УДК 004.4/.415+004.89/.9
DOI:

ВПЛИВ ХМАРНИХ СЕРВІСІВ НА БЕЗПЕЧНУ РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Максим Шишкін
Аспірант
Харківський національний економічний університет імені Семена Кузнеця
Харків, Україна
https://orcid.org/0009-0008-0553-944X

Анотація. Еволюція хмарних обчислень суттєво змінила підходи до розроблення, тестування, розгортання та захисту програмного забезпечення. Метою дослідження є аналіз інтеграції хмарних технологій у всі етапи життєвого циклу розробки програмного забезпечення (Software Development Life Cycle, SDLC) з акцентом на систематичне впровадження та застосування практик безпеки протягом усього процесу розробки. Для досягнення цілей дослідження використано порівняльний аналіз, що охоплює хмарні реалізації SDLC у співвідношенні з традиційними моделями за ключовими показниками: економічною ефективністю, швидкістю розгортання, рівнем співпраці, масштабованістю та безпекою. Розглянуто реальні тематичні приклади, які демонструють використання хмарних інструментів і платформ, зокрема Інфраструктури як коду (IaC) та Безперервної інтеграції/Безперервного розгортання (CI/CD), для підвищення продуктивності, гнучкості та раннього впровадження механізмів контролю безпеки (підхід «зміщення безпеки вліво»). Результати аналізу показують, що застосування хмарних практик у безпечному SDLC (Secure SDLC, SSDLC) сприяє скороченню часу виходу продукту на ринок, підвищує рівень проактивного управління безпекою та підтримує ітеративні й гнучкі цикли розробки. Водночас виявлено виклики, пов'язані з дотриманням нормативних вимог, управлінням ідентифікацією користувачів і ризиком залежності від постачальника. Запропоновано набір найкращих практик для впровадження безпечних хмарних робочих процесів SDLC і визначено напрями подальших досліджень, зокрема автоматизоване тестування безпеки та інтеграцію штучного інтелекту у процеси безпечної доставки програмного забезпечення. Практична цінність дослідження полягає у формуванні рекомендацій, що допоможуть організаціям створювати стійкі, ефективні та безпечні хмарні середовища розробки.

Ключові слова Практики кіберзахисту у розробці; Інфраструктура як код; Конвеєри безперервної інтеграції та доставки; Автоматизоване тестування вразливостей; Управління конфігураціями; Моделі розподілу відповідальності у хмарі; Наглядовість та моніторинг систем.

THE IMPACT OF CLOUD SERVICES ON SECURE SOFTWARE DEVELOPMENT

Maksym Shishkin
Postgraduate Student
Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine
https://orcid.org/0009-0008-0553-944X

Abstract. The evolution of cloud computing has significantly changed the approaches to developing, testing, deploying, and securing software. The paper examines the integration of cloud technologies into all stages of the Software Development Life Cycle (SDLC) with a focus on implementing and enforcing security practices throughout the development process. To achieve the objectives of the study, a comparative analysis was used that covers cloud implementations of the SDLC relative to traditional models in terms of key indicators: cost-effectiveness, speed of deployment, level of collaboration, scalability, and security. Real-world case studies are considered that demonstrate the use of cloud tools and platforms, including Infrastructure as Code (IaC) and Continuous Integration/Continuous Deployment (CI/CD), to increase productivity, agility, and early implementation of security controls (a "security shift to the left" approach). The analysis results show that the use of cloud practices in a secure SDLC (Secure SDLC, SSDLC) helps to reduce time to market, increases the level of proactive security management, and supports iterative and agile development cycles. At the same time, challenges related to compliance with regulatory requirements, user identity management, and vendor lock-in risk are identified. A set of best practices for implementing secure cloud SDLC workflows is proposed and areas for further research are identified, including automated security testing and integration of artificial intelligence into secure software delivery processes. The practical value of the study lies in the formulation of recommendations that will help organizations create sustainable, efficient, and secure cloud development environments.

Keywords: DevSecOps practices; Infrastructure as Code; CI/CD pipelines; Automated vulnerability testing; Configuration management; Cloud shared responsibility models; System observability.

