DOI: 10.20998/2079-0023.2025.01.14
УДК 004.02

Д. О. МІРОШНИЧЕНКО, аспірант, Харківський національний університет імені В. Н. Каразіна, м. Харків, Україна; e-mail: dmytro.miroshnychenko@student.karazin.ua; ORCID: https://orcid.org/0000-0003-1159-0985
О. Г. ТОЛСТОЛУЗЬКА, доктор технічних наук, старший науковий співробітник, професор кафедри комп'ютерних наук та робототехніки; Харківський національний університет імені В. Н. Каразіна, майдан Свободи, 4, Харків-22, Україна, 61022; e-mail: elena.tolstoluzka@karazin.ua; ORCID: https://orcid.org/0000-0003-1241-7906

ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДОЛОГІЇ «ІНФРАСТРУКТУРА ЯК КОД» ДЛЯ СТВОРЕННЯ ТА КЕРУВАННЯ ХМАРНОЮ ІНФРАСТРУКТУРОЮ

У роботі проведено комплексне дослідження ефективності застосування методології «Інфраструктура як Код» (Infrastructure as Code, IaC) для створення, масштабування та управління хмарною інфраструктурою. Методологія IaC розглядається як одна з ключових технологій цифрової трансформації та DevOps-підходу, що забезпечує програмну автоматизацію інфраструктурних процесів, зменшує залежність від людського фактора та підвищує повторюваність і передбачуваність IT-середовищ. У статті здійснено порівняльний аналіз провідних інструментів реалізації IaC, зокрема Terraform, Pulumi, AWS CloudFormation та Ansible, з позицій їх відкритості, сумісності з різними хмарними платформами, архітектурного підходу (декларативного або імперативного), управління станом та рівня гнучкості. У якості ключових метрик ефективності оцінено ступінь автоматизації, масштабованість, швидкість розгортання інфраструктури, адаптивність до змін, надійність конфігурацій та зручність управління. Для кожної метрики наведено теоретичне обґрунтування, аналітичну оцінку та порівняння з традиційними підходами адміністрування. Особливу увагу приділено аналізу реалізації IaC у провідних хмарних середовищах (AWS, Microsoft Azure, Google Cloud Platform, OpenStack) із розглядом відповідних платформних рішень (CloudFormation, ARM/Bicep, Deployment Manager, Heat) та сторонніх мультихмарних інструментів. Виявлено, що застосування IaC значно покращує DevOps-практики, спрощує CI/CD процеси та підвищує надійність хмарних рішень. У підсумку, доведено, що застосування IaC забезпечує значне підвищення операційної ефективності, зменшує витрати на обслуговування інфраструктури та сприяє її стандартизації, що робить цю методологію стратегічно важливою для сучасних IT-систем.

Ключові слова: Інфраструктура як Код, хмарні обчислення, Terraform, Pulumi, DevOps, CloudFormation, ефективність, автоматизація, управління конфігураціями, CI/CD.

Вступ. Сучасна практика DevOps і стрімкий розвиток хмарних технологій вимагають все більш автоматизованого та надійного підходу до управління інфраструктурою. Методологія Infrastructure as Code (IaC) – «Інфраструктура як Код» – виникла як відповідь на ці потреби. Ця методологія передбачає опис конфігурації та топології інфраструктури у вигляді коду чи наборів конфігурацій, що зберігаються у системах контролю версій. Це дозволяє автоматично розгортати та змінювати інфраструктуру за допомогою програмних інструментів замість ручних дій. Застосування IaC революціонізувало управління IT-інфраструктурою, трактуючи налаштування як версіонований код, що забезпечує автоматизацію, масштабованість і узгодженість середовищ [1]. У контексті DevOps IaC слугує містком між розробниками та IT-операціями, даючи змогу описувати середовище так само, як пишуться програми, й автоматично отримувати потрібний стан інфраструктури[2]. Це скорочує цикл доставки застосунків і мінімізує помилки, пов'язані з «людським фактором».

Мета та задачі дослідження.

Метою даного дослідження є оцінювання ефективності методології «Інфраструктура як Код» (Infrastructure as Code, IaC) як інструменту автоматизації створення, керування та масштабування хмарної інфраструктури, а також визначення її впливу на ключові метрики продуктивності в DevOps-практиках.

