УДК 004.056:004.7

**А. В. Кондратюк**
**Д. П. Присяжний**

# МОДЕЛІ ТА МЕТОДИ ПІДВИЩЕННЯ БЕЗПЕКИ INFRASTRUCTURE AS CODE У ХМАРНИХ СЕРЕДОВИЩАХ.

Вінницький національний технічний університет

***Анотація***
*У статті досліджується проблематика безпеки контейнерних хмарних інфраструктур, побудованих за парадигмою Infrastructure as Code (IaC) з використанням інструментів Terraform, Kubernetes та Helm. Проаналізовано ключові вектори загроз, зокрема помилки конфігурації (misconfiguration), ескалацію привілеїв та інфраструктурний дрейф (drift). Обґрунтовано необхідність переходу до автоматизованої моделі захисту через впровадження концепції Policy-as-Code (PaC) та парадигми «Shift-Left». Запропоновано комплексну модель безпечного розгортання (secure rollout) змін на базі CI/CD та GitOps, що дозволяє нівелювати людський фактор, жорстко блокувати вразливості на етапі написання коду та автоматично підтримувати еталонний, безпечний стан інфраструктури під час її експлуатації.*
***Ключові слова:*** *Infrastructure as Code (IaC), багатохмарні середовища, Terraform, Kubernetes, Helm, Policy-as-Code, ескалація привілеїв, інфраструктурний дрейф, безпечне розгортання, GitOps, сценарій, CI/CD, помилки конфігурацій, Shift-Left.*

***Abstract***
*This paper investigates the security challenges of multi-cloud infrastructures built on the Infrastructure as Code (IaC) paradigm using Terraform, Kubernetes, and Helm. The study analyzes key threat vectors, including configuration errors (misconfigurations), privilege escalation, and infrastructure drift. It substantiates the necessity of transitioning to an automated security model by implementing the Policy-as-Code (PaC) concept and the "Shift-Left" paradigm. A comprehensive model for the secure rollout of changes based on CI/CD pipelines and GitOps is proposed. This approach mitigates human error, strictly blocks vulnerabilities at the coding stage, and automatically maintains the baseline, secure state of the infrastructure during runtime operations.*
***Keywords:*** *Infrastructure as Code (IaC), multi-cloud environments, Terraform, Kubernetes, Helm, Policy-as-Code, privilege escalation, infrastructure drift, secure rollout, GitOps, scenario, CI/CD, misconfigurations, Shift-Left.*

## Вступ

Еволюція сучасних IT-інфраструктур здебільшого має ознаки переходу до контейнерних хмарних (multi-cloud) середовищ. Правильний розподіл обчислювальних потужностей між провайдерами дозволяє компаніям досягати високої відмовостійкості. Управління такими масштабами стало можливим завдяки зміні з ручного управління на Infrastructure as Code (IaC) [1]. Інструменти розгортання хмарних ресурсів (Terraform), контейнерів (Kubernetes) та пакетного управління застосунками (Helm) стали базовим стандартом у використанні. Однак, абстрагування апаратного забезпечення та перенесення архітектури дата-центрів у площину програмного коду створило новий, безпрецедентний за масштабами вектор загроз. Традиційні проблеми розробки програмного забезпечення автоматично перенеслися на рівень базової інфраструктури.

З огляду на зазначену проблему, формується методологічний апарат даної роботи:

– Мета дослідження: підвищення безпеки multi-cloud середовищ шляхом формалізації моделі контролю конфігурацій IaC;

– Об’єкт дослідження: процеси розгортання та експлуатації контейнерних хмарних інфраструктур;

– Предмет дослідження: методи виявлення й попередження помилок конфігурації в Terraform, Kubernetes і Helm;

– Використані методи: структурний аналіз, порівняльний аналіз, моделювання, експертна оцінка, аналіз політик безпеки.

## Результати дослідження

### Архітектурна модель Infrastructure as Code у контейнерних хмарних середовищах

Основою сучасних IT-систем є парадигма Infrastructure as Code (IaC), яка передбачає заміну ручного управління серверами на написання коду розгортання. Фундаментальними властивостями IaC є декларативність - інженер описує бажаний кінцевий результат, а не кроки для його досягнення, та ідемпотентність - гарантія того, що багаторазове виконання одного й того ж коду призведе до однакового стану системи без дублювання ресурсів).

