker-compose up. Оператор виконує лише одну операцію – ініціацію процесу та кінцеву верифікацію.

Результатом експерименту є результати вимірювань часу кожного етапу відновлення для трьох сценаріїв (табл. 1). Кожен сценарій відтворювався 5 разів – мінімальна кількість, що забезпечує розрахунок середнього значення та стандартного відхилення при малих вибірках (за критерієм Стьюдента для довірчого інтервалу 95 %). Наведено середні значення; стандартне відхилення TRS не перевищувало ±8 хв для сценарію А та ±2 хв для сценарію С, що підтверджує стабільність результатів.

Таблиця 1

Результати синтетичного порівняльного експерименту

Показник	Сценарій А (Ручне)	Сценарій В (Частковий IaC)	Сценарій С (Повний IaC)
Загальна кількість операцій (N_total)	23	12	8
Кількість автоматизованих операцій (N_auto)	4	8	7
Коефіцієнт автоматизації Ka (%)	17,4	66,7	87,5
Час виявлення збою T0 (хв)	15	10	5

Показник	Сценарій A (Ручне)	Сценарій B (Частковий IaC)	Сценарій C (Повний IaC)
Час підготовки ресурсів T1 (хв)	120	30	8
Час конфігурації T2 (хв)	90	20	5
Час розгортання сервісів T3 (хв)	60	15	7
Час монтування та верифікації T4 (хв)	30	15	5
Час відновлення TRS (хв)	315	90	30
Цільова точка відновлення RPO	до 24 год	до 8 год	до 1 год
Залежність від DRP-документа	Критична	Висока	Мінімальна

Розрахунок показників за формулами (1)–(4).

За формулою (1) для кожного сценарію отримано значення TRS:

$TRS^A = 15 + 120 + 90 + 60 + 30 = 315$ хв (Сценарій A);

$TRS^B = 10 + 30 + 20 + 15 + 15 = 90$ хв (Сценарій B);

$TRS^c = 5 + 8 + 5 + 7 + 5 = 30$ хв (Сценарій C).

Коефіцієнт автоматизації за формулою (2):

$K_a(A) = 4/23 \times 100\ \% \approx 17{,}4\ \%$;

$K_a(B) = 8/12 \times 100\ \% \approx 66{,}7\ \%$;

$K_a(C) = 7/8 \times 100\ \% = 87{,}5\ \%$.

Індекс ефективності IaC за формулою (3) при порівнянні сценаріїв A та C:

$E^{I}{}_{a}{}^{c} = (315 - 30) / 315 \times 100\ \% = 285/315 \times 100\ \% \approx 90{,}5\ \%$

Отже, впровадження повного IaC-підходу забезпечує скорочення часу відновлення на 90,5 % відносно ручного процесу. Цільова точка відновлення RPO за формулою (4) при щогодинному git-commit та щоденному backup даних:

$RPO^{I}{}_{a}{}^{c} = \max(24\text{ год}, 1\text{ год}) = 24$ год (при щоденному backup);

$RPO^{I}{}_{a}{}^{c} = \max(4\text{ год}, 0{,}25\text{ год}) = 4$ год (при резервному копіюванні кожні 4 год).

Таким чином, при застосуванні повного IaC-підходу вибір RPO стає виключно питанням частоти резервного копіювання даних, а не складності відновлення сервісів, оскільки програмно-конфігураційний стек відтворюється автоматично і не потребує участі оператора у прийнятті рішень. Це є принциповою перевагою IaC-підходу порівняно з традиційним DRP.

Водночас необхідно об’єктивно оцінити ресурсні витрати на впровадження такого підходу. Одноразові витрати на розробку повного IaC-репозиторію для інформаційної системи описаної конфігурації складають орієнтовно 40–60 людино-годин кваліфікованої інженерної праці, що включає: декларативний опис інфраструктури засобами Terraform (мережева конфігурація, правила розмежування доступу, томи зберігання), розробку Ansible-ролей для налаштування середовища (вебсервер, середовище виконання, параметри захисту) та формування Docker Compose-маніфесту для оркестрації сервісів. Додатково слід враховувати постійні витрати на супровід репозиторію: кожна зміна конфігурації системи має бути відображена у коді, що вимагає відповідної кваліфікації обслуговуючого персоналу та організаційної дисципліни.