Для того, щоб досягти поставленої мети було сформульовано наступні основні задачі дослідження:

- Провести теоретичний огляд концепції IaC, її основних принципів, місця в контексті хмарної інженерії та ролі в автоматизованих IT-процесах.
- Здійснити порівняльний аналіз провідних інструментів IaC (Terraform, Pulumi, AWS CloudFormation, Ansible) за критеріями архітектури, відкритості, мультихмарності, управління станом та зручності використання.
- Визначити ключові метрики ефективності управління інфраструктурою та здійснити їх оцінку з використанням підходу IaC (автоматизація, масштабованість, час розгортання, гнучкість, надійність, управління конфігураціями).

Огляд методології IaC та її значення для DevOps. Інфраструктура як Код (IaC) – це підхід до управління інфраструктурою, в якому середовище (сервери, мережі, налаштування) описується мовою програмування або декларативною мовою конфігурації. Фактично, IaC дозволяє програмувати інфраструктуру: потрібний стан систем (наприклад, кількість і параметри серверів, мережеві правила, налаштування ПЗ) задається у вигляді коду, який потім використовується для автоматичного розгортання або зміни цього стану за допомогою спеціалізованих інструментів.

На відміну від традиційного підходу, де інженери вручну налаштовують кожен сервер і компонент, IaC автоматизує цей процес, що дає суттєві переваги. Зокрема, опис інфраструктури у вигляді коду підлягає версійному контролю, як і звичайний програмний код. Це означає, що всі зміни коду відстежуються, можна

повернутися до попередньої версії конфігурації, або точно визначити, хто і коли додав ті чи інші правки [3].

Таким чином, IaC забезпечує єдине джерело для конфігурацій: опис у репозиторії коду повністю відображає реальний стан інфраструктури (за умови дотримання принципів IaC), що виключає ситуації, коли документація не відповідає дійсності.

Для практик DevOps поява IaC стала одним із ключових факторів успіху. Завдяки IaC досягається постійна узгодженість середовищ: середовище розробки, тестування та продуктиву можуть розгортатися з одного набору конфігурацій [2]. Це підвищує якість релізів і спрощує експлуатацію. Крім того, автоматизація інфраструктури прискорює випуск нових версій додатків: замість витратних ручних процедур розгортання серверів, мереж тощо, команди можуть за хвилини розгорнути цілі середовища, виконавши IaC-скрипти. Як наслідок, зменшується time-to-market для нових функцій.

Дослідження показують, що впровадження IaC сприяє збільшенню частоти деплойментів без шкоди стабільності: автоматизація через IaC приводить до впровадження CI/CD конвеєрів, всебічного тестування та стандартизації процесів, що, у свою чергу, знижує ризик помилок [4]. IaC тісно пов'язаний з концепцією Mutable vs Immutable infrastructure. У традиційному підході сервери вважаються довгоживучими об'єктами, які адміністратори налаштовують і оновлюють з часом (mutable infrastructure). Натомість IaC заохочує immutable infrastructure – сервери і контейнери створюються згідно з описом і більше не змінюються вручну: для оновлення конфігурації їх замінюють новими, згенерованими за оновленим кодом. Це усуває дрейф конфігурацій (накопичення невідслідковуваних змін) і сприяє стабільності та передбачуваності середовищ.

Значення методології IaC для DevOps та хмарної інженерії трудно переоцінити. Вона забезпечує програмну керованість інфраструктури, що веде до прискорення процесів, підвищення надійності (через повторюваність і тестованість), кращої масштабованості та тіснішої співпраці між командами розробки та експлуатації.

Огляд популярних інструментів IaC. Існує декілька категорій інструментів, що реалізують підхід «Інфраструктура як Код». За сферами застосування їх поділяють на: інструменти провіжингу інфраструктури (Infrastructure Provisioning), що автоматизують створення хмарних ресурсів (віртуальних машин, мереж, БД тощо); інструменти управління конфігураціями (Configuration Management), що автоматизують налаштування ПЗ на вже створених серверах; а також інструменти для автоматизації створення образів систем (Machine Image Building) [1]. Розглянемо чотири інструменти, які нині найчастіше застосовуються у практиці IaC: Terraform, Pulumi, AWS CloudFormation та Ansible. Кожен з них має свою нішу та особливості.