У складних хмарних середовищах, де компанії одночасно використовують обчислювальні потужності AWS, GCP та Azure для уникнення прив'язки до одного постачальника хмарних сервісів (vendor lock-in), розгортання інфраструктури відбувається на трьох взаємопов'язаних рівнях абстракції:

1. Terraform – Розгортання ресурсів базової хмари. Цей інструмент дозволяє описувати інфраструктуру єдиною мовою HashiCorp Configuration Language (HCL), яка слугує засобом керування API будь-якого хмарного провайдера. На цьому етапі код звертається до хмарного сервісу та створює фундаментальні ресурси: віртуальні мережі (VPC), бази даних (DB) та пули віртуальних машин [2].

2. Kubernetes – Управління. На серверах, створених за допомогою Terraform, розгортається кластер Kubernetes (K8S) [3]. Він виконує роль "операційної системи" кластера, яка абстрагує різницю між хмарами. K8S об'єднує розрізнені сервери та перетворює їх на єдиний пул обчислювальних ресурсів для запуску контейнерів.

3. Helm – Менеджер пакетів для управління застосунками. Оскільки розгортання застосунків у Kubernetes вимагає написання великого масиву YAML-маніфестів, використовується пакетний менеджер Helm. Він використовує стандартизовані шаблони (chart), дозволяючи зручно керувати конфігураціями через змінні [4].

Логічний ланцюг: Terraform створює фізичну/віртуальну хмару - K8S керує серверами в цій хмарі - Helm розгортає застосунки всередині K8S.

### Вектори загроз інфраструктурного коду: Misconfigurations, Privilege Escalation, Drift, Policy-as-code, Rollout

Перенесення інфраструктури у формат коду масштабує не лише швидкість роботи, але й ризики. Основною проблемою стають помилки налаштування (Misconfiguration) – логічні помилки або недбалість під час написання коду. Для системного розуміння цього вектора загроз, помилки конфігурації в IaC доцільно класифікувати за трьома основними напрямками:

1. Мережеві помилки (Network security) залишені відкритими порти, несанкціонований публічний доступ до приватних сховищ (наприклад, відкрите сховище S3 в AWS) та слабка сегментація мережі [7]. У Kubernetes це часто проявляється у вигляді "пласкої мережі", де відсутні мережеві політики ізоляції, що дозволяє всім сервісам вільно взаємодіяти між собою.

2. Ідентифікація та доступ (Identity & Access) надлишкові права у хмарних провайдерів або кластерах (IAM, RBAC). Використання небезпечних сервісних акаунтів із надмірними привілеями, відкриває зловмисникам легкий шлях до контролю над інфраструктурою.

3. Безпека середовища виконання (Runtime / Workload security) стосується безпосередньо розгорнутих застосунків. Включає запуск контейнерів від імені root-користувача, надання контейнерам небезпечних системних привілеїв, відсутність попереднього сканування образів на вразливості, а також ігнорування обмежень ресурсів, що створює ризик внутрішніх атак відмови в обслуговуванні.

Якщо при ручному адмініструванні помилка стосувалася одного сервера, то в парадигмі IaC помилковий шаблон автоматично розгортається по всім серверам в хмарних середовищах, критично збільшуючи площу атаки.

Помилки конфігурації стають початковою точкою для ескалації привілеїв (Privilege Escalation) – вектора атаки, за якого зловмисник, отримавши мінімальний доступ, використовує недоліки архітектури для здобуття прав адміністратора. Хакер проникає в контейнер через дрібну вразливість застосунку. Через відсутність мережевих політик та RBAC-обмежень, зловмисник звертається до внутрішнього API K8S, викрадає токен сервісного акаунта і отримує підвищені права. Таким чином, злам одного дрібного контейнера дає контроль над усім хмарним середовищем компанії [3].

