

# АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ ІТ-ІНФРАСТРУКТУРИ ЗА ДОПОМОГОЮ МЕТОДОЛОГІЇ GITOPS

Ільїнов Кирило Миколайович

*студент групи МПМ-1-25-1.4д,*

*Київського столичного університету імені Бориса Грінченка,*

*науковий керівник – д.т.н., проф. Бушма О.В.*

## ВСТУП

**Актуальність і постановка проблеми.** Сучасні програмні системи працюють у хмарних середовищах і оновлюються кілька разів на день. Щоб робити це швидко й без помилок, організації автоматизують розгортання ІТ-інфраструктури. Опис серверів, мереж і застосунків зберігають у вигляді програмного коду, а спеціальні засоби автоматично приводять систему до цього опису [1].

Найпоширенішою схемою такої автоматизації є модель централізованого керування, яку також називають моделлю «проштовхування» (від англ. push). У ній зовнішній сервер автоматизації сам ініціює з'єднання з цільовими серверами та надсилає їм команди [1]. Ця схема проста й зрозуміла, тому її використовують у більшості класичних конвеєрів розгортання.

Проте модель проштовхування має структурний недолік. Щоб прийняти команду, цільовий сервер мусить постійно тримати відкритим порт віддаленого керування. Прикладом є порт протоколу SSH (від англ. Secure Shell – протокол захищеного віддаленого доступу). Кожен відкритий порт розширює поверхню атаки, тобто множину точок, через які зловмисник може спробувати проникнути в систему [2]. Крім того, зовнішній сервер автоматизації зберігає облікові дані до всіх цільових серверів. Якщо зловмисник захопить цей сервер, він отримає доступ до всієї інфраструктури одночасно.

Отже, проблема полягає в тому, що модель централізованого керування поєднує автоматизацію розгортання з постійно відкритою точкою входу до робочих серверів.

**Аналіз останніх досліджень і публікацій.** Традиційні засоби захисту спрямовані на те, щоб ускладнити доступ до відкритого порту. До них належать віртуальні приватні мережі, проміжні (бастіонні) сервери та правила мережевих екранів [2]. Ці засоби зменшують ризик, але не усувають його причину – саму наявність точки входу.

Інший підхід пропонує методологія GitOps (від англ. Git та Operations – експлуатація). Її суть полягає в тому, що бажаний стан системи описують у репозиторії системи контролю версій Git, а зміни до робочого середовища вносять лише через цей репозиторій [3]. У 2021 році робоча група фонду CNCF (від англ. Cloud Native Computing Foundation – фонд хмарних обчислень) сформулювала чотири принципи GitOps [4]. Бажаний стан системи має бути декларативним, тобто описувати результат, а не послідовність дій. Він має зберігатися з повною історією версій. Програмні агенти мають самостійно витягувати (від англ. pull) цей стан із джерела. Нарешті, агенти мають безперервно узгоджувати з ним фактичний стан системи [4].

Ідея безперервного узгодження походить від систем керування кластерами Borg і Kubernetes. У них окремі програмні контролери циклічно порівнюють фактичний стан із бажаним і поступово усувають різницю [5]. Ф. Беец і С. Харпер розглядають GitOps як розвиток практик DevOps (від англ. Development та Operations – поєднання розробки та експлуатації). Автори наголошують передусім на відтворюваності розгортання та прозорості змін [3]. Документація інструменту Argo CD зазначає, що за автоматичної синхронізації конвеєрам неперервної інтеграції та доставки (CI/CD) не потрібен прямий доступ до сервера розгортання [6].

Таким чином, у розглянутих джерелах [3; 4; 6; 9] модель витягування описано переважно як зручний спосіб доставки змін. Її застосування як засобу мережевої ізоляції робочих серверів розкрито недостатньо. Мова йде про архітектуру, у якій вхідні з'єднання на порти керування заборонено повністю, а конфіденційні дані передаються лише в зашифрованому вигляді.

Актуальність дослідження зумовлена тим, що захист відкритого порту не ліквідує першопричину загрози. Відсутність цілісного рішення, яке поєднує автоматизацію розгортання з відмовою від вхідних з'єднань на порти керування, зумовила вибір теми цієї роботи.

**Мета дослідження** – розробити архітектуру конвеєра розгортання ІТ-інфраструктури на основі методології GitOps, у якій робочі сервери не приймають вхідних з'єднань на порти керування, а оновлення конфігурації відбувається лише через вихідні запити внутрішнього агента до репозиторію Git.