- HashiCorp Terraform. Terraform – це декларативний інструмент провіжингу, вперше випущений компанією HashiCorp у 2014 році [5]. Конфігурації Terraform описуються мовою HCL (HashiCorp Configuration Language) у вигляді файлів .tf, в яких користувач визначає бажаний кінцевий стан інфраструктури (ресурси та їх параметри). Terraform є мультихмарним та підтримує сотні провайдерів: з його допомогою можна створювати ресурси в AWS, Azure, GCP, Oracle Cloud, OpenStack та навіть сторонні сервіси (GitHub, Kubernetes тощо) – завдяки бібліотеці провайдерів і модулів.
- Pulumi. Pulumi – відносно новий інструмент (перший випуск у 2018 році), що пропонує інший підхід до IaC [5]. Замість декларативної DSL на зразок HCL, Pulumi дозволяє описувати інфраструктуру звичайними мовами програмування (TypeScript, Go, JavaScript, Python, C# тощо). Таким чином, розробники можуть використовувати звичні їм мови, бібліотеки і патерни (цикли, умови, функції) для визначення інфраструктури.
- AWS CloudFormation. CloudFormation – це пропрієтарний сервіс IaC від Amazon Web Services, запущений у 2011 році [5]. Він дозволяє описати інфраструктуру в AWS у вигляді шаблонів (JSON або YAML), які включають ресурси AWS та їх параметри. На відміну від Terraform чи Pulumi, CloudFormation є специфічним для AWS: він не підтримує інші хмари.
- Ansible. Ansible (розробник – Red Hat) – це інструмент конфігураційного управління, вперше випущений у 2012 році [6]. На відміну від попередніх трьох, Ansible за своєю природою є імперативним: користувач пише сценарії (playbooks) у форматі YAML, де покроково визначає завдання (tasks), які потрібно виконати на цільових вузлах. Ansible не створює інфраструктуру «з нуля» у хмарі, але може підключатися до існуючих серверів (через SSH на Linux або WinRM на Windows) і виконувати на них команди: встановлювати пакети, копіювати файли конфігурації, змінювати налаштування ОС чи ПЗ тощо. Таким чином, Ansible часто використовується в комбінації з Terraform/CloudFormation: спочатку Terraform розгортає, наприклад, 10 VM, а потім Ansible налаштовує на них потрібне програмне середовище [6]. У цілому, Terraform та Ansible часто розглядають не як взаємовиключні, а як доповнюючі інструменти: Terraform забезпечує базове розгортання ресурсів, а Ansible – гнучке налаштування сервісів на цих ресурсах.

Порівняльна характеристика інструментів IaC. Попри різні підходи, наведені вище інструменти мають спільну мету – автоматизувати управління інфраструктурою – але відрізняються за рядом ключових характеристик. У табл. 1 узагальнено порівняння Terraform, Pulumi, CloudFormation та Ansible за важливими критеріями.

Ефективність IaC за ключовими метриками. Оцінимо, як застосування методології «Інфраструктура як Код» впливає на основні показники ефективності управління інфраструктурою: автоматизацію, масштабованість, час (швидкість) розгортання, гнучкість, надійність та спрощення управління конфігураціями. Для кожної з цих метрик порівняємо традиційний підхід (ручне керування інфраструктурою або скрипти ad-hoc) з IaC-підходом.

Таблиця 1 – Порівняння інструментів IaC

Критерій	Terraform	Pulumi	AWS CloudFormation	Ansible
Відкритість	Так (OSS проект)	Так (OSS проект)	Ні (сервіс AWS)	Так (OSS проект)
Хмарна незалежність	Так, мультихмарний	Так, мультихмарний	Ні, лише AWS	Так, мультихмарний
Підхід до опису	Декларативний (HCL)	Програмний (TypeScript, Python та ін.)	Декларативний (YAML/JSON)	Імперативний (YAML playbooks)
Призначення	Провіжинг інфраструктури (IaaS/PaaS)	Провіжинг інфраструктури (кодом)	Провіжинг AWS-ресурсів	Конфігурація серверів, встановлення ПЗ
Управління станом	Так (стан зберігається в файлі/сховищі)	Так (стан локально або в Pulumi Cloud)	Так (стан відстежує AWS)	Ні (без централіз. стану)
Поріг входження	Середній	Середній (потрібні навички програмування)	Низький для AWS-спеціалістів	Низький (знайомий формат YAML)
Спільнота та екосистема	Дуже широка, багато модулів	Помірна, зростає	Вузька	Дуже широка (Ansible Galaxy тощо)