Додатковим фактором хаосу є інфраструктурний дрейф (Drift) – розбіжність між задекларованим кодом у репозиторії та фактичним станом хмари. Якщо під час інциденту інженер вручну відкриває порт, у хмарі з'являється вразливість яку важко ідентифікувати системам сканування. Усі ручні

налаштування під час наступного автоматичного оновлення (rollout) Terraform будуть видалені, що призведе до падіння продукту.

Запобіжником у цій системі працює Policy-as-Code (PaC) – який працює як написання правил безпеки мовою програмування [5]. Спосіб навчити автоматизовані системи самостійно відрізняти безпечний та робочий код від вразливого. Завдяки цьому підходу перевірка відповідності відбувається без участі людини, щоб не сподіватися на інженера під час перегляду коду (code review). Це являється прямою протидією misconfiguration та privilege escalation. Наприклад, контейнер у кластері не може бути запущений від імені root-користувача, або коли хтось намагається використати YAML маніфест який порушує правила. Рушій політики блокує це на рівні API Kubernetes під час CI/CD конвеєру. PaC перетворює паперові інструкції на фізичний бар'єр який неможливо обійти.

Rollout – це процес розгортання написаного коду інженером в робоче середовище (хмару чи кластер). Безпечне розгортання через парадигми CI/CD та GitOps означає, що жодна внесена зміна не потрапляє в роботу без проходження багаторівневих машинних перевірок на безпеку.

**Емпіричний аналіз, моделювання типових помилок конфігурації та їх нейтралізація засобами Policy-as-Code**

Для підтвердження ефективності проактивної моделі безпеки доцільно розглянути емпіричні сценарії типових помилок на різних рівнях абстракції інфраструктури. Наведені нижче сценарії демонструють, як саме парадигма Policy-as-Code, із застосуванням таких інструментів, як Open Policy Agent (OPA) або Checkov, фізично блокує вектори атак до моменту їх розгортання [8].

Сценарій 1 (Мережевий рівень) – Публічний доступ до хмарного сховища даних.

Інженер під час написання коду Terraform для розгортання AWS S3 залишає параметр списку контролю доступу (ACL) у значенні public-read або забуває увімкнути примусове блокування публічного доступу.

Вектор атаки. Несанкціоноване вивантаження конфіденційних даних компанії автоматизованими сканерами, що безперервно шукають відкриті сховища в інтернеті.

Нейтралізація: інструмент статичного аналізу (наприклад, Checkov), інтегрований у конвеєр, містить базову політику відповідності (наприклад, CKV_AWS_20). Під час сканування коду система виявляє відсутність блокування публічного доступу і перериває процес злиття коду, вказуючи інженеру на необхідність додати параметр block_public_acls = true.

Сценарій 2 (Рівень Runtime) – Запуск привілейованого контейнера в Kubernetes.

У YAML-маніфесті застосунку, в Helm-чарті розробник для спрощення налаштувань мережі або доступу до файлової системи встановлює параметр securityContext: privileged: true.

Вектор атаки. Контейнер отримує прямий доступ до ядра Linux на вузлі кластера. У разі компрометації застосунку хакер минає ізоляцію контейнера, здійснює ескалацію привілеїв і отримує root-доступ до фізичного/віртуального сервера.

Нейтралізація: рушій політик Kubernetes (наприклад, OPA Gatekeeper або Kyverno) працює як контролер визнання. Під час спроби застосувати маніфест через API Kubernetes, рушій перевіряє код на відповідність політиці "Pod Security Standards". Виявивши privileged: true, система відхиляє запит із помилкою доступу, унеможливлюючи запуск контейнера.

Сценарій 3 (Рівень Identity & Access) – Надлишкові дозволи хмарних ролей.

У коді Terraform створюється політика доступу (наприклад в AWS IAM), де для спрощення розробки вказано параметри Action: "*" та Resource: "*".

Вектор атаки. Сервіс або контейнер, що використовує цю роль, отримує адміністративний доступ до всього хмарного акаунта. У разі зламу цього сервісу зловмисник може видаляти бази даних, створювати нові, шкідливі ресурси або змінювати конфігурації безпеки.