**Наукова новизна.** Набув подальшого розвитку підхід до автоматизації розгортання ІТ-інфраструктури за методологією GitOps. На відміну від відомих рішень, де модель витягування використовують насамперед для зручності доставки змін, запропоновано застосовувати її як засіб мережевої ізоляції. Архітектура поєднує заборону вхідних з'єднань на порти керування, автономного агента синхронізації та асиметричне шифрування секретів. Це дає змогу усунути постійну точку входу для віддаленого керування без втрати автоматизації.

**Об'єкт дослідження** – процеси розгортання та керування конфігурацією хмарної ІТ-інфраструктури.

**Предмет дослідження** – архітектура та засоби автоматизованого розгортання ІТ-інфраструктури за методологією GitOps на основі моделі витягування.

Для досягнення мети поставлено такі **завдання**:

- 1) сформулювати архітектурну модель конвеєра розгортання на базі системи оркестрації контейнерів Kubernetes, у якій ініціатором усіх з'єднань є внутрішній агент;
- 2) розробити декларативну конфігурацію автоматичної синхронізації кластера з репозиторієм Git, що включає механізми самовідновлення та захищеного зберігання секретів;
- 3) отримати експериментальні оцінки доступності портів керування, часу розгортання та часу самовідновлення на тестовому стенді.

## МЕТОДИ ТА ЗАСОБИ ДОСЛІДЖЕННЯ

Для досягнення мети використано метод порівняльного аналізу (для зіставлення моделей прошивування та витягування), метод архітектурного моделювання та експеримент на тестовому стенді.

Увесь інфраструктурний код зберігається в репозиторії Git у вигляді декларативних маніфестів. Маніфести записано у форматі YAML (від англ. YAML Ain't Markup Language – текстовий формат опису даних). Репозиторій виступає єдиним джерелом достовірної інформації про конфігурацію системи [3; 4].

Архітектуру запропонованого рішення показано на рис. 1. Робоче середовище побудовано на системі оркестрації контейнерів Kubernetes. Кластер складається з керуючого вузла та робочих обчислювальних вузлів. У середині кластера працює програмний агент Argo CD. Його компоненти розгорнуто в окремому просторі імен, тобто в логічно відокремленій частині кластера. Агент сам ініціює вихідні запити до репозиторію через захищений протокол HTTPS (від англ. HyperText Transfer Protocol Secure). Мережевий екран цільового середовища відхиляє будь-які вхідні з'єднання на порти керування.