- Автоматизація. IaC майже повністю усуває ручні дії при розгортанні і зміні інфраструктури. У традиційному підході побудова нового середовища вимагала послідовного виконання багатьох ручних операцій (замовити сервери або ВМ, встановити ОС, налаштувати параметри, мережі, брандмауери, встановити необхідне ПЗ тощо). З IaC ці кроки кодуються один раз у вигляді сценарію або шаблону; далі будь-яке нове розгортання зводиться до запуску цього коду. Рівень автоматизації, що приносить IaC, дозволяє швидше і частіше виконувати зміни в інфраструктурі. За даними галузевих досліджень, використання IaC збільшує пропускну здатність деплойменту (throughput): команди можуть робити релізи і зміни інфраструктури більш регулярно, не витрачаючи час на рутинні налаштування [4]. Крім того, автоматизація усуває людські помилки при налаштуванні – всі зміни стандартнізовані і повторювані, що підвищує якість. Практика показує, що автоматизація через IaC також сприяє впровадженню DevOps-практик CI/CD для інфраструктури: зміни в коді інфраструктури можуть автоматично тестуватися і розгортатися через конвеєри, забезпечуючи безперервне доставлення змін. В результаті час на розгортання нових ресурсів чи середовищ скорочується з годин/днів до хвилин. Наприклад, увімкнення нового сервера вручну може займати години, включно з налаштуваннями, тоді як IaC-скрипт виконає те саме за кілька хвилин без участі людини.
- Масштабованість. Ручне керування інфраструктурою погано масштабується: адміністратору важко одночасно та безпомилково налаштувати десятки або сотні вузлів. IaC радикально покращує цей показник. Оскільки інфраструктура описана як код, збільшення масштабу часто зводиться до незначних змін у параметрах (наприклад, збільшити кількість екземплярів сервісу з 5 до 50) і повторного застосування коду. Інструмент IaC автоматично створить потрібну кількість ресурсів за тим самим шаблоном [2]. Таким чином, горизонтальне масштабування (тиражування однакових вузлів) здійснюється легко і послідовно. Без IaC розгортання додаткових 45 серверів потребувало б 45 раз виконати ті самі дії, що надзвичайно перевантажує команди і майже напевно призводить до відхилень у конфігурації. З IaC – достатньо один раз написати код сервера, після чого 5, або 50 серверів розгорнуться однаково [2]. Це дозволяє обслуговувати більші навантаження та швидко реагувати на пікові запити. Як зазначається в літературі, масштабування інфраструктури за IaC здійснюється просто через модифікацію коду і запуск інструмента, що економить час і зусилля та гарантує, що інфраструктура масштабується передбачувано і надійно [2]. Більш того, IaC спрощує впровадження авто-масштабування: наприклад, у хмарі можна заздалегідь описати політики автоматичного додавання вузлів при досягненні певних метрик (CPU, трафік), і платформа (або спеціальний скрипт) сама створить нові екземпляри за шаблоном IaC. В цілому, IaC робить можливим управління великими інфраструктурами невеликими командами, адже більшість операцій не залежить від об'єму – інструмент так само виконає 100 завдань, як і 10, відрізняючись лише тривалістю виконання і використаними ресурсами.
- Час розгортання і частота змін. Швидкість, з якою можна підготувати нове оточення або додати зміни, є критичною метрикою, особливо в Agile та DevOps циклах розробки. Від використання IaC час розгортання значно виграє. По-перше, завдяки автоматизації зникають затримки між кроками, пов'язані з очікуванням дій людини – інструмент виконує все максимально швидко і паралельно, де можливо. По-друге, IaC сприяє паралельній роботі над інфраструктурою: кілька інженерів можуть одночасно оновлювати код різних компонент в системі контролю версій, потім об'єднувати зміни, що швидше, ніж коли одна людина послідовно налаштовує компоненти. Дослідження DORA (DevOps Research and Assessment) демонструють, що команди, які впровадили IaC, досягають більшої частоти деплойменту і коротшого часу поставки змін без втрати стабільності [4]. Це можливо тому, що IaC дозволяє автоматизувати навіть тес-

тування інфраструктури: можна підняти тимчасове середовище за тим же кодом, прогнати на ньому тестування, і згорнути – все це за лічені години, тоді як раніше на налаштування стейджингу могли йти дні. Отже, програмний опис інфраструктури збільшує швидкість всього циклу «проектування–розгортання–перевірка». Важливим є і зменшення Mean Time to Recovery (MTTR) – середнього часу відновлення після збою. Якщо певний компонент інфраструктури вийшов з ладу, наявність скриптів IaC дозволяє швидко розгорнути його копію або перевести трафік на новий екземпляр з того ж шаблону, значно скоротивши час простою [4].