Нейтралізація: політика PaC забороняє використання символів узагальнення (wildcards) у блоках дій та ресурсів. Аналізатор блокує розгортання, вимагаючи від інженера дотриматися принципу найменших привілеїв (PoLP) і чітко вказати конкретні дії для конкретного ресурсу.

Сценарій 4 (Рівень стабільності) – Відсутність квот на споживання ресурсів.

Під час розгортання мікросервісу через Helm у конфігурації розгортання не вказані квоти (resources.limits - обмеження CPU та оперативної пам'яті).

Вектор атаки. У разі внутрішньої помилки програми (наприклад, витоку пам'яті) або DDoS-атаки контейнер починає безконтрольно споживати ресурси сервера. Це призводить до стану OOM (Out of Memory) на вузлі та відмови в обслуговуванні інших критичних сервісів кластера (ефект галасливого сусіда).

Нейтралізація: політика PaC на рівні кластера вимагає обов'язкової наявності блоків requests та limits у кожному маніфесті. Код без цих параметрів автоматично відхиляється контролером визнання.

Завдяки впровадженню описаних PaC-сценаріїв інфраструктурний конвеєр перетворюється з простого інструмента доставки на автоматизовану систему управління ризиками, що превентивно блокує найбільш критичні вектори атак.

**Безпечне розгортання змін та управління життєвим циклом інфраструктури через GitOps**

Наявність задекларованих правил Policy-as-Code (PaC) вирішує проблему безпеки лише теоретично. Для їх практичного застосування необхідний механізм, який фізично не дозволить вразливому коду потрапити у хмарне середовище. Цю функцію виконує процес Безпечного розгортання (Secure Rollout) – автоматизований конвеєр доставки інфраструктурного коду (CI/CD), який виключає можливість розгортання змін без проходження багаторівневих перевірок.

В основі безпечного розгортання лежить парадигма "Shift-Left" (зсув безпеки вліво). Вона передбачає, що перевірка коду Terraform або маніфестів Kubernetes відбувається на найбільш ранньому етапі – у момент створення запиту на злиття коду (Pull Request), а не після його застосування. Конвеєр CI/CD виступає транспортною системою, у яку інтегровані рушії PaC: вони автоматично аналізують код на відповідність стандартам і блокують доставку коду у разі виявлення найменших відхилень.

Після успішного розгортання інфраструктури критичним завданням стає захист від людського фактора під час експлуатації, зокрема від інфраструктурного дрейфу. Для цього застосовується підхід GitOps – методологія, за якої Git-репозиторій визнається єдиним джерелом істини для всієї системи. Програмні контролери GitOps безперервно порівнюють поточний стан кластера з еталонним кодом у репозиторії. Якщо фіксується несанкціоноване ручне втручання (наприклад, адміністратор тимчасово розширив права доступу), система автоматично "зцілює" інфраструктуру, примусово повертаючи її до стану, описаного в Git, і тим самим ліквідує тіньові вразливості.

**Ідеальний безпечний процес управління multi-cloud середовищем**

Інтеграція описаних вище технологій формує замкнений, захищений життєвий цикл управління хмарною інфраструктурою, який складається з наступних кроків:

1. Ініціалізація та сканування: Інженер розробляє код Terraform або Helm-чарт і відправляє його до системи контролю версій (Git). Це ініціює запуск CI/CD конвеєра.

2. Блокування вразливостей (PaC): Під час виконання конвеєра автоматично активуються інструменти статичного аналізу та перевірки політик. Якщо у коді наявна помилка, рушій політик фіксує порушення і блокує розгортання.

3. Виправлення та розгортання: Інженер усуває виявлену вразливість. Оновлений код успішно проходить перевірку PaC і розгортається у хмарному середовищі.

4. Внутрішня ізоляція: У розгорнутому кластері Kubernetes автоматично активуються заздалегідь налаштовані мережеві політики та суворі правила RBAC, що нівелюють можливість Privilege Escalation у разі компрометації окремого контейнера [6].

5. Безперервний контроль (Drift Remediation): Після розгортання системи контролю GitOps переходять у режим постійного моніторингу. Будь-яка спроба ручного редагування конфігурацій хмари миттєво скасовується системою, гарантуючи еталонну безпеку інфраструктури.