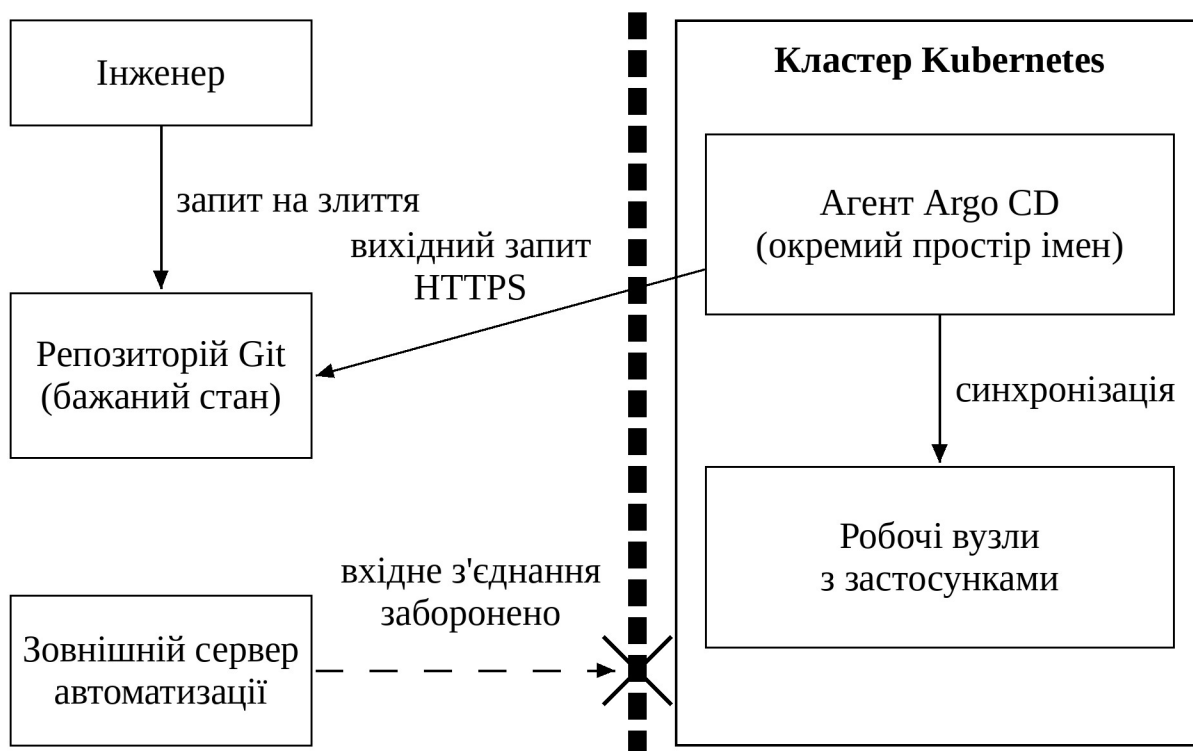


Рис. 1. Архітектура конвеєра розгортання за моделлю витягування (пунктирна смуга – мережевий екран)

Права агента обмежено механізмом RBAC (від англ. Role-Based Access Control – керування доступом на основі ролей). Агент може змінювати лише ті ресурси, які дозволено його ролі [7]. Тому, навіть у разі помилки, в конфігурації агент не отримує повного контролю над кластером.

Агент працює за принципом замкненого циклу узгодження. Він періодично зчитує бажаний стан із репозиторію, порівнює його з фактичним

станом кластера та усуває розбіжності [5; 6]. Логіку синхронізації задає декларативний маніфест, наведений на рис. 2.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: main-web-service
  namespace: argocd
spec:
  source:
    repoURL: https://github.com/org/infra.git
    targetRevision: HEAD
  destination:
    server: https://kubernetes.default.svc
    namespace: production
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

Рис. 2. Маніфест автономної синхронізації застосунку з репозиторієм Git

Поле `source` визначає адресу репозиторію та гілку, з якої агент бере бажаний стан. Поле `destination` вказує кластер і простір імен, куди розгортається застосунок. Блок `automated` вмикає автоматичну синхронізацію без участі людини [6].

Параметр `selfHeal` (самовідновлення) забезпечує детермінованість системи. Якщо хтось вносить зміни безпосередньо в кластер, оминаючи репозиторій, агент фіксує відхилення від еталона і повертає стан, описаний у Git [6]. Параметр `prune` (видалення зайвого) забезпечує автоматичне видалення ресурсів, які прибрали з репозиторію [6].

Окремою проблемою є зберігання конфіденційних даних, наприклад паролів. Для цього застосовано асиметричне шифрування за допомогою контролера `Sealed Secrets` [8]. Розробник шифрує пароль локально відкритим ключем кластера. Репозиторій Git зберігає лише шифротекст. Розшифрувати його може тільки контролер усередині кластера, бо закритий ключ не залишає меж кластера [8].

Процес розгортання виключає прямий доступ інженерів до робочого середовища. Будь-яка зміна інфраструктури проходить через запит на злиття коду в репозиторії. Такий запит переглядають інші учасники команди, а історія Git автоматично стає журналом усіх змін [3].

Ефективність архітектури перевірено на тестовому кластері Kubernetes. Апаратне забезпечення стенда: 4 віртуальні процесорні ядра, 8 ГБ оперативної пам'яті, операційна система Linux. Як тестове навантаження використано

мікросервісний застосунок. Експеримент містив три перевірки. Перша – сканування мережевих портів кластера ззовні. Друга – вимірювання часу повного розгортання після злиття змін у репозиторії. Третя – імітація несанкціонованої локальної зміни для оцінки швидкості самовідновлення.

## РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Сканування мережевих портів кластера засобами мережевого аудиту підтвердило, що порти керування недоступні ззовні мережі. Отже, у кластері немає точки входу, через яку зовнішній сервер міг би передавати команди.

Вимірювання показали, що агент оновлює систему в середньому за 2 хвилини 15 секунд від моменту злиття змін у репозиторії. Це значення охоплює очікування чергового циклу опитування репозиторію та запуск нових контейнерів.