- Гнучкість змін (адаптивність). Під гнучкістю мається на увазі здатність легко вносити зміни в інфраструктуру у відповідь на нові вимоги або експерименти. Традиційна інфраструктура відома своєю інертністю – будь-яка зміна потребує обережного планування, часто – ручного виконання і, відповідно, несе ризики. IaC робить інфраструктуру набагато більш гнучкою. Оскільки конфігурація – це код, розробники можуть застосовувати практики розробки ПЗ: створювати окремі гілки (branch) для експериментів з інфраструктурою, тестувати їх у ізольованих середовищах, робити code review змін конфігурацій тощо [2]. Якщо виникає потреба спробувати нову топологію мережі чи іншу версію сервісу, достатньо ввести зміни у код і розгорнути новий стенд автоматично. За наявності хмарних ресурсів це займає мінімум часу і не впливає на основне (продуктивне) середовище. Таким чином, IaC сприяє впровадженню інфраструктурних змін iteratively: можна часто і малими кроками оновлювати систему, замість рідких масштабних реорганізацій. Крім того, модульність IaC (як-от використання модулів Terraform чи ролей Ansible) дозволяє гнучко перевикористовувати компоненти: наприклад, підключити до існуючої архітектури новий модуль моніторингу, імпортувавши готовий код, і таким чином швидко додати новий функціонал. Відповідно до принципу Don’t Repeat Yourself (DRY) [7], модульний код IaC знижує зусилля при змінах: правка в одному модулі (скажімо, змінити тип машин для шару баз даних) автоматично застосується всюди, де цей модуль використовується [2]. Експерименти з інфраструктурою також полегшуються – інженери можуть швидко «виводити» окремі тестові середовища, пробувати різні конфігурації, а потім так само швидко їх видаляти, не залишаючи «сміття», адже все створене контролюється кодом і може бути знищене командою destroy. Це додає гнучкості у процес архітектурного дизайну і оптимізації, дозволяючи знаходити кращі рішення.
- Надійність та стабільність. Надійність інфраструктури багато в чому визначається послідовністю конфігурацій та можливістю швидкого відновлення після збоїв [2]. IaC значно підвищує консистентність середовищ: середовище, розгорнуте за одним і тим же кодом, буде кожного разу ідентичним. Це означає передбачувану роботу застосунків при перенесенні з одного середовища в інше. Крім того, усувається проблема configuration drift – коли з часом сервери «роз’їжджаються» у налаштуваннях через точкові ручні зміни [8]. З IaC усі зміни вносяться через код і одразу застосовуються до всіх відповідних вузлів, або ж можна регулярно перезапускати застосування IaC-конфігурацій, щоб гарантувати відповідність бажаному стану. Як наслідок, зменшується кількість збоїв, спричинених некоректними або непослідовними конфігураціями. Статистика показує, що запровадження повністю автоматизованих конвеєрів (включно з IaC) веде до зниження частки невдалих змін на продуктиві (Change Failure Rate)[9], оскільки проблеми виявляються раніше (на етапі автоматичних тестів, статичного аналізу коду інфраструктури тощо) або не виникають взагалі через однаковість середовищ [4]. Надійність також проявляється у можливості швидкого відтворення оточення з нуля. Якщо, наприклад, сталася серйозна аварія, що знищила частину інфраструктури, наявність опису IaC дозволяє натиснути «умовну кнопку» і розгорнути все заново в резервному датацентрі або хмарі, з мінімальними втратами даних. Це кардинально відрізняється від підходу, де відновлення залежить від резервних копій конфігурацій, документів і пам’яті адміністраторів. На додачу, IaC сприяє надійності за рахунок інтеграції з практиками каскадного розгортання та відкату: можна реалізувати шаблони blue-green deployment [10] або canary release [11] для інфраструктури, коли нове середовище розгортається паралельно і переключення відбувається лише після перевірки, з можливістю швидкого відкату на попередню версію середовища – і все це автоматично через код.
- Спрощення управління конфігураціями. IaC радикально змінює процес керування конфігураціями, роблячи його більш системним і підконтрольним. У традиційному підході, після внесення змін, адміністраторам доводиться вручну документувати новий стан систем (часто в wiki, таблицях), відслідковувати хто і коли змінював налаштування, забезпечувати синхронізацію цих знань між колегами. Це трудомістко і схильне до помилок – документацію можуть забути оновити, або вона не охоплює всіх нюансів. З IaC документування і управління змінами відбувається автоматично, як побічний продукт використання версіонованого коду. Кожна зміна фіксується у системі контролю версій (Git тощо) із зазначенням автора, часу і вмісту – тому завжди можна відповісти, хто змінив, наприклад, параметр масштабування кластера і чому. Це забезпечує повний аудит змін і прозорість процесів [12]. Більше не потрібно вести окремий журнал вручну, бо історія змін «живе» в репозиторії, а для критичних середовищ можна налаштувати політики затвердження (pull requests, код-рев’ю) перед злиттям конфігурацій, що еквівалентно процесу управління змінами, але в більш структурованій формі. Щодо актуальності інформації про інфраструктуру: IaC дозволяє завжди отримати актуальний стан через самі інструменти. Наприклад, Terraform зберігає state-файл, де зазначені всі ресурси і їхні атрибути, – це по суті автоматично оновлюваний «список конфігурацій» системи [12]. В будь-який момент можна проглянути цей стан або запитати реальну інфраструктуру (наприклад, команда terraform plan покаже, що змінено вручну відносно ко-