## Висновки

У ході проведеного дослідження було проаналізовано вплив парадигми Infrastructure as Code на безпеку управління контейнерними хмарними (multi-cloud) середовищами. В якості першого практичного результату в роботі систематизовано типові помилки конфігурації для ключових інструментів розгортання: Terraform, Kubernetes та Helm. Показано, що залишені поза увагою мережеві вразливості, надлишкові IAM-права та небезпечні налаштування середовища виконання через автоматизоване масштабування створюють ідеальні умови для ескалації привілеїв та компрометації цілих хмарних облікових записів.

Для ефективної протидії виявленим загрозам у дослідженні розроблено та обґрунтовано узагальнену модель контролю безпеки IaC у multi-cloud середовищах. Оскільки традиційні методи ручного аудиту визнані недостатніми, в основу моделі покладено перехід до проактивної автоматизованої безпеки. У роботі показано, що інтегроване поєднання концепції PaC, парадигми Shift-Left та методології GitOps критично знижує ризик несанкціонованих змін і появи привілейованих конфігурацій.

У межах запропонованої моделі інтеграція PaC у конвеєри CI/CD дозволяє рушіям політик

виконувати роль безапеляційного запобіжника, який фізично блокує розгортання вразливого коду ще на етапі його написання. Своєю чергою, контроль на етапі експлуатації через GitOps повністю вирішує проблему інфраструктурного дрейфу шляхом безперервного примирення фактичного стану хмари з еталонним репозиторієм. Таким чином, сформований замкнений цикл «написання – автоматизована перевірка – безпечне розгортання – запобігання конфігураційному дрейфу» є доведеною та обов'язковою умовою для побудови стійких контейнерних хмарних інфраструктур.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ


1. What is Infrastructure as Code (IaC)?. *Red Hat - We make open source technologies for the enterprise*. [Електронний ресурс]. Режим доступу: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (дата звернення: 19.03.2026).
2. HashiCorp. What is Terraform | Terraform | HashiCorp Developer. *What is Terraform | Terraform | HashiCorp Developer*. [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/terraform/intro (дата звернення: 19.03.2026).
3. Overview. *Kubernetes*. [Електронний ресурс]. Режим доступу: https://kubernetes.io/docs/concepts/overview/ (дата звернення: 19.03.2026).
4. Using Helm | Helm. *Helm*. [Електронний ресурс]. Режим доступу: https://helm.sh/docs/intro/using_helm/ (дата звернення: 19.03.2026).
5. HashiCorp. Policy as Code | Sentinel | HashiCorp Developer. *Policy as Code | Sentinel | HashiCorp Developer*. [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/sentinel/docs/concepts/policy-as-code (дата звернення: 19.03.2026).
6. *NIST Technical Series Publications*. [Електронний ресурс]. Режим доступу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-128.pdf (дата звернення: 19.03.2026).
(дата звернення: 19.03.2026).
7. Security in Amazon S3 - Amazon Simple Storage Service. [Електронний ресурс]. Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/security.html (дата звернення: 19.03.2026).
8. OPA for Kubernetes Admission Control | Open Policy Agent. *Open Policy Agent - Homepage | Open Policy Agent*. [Електронний ресурс]. Режим доступу: https://www.openpolicyagent.org/docs/kubernetes (дата звернення: 19.03.2026).



**Артем Вячеславович Кондратюк** – студент групи 2КІТС-24б, факультет менеджменту та інформаційної безпеки, Вінницький національний технічний університет, м. Вінниця, e-mail: bbeverlyhillsss@gmail.com;

Науковий керівник: **Присяжний Дмитро Петрович** – асистент кафедри Менеджменту та безпеки інформаційних систем, e mail:d@vntu.edu.ua

**Artem Kondratiuk V.** – student of group 2KITS-24b, Faculty of Management and Information Security, Vinnytsia National Technical University, Vinnytsia, e-mail: bbeverlyhillsss@gmail.com;

Supervisor: **Dmytro Prysiazhnyi P**. – assistant of the Department of Management and Security of Information Systems, e-mail: e-mail: d@vntu.edu.ua