Другий етап експерименту підтвердив роботу циклу узгодження. Агент виявив несанкціоновану зміну за 3 секунди. Повне повернення до стану, описаного в репозиторії, тривало 14 секунд і відбулося без втручання людини. Порівняння двох моделей розгортання наведено в табл. 1.

Таблиця 1

Порівняння моделей розгортання інфраструктури

| Критерій                   | Модель<br>проштовхування               | Модель витягування<br>(GitOps)             |
|----------------------------|----------------------------------------|--------------------------------------------|
| Стан портів керування      | Відкриті для входних з'єднань          | Закриті для входних з'єднань               |
| Ініціатор з'єднання        | Зовнішній сервер автоматизації         | Внутрішній агент                           |
| Напрямок трафіку керування | Вхідний                                | Вихідний                                   |
| Джерело конфігурації       | Сервер автоматизації, локальні скрипти | Репозиторій Git                            |
| Реакція на ручну зміну     | Зміна залишається                      | Автоматичне повернення до еталона          |
| Зберігання паролів         | На сервері автоматизації               | У репозиторії лише в зашифрованому вигляді |

Отримані дані свідчать про перевагу моделі витягування з погляду мережевої безпеки. Вилучення людини з безпосереднього внесення змін підвищило передбачуваність стану інфраструктури.

Водночас запропонована архітектура має обмеження. Агентів потрібен вихідний доступ до репозиторію. Тому захист переноситься на сам репозиторій: важливо обмежити права на злиття змін і перевіряти кожен запит. Також порт, через який користувачі звертаються до застосунку, залишається відкритим, але він не дає змоги керувати сервером.

## ВИСНОВКИ

У роботі показано, що модель централізованого керування створює постійну точку входу до робочих серверів через відкриті порти керування. Для

усунення цієї вразливості розроблено архітектуру конвеєра розгортання на основі методології GitOps.

При виконанні дослідження, по-перше, сформовано архітектурну модель на базі Kubernetes, у якій ініціатором усіх з'єднань є внутрішній агент Argo CD. По-друге, розроблено декларативну конфігурацію синхронізації з механізмами самовідновлення та асиметричного шифрування секретів. По-третє, отримано експериментальні оцінки: порти керування недоступні ззовні, середній час розгортання становить 2 хвилини 15 секунд, а час самовідновлення – 14 секунд.

Таким чином, запропоноване архітектурне рішення конвеєра розгортання ІТ-інфраструктури на основі методології GitOps дозволило успішно реалізувати автоматичне оновлення робочих серверів лише через вихідні запити без відкритих портів керування.

### **ДЖЕРЕЛА**

1. Morris K. Infrastructure as Code: Dynamic Systems for the Cloud Age. 2nd ed. Sebastopol : O'Reilly Media, 2020.

2. Scarfone K., Jansen W., Tracy M. Guide to General Server Security : NIST Special Publication 800-123. Gaithersburg : National Institute of Standards and Technology, 2008. DOI: <https://doi.org/10.6028/NIST.SP.800-123>.

3. Beetz F., Harrer S. GitOps: The Evolution of DevOps? IEEE Software. 2022. Vol. 39, No. 4. P. 70–75. DOI: <https://doi.org/10.1109/MS.2021.3119106>.

4. GitOps Principles v1.0.0 / OpenGitOps. 2021. URL: <https://opengitops.dev/> (дата звернення: 08.10.2026).

5. Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J. Borg, Omega, and Kubernetes. Communications of the ACM. 2016. Vol. 59, No. 5. P. 50–57. DOI: <https://doi.org/10.1145/2890784>.

6. Automated Sync Policy. Argo CD Documentation. URL: [https://argo-cd.readthedocs.io/en/stable/user-guide/auto\\_sync/](https://argo-cd.readthedocs.io/en/stable/user-guide/auto_sync/) (дата звернення: 08.10.2026).

7. Using RBAC Authorization. Kubernetes Documentation. URL: <https://kubernetes.io/docs/reference/access-authn-authz/rbac/> (дата звернення: 08.10.2026).

8. Sealed Secrets: a Kubernetes controller and tool for one-way encrypted Secrets / Bitnami Labs. URL: <https://github.com/bitnami-labs/sealed-secrets> (дата звернення: 08.10.2026).

9/ Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

URL: <https://doi.org/10.28925/9720213284km>