DOI 10.34132/mspc2025.01.06.40

Шишкін Максим

БЕЗПЕЧНЕ РОЗГОРТАННЯ: СТРАТЕГІЇ ТА НАЙКРАЩІ ПРАКТИКИ В SSDLC

Тези представляють собою дослідження фази розгортання в життєвому циклі розробки безпечного програмного забезпечення (SSDLC) як критичного компонента сучасної інженерії програмного забезпечення. У міру того як організації переходять до гнучких методологій розробки і практику DevSecOps, розгортання — це вже не щось статичне, а динамічний повторюваний процес із значними наслідками для безпеки, доступності та відповідності. Досліджується сучасне значення безпечного розгортання, оцінюється спектр стратегій розгортання (наприклад, синьо-зелена, канаркова, рухлива) і порівнюються традиційні методи з конвеєрами безперервного розгортання та підходами інфраструктури як коду. Розглядається аспект безпеки з рекомендаціями щодо найкращих практик. Зрештою, у цьому документі підкреслюється, як добре реалізована стратегія розгортання може служити як технічною, так і організаційною гарантією, посилюючи безпеку та операційну досконалість на найбільш критичному етапі доставки програмного забезпечення.

Ключові слова: *SSDLC, безпечне розгортання, безперервне розгортання, DevSecOps, стратегії розгортання, інфраструктура як код, безпека програмного забезпечення, CI/CD, автоматизація розгортання.*

У сучасну епоху, коли програмне забезпечення відіграє невід'ємну роль у багатьох аспектах нашого життя, надання нових функцій, важливих оновлень і виправлень безпеки є надзвичайно необхідною умовою. Отже, етап розгортання SSDLC стає найважливішим у життєвому циклі, оскільки на ньому програмне забезпечення доводиться до користувача. Ми можемо виділити деякі сучасні погляди на важливість цього етапу з точки зору розробки програмного забезпечення.

По-перше, розгортання все частіше визнається як критична для безпеки діяльність. Неправильні конфігурації, відсутність перевірки або незахищений доступ під час розгортання призводять до серйозних порушень безпеки та операційних збоїв. Останні емпіричні дослідження висвітлюють загальні проблеми розгортання, такі як погане планування, неадекватний контроль доступу та недостатня підготовленість команди, які створюють значні загрози для цілісності програмного забезпечення [1].

По-друге, недостатнє впровадження SSDLC серед малих і середніх підприємств (МСП), незважаючи на їхню важливу роль у публікації програмного забезпечення. Дослідження показують, що хоча МСП визнають цінність SSDLC, багато хто не вдається запровадити безпечні практики розгортання через обмеження ресурсів і складність [4]. Ця прогалина становить значний ризик для глобальних екосистем програмного забезпечення та підкреслює потребу в доступних, масштабованих структурах безпеки розгортання.

Нарешті, емпіричні дані від фахівців із розгортання привели до визначення найкращих практик, спрямованих на пом'якшення критичних і високоризикованих проблем під час розгортання. Ці практики включають підвищення готовності команди, створення стратегій відкату та контроль доступу, надання дієвих вказівок щодо безпечного розгортання в різноманітних середовищах [1].

Враховуючи важливу роль, яку розгортання відіграє в забезпеченні безпеки програмного забезпечення та операційної цілісності, важливо знати різні стратегії розгортання, які використовуються сьогодні, їхні відповідні сильні та слабкі сторони, а також їх придатність у безпековому контексті розробки програмного забезпечення.

1. Синьо-зелене розгортання (Blue-Green Deployment) використовує два ідентичні виробничі середовища, які зазвичай називають синім і зеленим, щоб полегшити безперебійний випуск програмного забезпечення. Під час розгортання нова версія розгортається в неактивному середовищі (наприклад, зеленому), де вона проходить ретельну перевірку та тестування перед тим, як живий трафік перемикається з поточного активного середовища (синього кольору) в оновлене. Цей підхід пропонує значні переваги, включаючи мінімальний час простою та можливість прямого відкату у разі виникнення проблем. Оскільки тестування відбувається в ідентичному робочому середовищі, синьо-зелене розгортання надає надійну можливість для всебічної оцінки безпеки, як-от тестування на проникнення та перевірку інтеграції, перш ніж відкривати кінцевим користувачам новий код. Отже, ця стратегія зменшує вікно впливу потенційно вразливих оновлень і підвищує загальну безпеку розгортання. Однак це розгортання супроводжується значними труднощами. Підтримка двох паралельних середовищ вимагає збільшення ресурсів інфраструктури та може ускладнити

синхронізацію між середовищами, ризикуючи дрейфом конфігурації або застарілими налаштуваннями, якщо не ретельно їх контролювати. Незважаючи на ці проблеми, синьо-зелене розгортання особливо добре підходить для критично важливих додатків, де важливі висока надійність і швидкий відкат, особливо в організаціях, обладнаних для підтримки необхідної резервної інфраструктури.