Критично важливим аргументом на користь IaC в контексті інформаційних систем спеціального призначення є не порівняння витрат, а гарантія відтворюваності: на відміну від DRP-документа, актуальність якого залежить від людського чинника, IaC-репозиторій зі своєї природи є синхронізованим із реальним станом системи за умови дотримання практики «все розгорнуте – описане у коді». В умовах обмеженого часу на відновлення та можливої відсутності профільного фахівця на місці, детермінована автоматизована процедура

відновлення є суттєвою перевагою перед покроковою ручною інструкцією – незалежно від обсягу підготовчих витрат.

Як цільовий орієнтир для порівняння обрано значення Ka = 80 %, що відповідає рекомендаціям DORA щодо мінімального рівня автоматизації процесів розгортання та відновлення для зрілих DevOps-організацій [10], а також межі допустимого «toil» в практиці SRE [11].

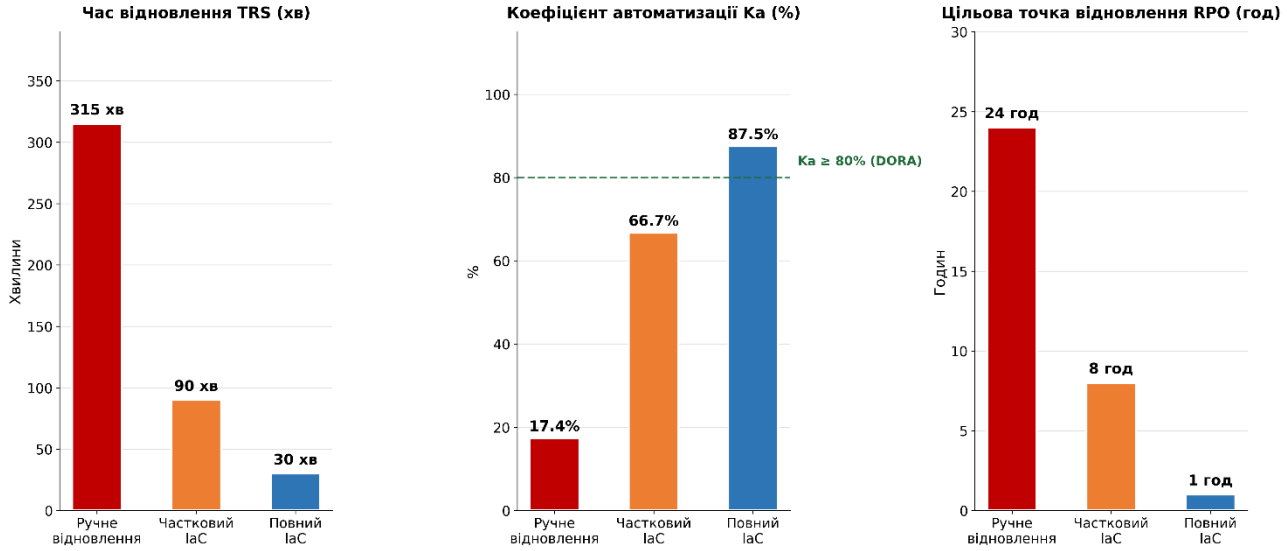

Рис. 3. Порівняльні результати синтетичного експерименту:
метрики відновлення для трьох сценаріїв (A, B, C)

Результати на рисунку 3 демонструють нелінійне покращення метрик при переході між сценаріями. Перехід A → B дає найбільший приріст *Ka* (+49,3 п.п.) завдяки автоматизації найтриваліших ручних операцій (налаштування ОС та СУБД). Перехід B → C дає менший приріст *Ka* (+20,8 п.п.), але значно більший ефект по RPO (скорочення з 8 год до 1 год) – завдяки декларативному опису підключення зовнішнього сховища S3 в IaC-конфігурації. Водночас необхідно зазначити обмеження підходу: впровадження повного IaC потребує одноразових витрат на розробку та супровід репозиторію (40–60 год), а також підвищених вимог до кваліфікації інженерного персоналу. Для систем із рідкісними інцидентами (менше 1–2 на рік) ці витрати можуть не окупитися, що підтверджує необхідність індивідуального аналізу доцільності впровадження IaC у кожному конкретному випадку.

Ієрархія рівнів персистентності даних в IaC-архітектурі. Для практичної реалізації принципу розділення важливо визначити ієрархію рівнів персистентності даних. Кожен рівень характеризується своїм часом збереження, відмовостійкістю та способом монтування з боку IaC-коду.

Рівні персистентності, від найнижчого до найвищого, утворюють чітку структуру: контейнерна файлова система (ephemeral) → Docker Volumes / Bind Mounts → зовнішнє мережеве сховище (NFS/iSCSI/S3) → керовані бази даних (Managed DB) → резервні копії (BaaS/snapshot). На рисунку 4 IaC-репозиторій описує монтування для кожного рівня відповідно до вимог конкретного сервісу [9; 11].