ду). Це протиставляється практиці ведення таблиць з налаштуваннями вручну, коли висока ймовірність розсинхронізації.

Для наочності, у табл. 2 підсумовано вплив застосування IaC на кожну із перелічених метрик ефективності у порівнянні з традиційним підходом.

Висновки. Методологія «Інфраструктура як Код» (IaC) демонструє високу ефективність у створенні, масштабуванні та управлінні хмарною інфраструктурою. Застосування IaC дозволяє автоматизувати ключові процеси, скоротити час розгортання, покращити повторюваність і надійність середовищ, а також підвищити гнучкість та контрольованість змін. Проведений аналіз показав, що такі інструменти, як Terraform, Pulumi, CloudFormation та Ansible, мають свої сильні сторони і можуть адаптуватися до різних хмарних платформ.

IaC стає невід'ємною частиною DevOps-практик і суттєво знижує витрати на обслуговування інфраструктури. За умови впровадження кращих практик, IaC сприяє створенню стабільного, керованого та стандартизованого IT-середовища, що відповідає вимогам сучасного бізнесу.

Таблиця 2 – Ефективність застосування IaC

Метрика	Традиційний підхід (без IaC)	Підхід IaC (Інфраструктура як Код)
Автоматизація	Багато ручних кроків при налаштуванні, ризик помилок; скрипти ad-hoc частково допомагають, але важко підтримуються	Повна автоматизація через код, мінімум ручної роботи; стандартизовані процеси розгортання виконуються інструментами IaC без відхилень
Масштабованість	Масштабування вимагає пропорційно більше зусиль (лінійно росте з кількістю серверів); висока ймовірність неконсистентності при великій кількості вузлів	Легке тиражування конфігурацій на десятки і сотні ресурсів; консистентність зберігається незалежно від кількості, автоматична паралелізація розгортання
Час розгортання	Повільне розгортання нових середовищ (години/дні); затримки між етапами через ручні дії, довгий цикл доставки змін	Швидке розгортання (хвилини/години) за рахунок автоматизації; можливість частих і малих деплойментів, прискорене доставлення функцій на ринок
Гнучкість змін	Обережне внесення змін, страх порушити стабільність; експерименти затратні, потребують окремих ресурсів і часу	Висока гнучкість: зміни вносяться у код і перевіряються автоматично; можна створювати ізольовані тестові середовища за потреби, швидко пробувати нові конфігурації і компоненти
Надійність	Залежність від людського фактору: можливі помилки конфігурації, розбіжності між середовищами; тривале відновлення після збоїв, складність підтримки однакового стану	Підвищена надійність завдяки узгодженим конфігураціям (код єдиного зразка для всіх середовищ); швидке відновлення/відтворення інфраструктури з кодом, менше збоїв через дрейф чи помилки; вбудовані механізми відкату і тестування знижують ризики невдалих змін
Управління конфігураціями	Ручне документування і відстеження змін (таблиці); можливі пропуски в документації, «трайбові» знання у головах команди	Керування конфігураціями значно спрощене: актуальна документація – це сам код; всі зміни фіксуються системою контролю версій, стан інфраструктури можна отримати автоматично; прозорий аудит та контроль версій забезпечують відповідність конфігурацій політикам