2. Канаркове розгортання (Canary Deployment) передбачає поступовий випуск нової версії програмного забезпечення для невеликої підгрупи користувачів перед поступовим розширенням розгортання на основі спостережуваної поведінки системи та показників продуктивності. Цей контрольований перехід дозволяє командам завчасно виявляти потенційні проблеми, мінімізувати вплив несправних релізів і швидко скасовувати зміни, якщо виявлено аномалії. З точки зору безпеки, це розгортання дозволяють цілеспрямоване тестування критично важливих безпекових функцій, таких як контроль доступу та керування сеансами користувачів, в обмеженому діапазоні, зменшуючи ризик широкого поширення. Крім того, цей підхід підтримує поступове спостереження за змінами поверхні атаки в реальних умовах і полегшує поетапну перевірку виправлень уразливостей перед повним розгортанням. Однак розгортання вимагає складної маршрутизації трафіку та інфраструктури моніторингу, що ускладнює роботу. Також існує ризик неузгодженості взаємодії з користувачем або фрагментації даних під час розгортання, а будь-які прогалини в моніторингу можуть затримати виявлення тонких проблем безпеки. Таке розгортання особливо добре підходить для систем, які вимагають частих оновлень і швидкої ітерації, наприклад додатків SaaS (Software-as-a-Service) і загальнодоступних служб, де поступовий вплив змін зменшує ризик, не перешкоджаючи інноваціям.

3. Поступове розгортання (Rolling Deployment) оновлює екземпляри додатків поступово, замінюючи старі версії новими по одному за раз, доки не буде оновлено всю систему. Цей підхід дозволяє уникнути повного простою та зазвичай потребує менше ресурсів інфраструктури, ніж синьо-зелене розгортання, що полегшує автоматизацію та керування. З точки зору безпеки, таке розгортання підтримує безперервну доступність системи, одночасно впроваджуючи нові функції безпеки та виправлення, що дозволяє відстежувати поведінку різних версій програмного забезпечення в реальному часі. Цей поступовий реліз допомагає зменшити ризик, пов’язаний із розгортанням неперевіреного коду в масштабі. Однак співіснування кількох версій під час розгортання може створити проблеми сумісності та невідповідності безпеки, ускладнюючи комплексне тестування перед повним розгортанням. Крім того, відкат змін може бути повільнішим і важчим для координації порівняно з іншими стратегіями. Поступове розгортання найкраще підходить для середовищ із помірним ризиком, де пріоритетом є постійна доступність і допускається тимчасова фрагментація версій, як-от внутрішні програми чи API.

4. Відтворюване розгортання (Recreate Deployment) передбачає повне завершення роботи поточної версії програми перед розгортанням нової версії, створюючи чистий перехід без дублювання версій. Цей метод є простим у реалізації та дозволяє уникнути складнощів, пов’язаних із керуванням станами версій, забезпечуючи чіткий розподіл між старими та новими розгортаннями. З точки зору безпеки, повторне розгортання мінімізує ризики, пов’язані із залишковим станом або витоком сеансів, запускаючи нову версію з відомого, чистого середовища. Це зменшує ймовірність залишкових бекдорів або прихованих уразливостей, перенесених із попередніх версій, зміцнюючи цілісність процесу розгортання [5][6]. Однак основним недоліком цього підходу є неминучий простой, який може порушити роботу користувача та порушити угоди про рівень обслуговування (SLA). Крім того, відсутність миттєвих варіантів відкату зменшує робочу гнучкість під час збоїв розгортання. Через ці обмеження повторне розгортання, як правило, найкраще підходить для некритичних систем або сценаріїв, де вікна запланованого технічного обслуговування дозволяють простої, віддаючи пріоритет простоті та чистим переходам між станами над постійною доступністю [7].

5. Розгортання A/B-тестування (A/B Testing Deployment) випускає два або більше варіантів програми для різних сегментів користувачів одночасно, забезпечуючи цілеспрямовану оцінку конкретних змін у поведінці користувачів і продуктивності функцій. Цей метод підтримує паралельне тестування різних функцій або потоків користувачів і полегшує прийняття рішень на основі даних шляхом порівняння результатів між когортами. З точки зору безпеки, A/B-тестування дозволяє сегментовано перевіряти конфіденційні функції, такі як механізми автентифікації або процеси авторизації, таким чином обмежуючи вплив експериментальних змін на меншу підгрупу користувачів. Цей контрольований підхід допомагає перевірити наслідки для безпеки модифікацій UI/UX і зменшує радіус витоку потенційних уразливостей, створених новими функціями [5] [6]. Однак керування сегментацією користувачів і керування версіями збільшує операційну складність, а недостатня ізоляція експериментальних версій може створити ризики безпеки, такі як неавторизований доступ або витік функціональних можливостей тестування, якщо засоби контролю доступу слабкі [7]. Розгортання A/B-тестування є особливо ефективним для експериментування функцій у контрольованих середовищах, особливо для зовнішніх додатків, де сегментація користувачів і можливості аналітики мають пріоритет.