Рис. 4. Ієрархія рівнів персистентності даних в IaC-архітектурі

Рівень 1 (Ephemeral) охоплює внутрішню файлову систему контейнера та оперативну пам'ять. Дані на цьому рівні втрачаються при перезапуску контейнера і не підлягають захисту. Сюди відносяться тимчасові файли, кеш-дані, сесійна інформація в пам'яті. IaC-код не описує монтування для цього рівня.

Рівень 2 (Persistent Volumes) включає Docker Volumes та Bind Mounts. Дані зберігаються при рестарті контейнера, але пов'язані з конкретним хостом. У docker-compose.yml монтування описується директивою volumes:. При відмові хоста дані на цьому рівні потребують окремого механізму реплікації.

Рівень 3 (Зовнішнє сховище) включає NFS, iSCSI, розподілені файлові системи (Ceph, GlusterFS) та S3-сумісні об'єктні сховища (MinIO, AWS S3). Дані доступні незалежно від стану хоста. У Terraform цей рівень описується як ресурс (aws_s3_bucket, oci_object_storage_bucket тощо). Це є мінімально достатнім рівнем для реалізації IaC як DRP [9; 11].

Рівень 4 (Managed Database) – зовнішня керована база даних (PostgreSQL/MySQL як окремий контейнер або хмарний managed-сервіс). Забезпечує реплікацію, автоматичний failover та point-in-time recovery. IaC описує параметри підключення через змінні середовища або secrets-менеджер.

Рівень 5 (Резервне копіювання) – snapshot-резервування та BaaS (Backup-as-a-Service). Забезпечує довгострокове зберігання та географічну розподіленість. Інтервал резервного копіювання безпосередньо визначає RPO системи відповідно до формули (4).

Висновки.

1. Обґрунтовано концепцію розділення життєвого циклу сервісів та даних – підхід до архітектурного проєктування ІС, що базується на положеннях теорії надійності [6; 8] та практиці SRE [11] і передбачає незалежність шляхів відмов ephemeral- та persistent-рівнів. Встановлено, що лише за умови зберігання даних на рівнях 3–5 ієрархії персистентності (зовнішнє сховище, Managed DB, BaaS) IaC-репозиторій може виступати повноцінним виконуваним замінником DRP-документа.

2. Проведено синтетичний порівняльний експеримент для трьох сценаріїв відновлення ІС. Отримано кількісні значення показників: TRS знижується із 315 хв (ручний процес) до

30 хв (повний IaC); *Ka* зростає з 17,4 % до 87,5 %; RPO скорочується з 24 год до 1 год. Індекс ефективності IaC ($E^{I}_{a}{}^{c}$) становить 90,5 %.

3. Адаптовано та систематизовано систему метрик оцінювання IaC-ефективності стосовно специфіки синтетичного порівняльного експерименту: час відновлення TRS (формула (1)), коефіцієнт автоматизації *Ka* (формула (2)) з обґрунтованим цільовим рівнем 80 % [10; 11], індекс ефективності $E^{I}_{a}{}^{c}$ (формула (3)) та RPO для IaC-підходу (формула (4)). Метрики Change Failure Rate, Rollback Frequency та Configuration Drift Rate, що широко застосовуються у DevOps-практиці, виходять за межі синтетичного експерименту і потребують тривалого моніторингу продуктивної системи – їх включення до системи оцінювання є перспективою подальших досліджень.

4. Встановлено, що ефект переходу між сценаріями є нелінійним: приріст *Ka* при переході A → B (+49,3 п.п.) більший, ніж при B → C (+20,8 п.п.), однак покращення RPO є значно більш вираженим на другому переході (з 8 год до 1 год). Це пояснюється тим, що на першому переході автоматизуються найбільш трудомісткі ручні операції, а на другому – усувається залежність від локального зберігання даних. Водночас виявлено, що впровадження повного IaC потребує одноразових витрат (40–60 год інженерної праці) та підвищеної кваліфікації персоналу, що може бути обмежувальним фактором для малих команд.

5. **Наукова новизна роботи полягає** у: (а) обґрунтуванні концепції розділення життєвого циклу, розробці системи кількісних метрик ефективності IaC стосовно специфіки сценаріїв відновлення; (б) кількісному обґрунтуванні цільового рівня автоматизації Ka = 80 % на основі DORA-рекомендацій; (в) експериментальному виявленні нелінійності ефекту переходу між сценаріями та виявленні обмежень IaC-підходу щодо ресурсних витрат на впровадження.