Список використаної літератури

1. Pandu Ranga Reddy Konala, Vimal Kumar, David Bainbridge, Junaid Haseeb. A Framework for Measuring the Quality of Infrastructure-as-Code Scripts. URL: https://arxiv.org/html/2502.03127v1 (дата звернення: 30.01.2025).
2. Codefresh.io. Infrastructure as Code in Devops: 7 Steps to Success. *Codefresh Learning Center*. URL: https://codefresh.io/learn/infrastructure-as-code/infrastructure-as-code-in-devops-7-steps-to-success/ (дата звернення: 14.03.2025).
3. Mariusz Michalowski. Benefits and Best Practices for Infrastructure as Code. *DevOps.com.* URL: https://devops.com/benefits-and-best-practices-for-infrastructure-as-code (дата звернення: 14.03.2025).
4. Ned Bellavance. DORA Metrics: An Infrastructure as Code Perspective. *Env0 Blog*. URL: https://www.env0.com/blog/dora-metrics-an-infrastructure-as-code-perspective (дата звернення: 11.02.2025).
5. Bunnyshell team. Terraform vs. Cloudformation vs. Pulumi. *Bunnyshell blog.* URL: https://www.bunnyshell.com/blog/terraform-vs.-cloudformation-vs.-pulumi (дата звернення: 22.02.2025).
6. Pavan Gumaste. Terraform vs CloudFormation vs Ansible. *Whizlabs. Blog*. URL: https://www.whizlabs.com/blog/terraform-vs-cloudformation-vs-ansible (дата звернення: 10.03.2025).
7. Daniel Poppy. DRY (Don't Repeat Yourself) — Programming Practices. *Online publishing platform medium.com.* URL: https://medium.com/@vaibhavmojidra/dry-dont-repeat-yourself-programming-practices-a43e2318f2c4 (дата звернення: 15.03.2025).
8. Ioannis Moustakis. What is Configuration Drift? Tools, Causes & Risks. *Spacelift.io blog*. URL: https://spacelift.io/blog/what-is-configuration-drift (дата звернення: 15.03.2025).
9 Stephen J. Bigelow. What is change failure rate (CFR)? *Techtarget blog.* https://www.techtarget.com/searchitoperations/definition/change-failure-rate (дата звернення: 18.03.2025).
10. Octopus.com. The DevOps engineer's handbook. Blue/green deployments: how they work, pros and cons, and 8 critical best practices. *The DevOps engineer's handbook.* URL: https://octopus.com/devops/software-deployments/blue-green-deployment/ (дата звернення: 06.04.2025).

11. Octopus.com. Canary deployments: Pros, cons, and 5 critical best practices. *The DevOps engineer's handbook.* URL: https://octopus.com/devops/software-deployments/blue-green-deployment/ (дата звернення: 06.04.2025).
12. Puppet.com blog. What is Infrastructure as Code (IaC)? Best Practices, Tools, Examples & Why Every Organization Should Be Using It. *Puppet.com blog*. URL: https://www.puppet.com/blog/what-is-infrastructure-as-code (дата звернення: 16.03.2025).

References (transliterated)