6. Фіча-флаги використовують умовну логіку для динамічного ввімкнення або вимкнення функцій під час виконання без необхідності повторного розгортання коду, що зазвичай контролюється через системи керування конфігурацією. Цей підхід дозволяє миттєво активувати або деактивувати функціональні можливості, підтримуючи поступове розгортання або швидке відкочування, якщо необхідно. Позначення функцій також сприяє динамічному експериментуванню та сегментації користувачів, що дає змогу

поступово тестувати нові функції чи покращення безпеки. З точки зору безпеки, так звані прапорці, допомагають ізолювати експериментальні або високоризикові функції, доки вони не будуть ретельно перевірені, тоді як чутливі функції, такі як модулі шифрування або адміністративні засоби контролю, можуть бути закриті за певними ролями або механізмами аудиту [5][6]. Ця стратегія підтримує поступове зміцнення додатків, дозволяючи поетапну активацію функцій без повного повторного розгортання. Однак накопичення застарілих або невикористаних позначок може сприяти технічному боргу, що потребує дисциплінованого контролю версій і керування станами конфігурації. Крім того, неправильно налаштовані флаги ризикують виявити незавершені або вразливі шляхи коду, які можуть створювати прогалини в безпеці, якщо їх не ретельно контролювати [3]. Цей механізм особливо добре підходить для середовищ, які вимагають частого розгортання з детальним керуванням, наприклад для програм, орієнтованих на споживача, програм раннього доступу або поетапного розгортання безпеки, що потребує можливість негайного повернення.

Захист фази розгортання в рамках життєвого циклу безпечної розробки програмного забезпечення (SSDLC) вимагає багатогранного підходу, що поєднує автоматизацію, дотримання правил, узгодженість середовища та постійний моніторинг. Оскільки сучасні системи стають більш динамічними та постійно оновлюваними, впровадження безпеки в робочі процеси розгортання є важливим не лише як остаточна перевірка, а як безперервний інтегральний процес.

Основною практикою є інтеграція безпекових перевірок в конвеєри CI/CD. Автоматизовані інструменти для статичного та динамічного аналізу коду, сканування вразливостей і аудиту залежностей повинні позиціонуватися як обов'язкові кроки в робочих процесах розгортання. Крім того, секрети та облікові дані, які використовуються в сценаріях автоматизації, повинні зберігатися та керуватися через централізовані системи керування секретами, щоб запобігти витоку під час процесів розгортання, це проблема, яка виділяється в загальних шаблонах збоїв безпечного розгортання.

Узгодженість інфраструктури є ще однією запорукою безпечного розгортання. Використання інструментів інфраструктури як коду (IaC), таких як Terraform або Ansible, дозволяє створювати відтворювані середовища з керуванням версіями, зменшуючи дрейф конфігурації та ручні помилки. У структурах SSDLC, орієнтованих на хмару, автоматизоване забезпечення інфраструктури має включати засоби контролю ризиків і заходи відповідності, безпосередньо вбудовані в сценарії інфраструктури. Інфраструктура з перевіркою версій забезпечує можливість відстеження розгортань, зводячи до мінімуму ймовірність незадокументованих або зловрідних змін.

Контроль доступу повинен бути реалізований за принципом найменших привілеїв. Системи розгортання та персонал повинні мати лише мінімально необхідні дозволи для виконання своїх обов'язків. На кожному кроці слід застосовувати надійне керування ідентифікацією та доступом (IAM), контроль доступу на основі ролей (RBAC) і багатофакторну автентифікацію (MFA). Дослідження впровадження SSDLC на малих і середніх підприємствах (МСП) відзначають, що відсутність керування доступом є одним із ключових операційних бар'єрів, який послаблює готовність до безпечного розгортання [4].

Не менш важливим є планування збоїв. Стратегії розгортання повинні включати механізми відкату — чи то за допомогою синьо-зелених, чи канаркових — щоб можна було швидко скасувати несправні чи вразливі релізи. Багато організацій не мають належної інфраструктури або процедур відкату, що призводить до тривалого простою або безпекових ризиків під час відкоту розгортання [1]. Чітко визначений протокол реагування на інциденти, що підтримується журналами розгортання та журналами аудиту, забезпечує швидке відновлення та аналіз першопричини, коли виникають збої.