Перспективи подальших досліджень пов’язані з: розширенням системи метрик оцінювання IaC шляхом включення Change Failure Rate, Rollback Frequency та Configuration Drift Rate на основі даних тривалого моніторингу продуктивних систем; аналізом застосування IaC в умовах мультихмарного розгортання; дослідженням GitOps-підходів у Kubernetes-середовищах; розробкою методики автоматизованого тестування відновлення (Chaos Engineering) з використанням IaC-інструментів; порівняльним дослідженням ефективності IaC на інших конфігураціях інфраструктури (мікросервісна архітектура, edge-computing, приватна хмара).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Додонов О. Г., Путятін В. Г., Додонов В. О., Куценко С. А., Германюк А. П., Ізварін І. В., Кравчук К. О. Технологія забезпечення живучості територіально-розподілених інформаційних комп’ютерних систем в єдиному інформаційному просторі // Реєстрація, зберігання і обробка даних. 2024. Т. 26, № 1. С. 121–143. DOI: 10.35681/1560-9189.2024.26.1.308659.

2. Білоконь А. С., Борисов С. О., Усатенко М. В., Федорченко В. М. Аналіз функціонування розподілених систем обробки та зберігання даних // Системи управління, навігації та зв’язку. 2024. № 3. С. 84–89. DOI: 10.26906/SUNZ.2024.3.084.

3. Miroshnychenko D., Tolstoluzka O. Evaluation of the Effectiveness of the “Infrastructure as Code” Methodology for Creating and Managing Cloud Infrastructure // Bulletin of NTU “KhPI”. Ser.: System Analysis, Control and Information Technologies. 2025. № 1 (13). DOI: 10.20998/2079-0023.2025.01.14.

4. Копцев О. О., Мартовицький В. О., Бологова Н. М., Федак І. Б. Особливості автоматичного розгортання інфраструктури як коду для хмарних сервісів // Системи управління, навігації та зв’язку. 2024. № 1. С. 104–109. DOI: 10.26906/SUNZ.2024.1.104.

5. Двірна О. А., Набока С. В. Оптимізація продуктивності хмарних сервісів: методи та їх ефективність // Системи управління, навігації та зв’язку. 2026. № 1. С. 62–68. DOI: 10.26906/SUNZ.2026.1.062.

6. Яскевич В. Л., Клочко О. В. Методи підвищення відмовостійкості інтернет сервісів // Кібербезпека: освіта, наука, техніка. 2019. Т. 3, № 3. С. 104–111. DOI: 10.28925/2663-4023.2019.3.104111.

7. Kapiton A., Franchuk T., Tyshchenko D., Desiatko A., Zakharov R. Requirements for Modern Processors for Secure Operation of Information Systems and Networks // Системи управління, навігації та зв’язку. 2025. № 3. С. 101–105. DOI: 10.26906/SUNZ.2025.3.101-105.

8. Фролов Д. Є. Модель забезпечення відмовостійкості інформаційних систем на основі багаторівневих структур // Вісник Херсонського національного технічного університету. 2025. Т. 2, № 3 (94). DOI: 10.35546/kntu2078-4481.2025.3.2.61.

9. Morris K. Infrastructure as Code: Dynamic Systems for the Cloud Age. 2nd ed. Sebastopol: O'Reilly Media, 2021. 423 p.

10. Kim G., Humble J., Debois P., Willis J., Forsgren N. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. 2nd ed. Portland: IT Revolution Press, 2021. 480 p.

11. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol: O'Reilly Media, 2016. 552 p.

12. Brikman Y. Terraform: Up and Running. 3rd ed. Sebastopol: O'Reilly Media, 2022. 480 p.

13. Hochstein L., Moser R. Ansible: Up and Running. 3rd ed. Sebastopol: O'Reilly Media, 2022. 512 p.

14. Burns B., Beda J., Hightower K., Evenson L. Kubernetes: Up and Running: Dive into the Future of Infrastructure. 3rd ed. Sebastopol: O'Reilly Media, 2022. 326 p.

15. Дяченко Д. О., Гук А. С., Міхаль О. П. Моделювання ресурсовідновлення розподілених інформаційних систем // Системи управління, навігації та зв’язку. 2024. № 1. С. 61–65. DOI: 10.26906/SUNZ.2024.1.061.

Надійшла до редколегії 05.03.2026.
Схвалена до друку 22.05.2026.
Дата публікації 29.05.2026.