1. Pandu Ranga Reddy Konala, Vimal Kumar, David Bainbridge, Junaid Haseeb. A Framework for Measuring the Quality of Infrastructure-as-Code Scripts. Available at: https://arxiv.org/html/2502.03127v1 (accessed: 30.01.2025).
2. Codefresh.io. Infrastructure as Code in Devops: 7 Steps to Success. *Codefresh Learning Center*. Available at: https://codefresh.io/learn/infrastructure-as-code/infrastructure-as-code-in-devops-7-steps-to-success/ (accessed: 14.03.2025).
3. Mariusz Michalowski. Benefits and Best Practices for Infrastructure as Code. *DevOps.com.* Available at: https://devops.com/benefits-and-best-practices-for-infrastructure-as-code (accessed: 14.03.2025).
4. Ned Bellavance. DORA Metrics: An Infrastructure as Code Perspective. *Env0 Blog*. Available at: https://www.env0.com/blog/dora-metrics-an-infrastructure-as-code-perspective (accessed: 11.02.2025).
5. Bunnyshell team. Terraform vs. Cloudformation vs. Pulumi. *Bunnyshell blog.* Available at: https://www.bunnyshell.com/blog/terraform-vs.-cloudformation-vs.-pulumi (accessed: 22.02.2025).
6. Pavan Gumaste. Terraform vs CloudFormation vs Ansible. *Whizlabs. Blog*. Available at: https://www.whizlabs.com/blog/terraform-vs-cloudformation-vs-ansible (accessed: 10.03.2025).
7. Daniel Poppy. DRY (Don't Repeat Yourself) — Programming Practices. *Online publishing platform medium.com.* Available at: https://medium.com/@vaibhavmojidra/dry-dont-repeat-yourself-programming-practices-a43e2318f2c4 (accessed: 15.03.2025).
8. Ioannis Moustakis. What is Configuration Drift? Tools, Causes & Risks. *Spacelift.io blog*. Available at: https://spacelift.io/blog/what-is-configuration-drift (accessed: 15.03.2025).
9 Stephen J. Bigelow. What is change failure rate (CFR)? *Techtarget blog.* https://www.techtarget.com/searchitoperations/definition/change-failure-rate (дата звернення: 18.03.2025).
10. Octopus.com. The DevOps engineer's handbook. Blue/green deployments: how they work, pros and cons, and 8 critical best practices. *The DevOps engineer's handbook.* Available at: https://octopus.com/devops/software-deployments/blue-green-deployment/ (accessed: 06.04.2025).
11. Octopus.com. The DevOps engineer's handbook. Canary deployments: Pros, cons, and 5 critical best practices. *The DevOps engineer's handbook.* Available at: https://octopus.com/devops/software-deployments/blue-green-deployment/ (accessed: 06.04.2025).
12. Puppet.com blog. What is Infrastructure as Code (IaC)? Best Practices, Tools, Examples & Why Every Organization Should Be Using It. *Puppet.com blog.* Available at: https://www.puppet.com/blog/what-is-infrastructure-as-code (accessed: 16.03.2025).

Надійшла (received) 14.05.2025

UDC 004.02

D. O. MIROSHNYCHENKO, Postgraduate student, V. N. Karazin Kharkiv National University, Kharkiv, Ukraine; e-mail: dmytro.miroshnychenko@student.karazin.ua; ORCID: https://orcid.org/0000-0003-1159-0985
O. H. TOLSTOLUZKA, Doctor of Technical Sciences, Senior Research Fellow, Professor of the Department of Computer Science and Robotics in V. N. Karazin Kharkiv National University, Kharkiv, Ukraine; e mail: elena.tolstoluzka@karazin.ua; ORCID: https://orcid.org/0000-0003-1241-7906

EVALUATION OF THE EFFECTIVENESS OF THE "INFRASTRUCTURE AS CODE" METHODOLOGY FOR CREATING AND MANAGING CLOUD INFRASTRUCTURE

The article describes a comprehensive study of the effectiveness of using the Infrastructure as Code (IaC) methodology to create, scale, and manage cloud infrastructure. The IaC methodology is considered one of the key technologies of digital transformation and the DevOps approach, which provides software automation of infrastructure processes, reduces dependence on the human factor, and increases the repeatability and predictability of IT environments. The article provides a comparative analysis of leading IaC implementation tools, in particular Terraform, Pulumi, AWS CloudFormation, and Ansible, from the standpoint of their openness, compatibility with various cloud platforms, architectural approach (declarative or imperative), state management, and level of flexibility. The degree of automation, scalability, speed of infrastructure deployment, adaptability to change, configuration reliability, and ease of management are evaluated as key performance metrics. For each metric, a theoretical justification, analytical assessment, and comparison with traditional approaches to administration are provided. Special attention is paid to the analysis of IaC implementation in leading cloud environments (AWS, Microsoft Azure, Google Cloud Platform, OpenStack) taking into account the corresponding platform solutions (CloudFormation, ARM/Bicep, Deployment Manager, Heat) and third-party multi-cloud tools. It was found that the use of IaC significantly improves DevOps practices, simplifies CI/CD processes and increases the reliability of cloud solutions. As a result, it is proven that the use of IaC provides a significant increase in operational efficiency, reduces infrastructure maintenance costs and promotes its standardization, which makes this methodology strategically important for modern IT systems..

Keywords: infrastructure as code, cloud computing, Terraform, Pulumi, DevOps, CloudFormation, efficiency, automation, configuration management, CI/CD.

Повні імена авторів / Author's full names

Автор 1 / Author 1: Мірошниченко Дмитро Олександрович / Miroshnychenko Dmytro Oleksandrovych
Автор 2 / Author 2: Толстолузька Олена Геннадіївна / Tolstoluzka Olena Hennadiivna