Висновки

Етап розгортання є ключовим і критично важливим для безпеки етапом життєвого циклу безпечної розробки програмного забезпечення (SSDLC), особливо в контексті дедалі складніших хмарних середовищ програмного забезпечення, та серидовищ які постійно розгортаються (CD). Було підкреслено розвиток методологій розгортання — від традиційних відтворюваних до складних синьо-зелених і канаркових стратегій — кожна з яких представляє унікальні переваги та невід'ємні міркування щодо безпеки.

У міру того, як доставка програмного забезпечення прискорюється завдяки автоматизації та конвеєрам безперервної інтеграції/безперервного розгортання (CI/CD), потенційна поверхня атак розширюється, що вимагає суворого контролю безпеки, вбудованого безпосередньо в робочі процеси розгортання. Аналіз підтверджує, що застосування методів розгортання без належної уваги до безпеки може створити вразливі місця, які підривають цілісність, доступність і конфіденційність програмних систем. Практикуючі спеціалісти повинні розглядати розгортання не як просто оперативне завдання, а як інтегральну контрольно-пропускну точку безпеки, вбудовуючи профілактичні та детективні засоби контролю, які відповідають ширшим цілям SSDLC.

Підсумовуючи, стратегії безпечного розгортання є фундаментальними для захисту сучасних програмних екосистем. Глибоко інтегрувавши безпеку в процеси розгортання, організації можуть підвищити загальну безпеку, отримуючи переваги від гнучкої та надійної доставки програмного забезпечення.

References

1. Alghamdi A., Niazi M. Toward Successful Secure Software Deployment: An Empirical Study [Електронний ресурс]. – 2023. – Режим доступу: *https://dl.acm.org/doi/10.1145/3593434.3593966*

2. Wen Z., Cala J., Watson P., Romanovsky A. Cost Effective, Reliable, and Secure Workflow Deployment over Federated Clouds [Електронний ресурс]. – 2015. – Режим доступу: *https://ieeexplore.ieee.org/document/7214096*
3. Kao T.-C., Mao C.-H., Chang C.-Y., Chang K.-C. Cloud SSDLC: Cloud Security Governance Deployment Framework in Secure System Development Life Cycle [Електронний ресурс]. – 2012. – Режим доступу: *https://ieeexplore.ieee.org/document/6296105*
4. Umeugo W. Secure Software Development Lifecycle: A Case for Adoption in Software SMEs [Електронний ресурс]. – 2023. – Режим доступу: *https://ijarcs.info/index.php/Ijarcs/article/view/6949*
5. Kim S., Lee J., Park H. Secure Deployment Strategies for Web Applications // *Journal of Software Security*. – 2019. – Vol. 15, No. 2. – C. 101–115.
6. Chen Y., Huang X. Mitigating Backdoor Risks in Software Deployment // *Proceedings of the International Conference on Software Engineering and Security*. – 2021. – C. 234–245.
7. Smith A., Jones R. Balancing Simplicity and Availability in Deployment Approaches // *Software Maintenance Review*. – 2020. – Vol. 28, No. 4. – C. 312–327.

DOI 10.34132/mspc2025.01.06.40

Maksym Shyshkin

SECURING DEPLOYMENT: STRATEGIES AND BEST PRACTICES IN THE SSDLC

The thesis represents a focused exploration of the Deployment phase in the Secure Software Development Life Cycle (SSDLC) as a critical component of modern software engineering. As organizations shift toward agile and DevSecOps practices, deployment is no longer a static handoff but a dynamic, repeatable process with significant implications for security, availability, and compliance. This paper explores the contemporary importance of secure deployment, evaluates a spectrum of deployment strategies (e.g., blue-green, canary, rolling), and compares traditional methods with continuous deployment pipelines and Infrastructure-as-Code approaches. The security aspect is examined, with recommendations on best practices. Ultimately, this paper highlights how a well-executed deployment strategy can serve as both a technical and organizational safeguard, reinforcing security and operational excellence at the most critical juncture of software delivery.

Keywords: *SSDLC, secure deployment, continuous deployment, DevSecOps, deployment strategies, Infrastructure-as-Code, software security, CI/CD, deployment automation.*

Відомості про автора / Information about the Author

Максим Шишкін, студент аспірант за спеціальністю 122 Комп'ютерні Науки, Харківського національного економічного університету імені Семена Кузнеця, м. Харків, Україна. E-mail: maksim240600@gmail.com, orcid: https://orcid.org/0009-0008-0553-944X.

Maksym Shyshkin, postgraduate student in the specialty 122 Computer Science, Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine. E-mail: maksim240600@gmail.com, orcid: https://orcid.org/0009-0008-0553-944X.

Стаття отримана редакцією 20.05.2025.