Вступ

На момент 2025 року розробка програмного забезпечення відбувається в умовах дедалі більшої динаміки бізнес-вимог, глобалізації команд, високого навантаження на інфраструктуру та безперервної доставки релізів. У такому середовищі традиційні моделі SDLC, хоч і залишаються актуальними для стабільних та передбачуваних проєктів, часто виявляються недостатньо гнучкими без інтеграції хмарних сервісів. Відсутність адаптивності може призводити до підвищення ризиків зловмисних атак, витоків даних, порушень відповідності та втрати контролю над середовищем. Водночас хмарні сервіси пропонують масштабованість, автоматизацію та інтеграцію безпекових механізмів на всіх етапах SDLC, що відкриває нові можливості для створення стійких, безпечних і гнучких процесів розробки. Таким чином, в цих умовах важливо не відмовлятися від традиційних підходів, а навчитися робити обґрунтований вибір між класичними та хмарно-орієнтованими моделями залежно від вимог, ризиків і цілей конкретного проєкту.

У роботі науковців Saleh et al. (2024) проведено систематичний огляд 66 публікацій, що присвячені безпеці конвеєрів CI/CD у хмарному середовищі; автори виявили основні інструменти (наприклад, Harbor, SonarQube, GitHub Actions) та ключові виклики (маніпуляції образами, несанкціонований доступ, слабка автентифікація). Дослідники дійшли висновку, що саме інтеграція безпеки на етапах CI/CD забезпечує суттєве зниження кількості вразливостей у продуктивних середовищах. У дослідженні науковців Chauhan and Shiaeles (2023) розглянуто фреймворки безпеки хмар (The National Institute of Standards and Technology, NIST Cloud; Cloud Security Alliance, CSA STAR, ISO/IEC 27017) та зазначено, що жоден із них не охоплює всі критерії безпеки у повному обсязі, що створює «розрив» у стандартизації. Автори запропонували власну узагальнену модель оцінювання безпеки хмарних сервісів, що комбінує елементи існуючих фреймворків для зменшення розривів між стандартами. У статті автори Kiashemshaki et al. (2025) зосереджуються на практиках безпечного програмування, включно з оцінкою ролі великих мовних моделей (LLM) у генерації коду — зазначено, що незважаючи на зростання автоматизації, впровадження безпечного коду лишається фрагментованим. Вони наголошують, що використання LLM у хмарних середовищах може стати ефективним лише за умови інтеграції систем автоматичної валідації безпеки. У роботі дослідників Vakhula and

Opirskyy (2023), досліджують інтеграцію підходу «Security as Code» у хмарних середовищах для автоматизації безпеки на рівні IaC та DevSecOps. Вони показали, що цей підхід дозволяє зменшити людський фактор у процесі DevSecOps і забезпечити безперервну відповідність систем вимогам стандартів. Праця науковців Matseniuk and Partyka (2024) запропонувала автоматизований метод перевірки відповідності у хмарних облікових записах AWS/GCP/Azure та підкреслила значення стандартизації конфігурацій. Автори розробили модель контролю конфігурацій, що використовує політики IaC для забезпечення постійного аудиту безпеки у DevOps-процесах. У оглядовій роботі дослідник Pawar (2025) описав сучасні підходи до захисту контейнеризованих та мікросервісних середовищ — з акцентом на безпечні практики у «cloud-native» системах, таких як service mesh, runtime protection та DevSecOps. У роботі підкреслено, що перехід до хмарно-нативних технологій потребує нового мислення щодо безпеки, де акцент зміщується з периметрового захисту на безпеку компонентів та процесів розробки.

Незважаючи на значний прогрес у дослідженні безпеки хмарних середовищ та інструментів DevSecOps, меншою мірою аналізовано безпосередній вплив хмарних сервісів на весь життєвий цикл розробки програмного забезпечення (SDLC) у контексті забезпечення безпеки; недостатньо досліджено, як саме застосування хмар-сервісів змінює підходи до вимог, проєктування, тестування та розгортання з точки зору безпеки, особливо в українському чи регіональному контексті. Саме це обґрунтовує проведення даного огляду. Метою дослідження було здійснити систематичний аналіз наукових праць останніх п’яти років, що стосуються впливу хмарних сервісів на безпечну розробку програмного забезпечення, виявити ключові тренди, переваги й виклики, а також окреслити напрями подальших досліджень.

Матеріали і Методи

Життєвий цикл розробки програмного забезпечення описує повну послідовність етапів створення програмного продукту — від аналізу вимог до супроводу в експлуатації. До основних етапів належать: аналіз вимог, проєктування, розробка, тестування, розгортання та підтримка. Така структура забезпечує системність, передбачуваність і контроль якості на всіх стадіях розробки, що дає змогу стандартизувати процес і порівнювати різні підходи до його реалізації. В межах цього дослідження здійснювалося порівняння традиційного та хмарно-орієнтованого життєвого циклу (Cloud SSDLC) з метою визначення їхніх переваг, обмежень і впливу на безпеку розробки.

Традиційний SDLC був охарактеризований як лінійний і послідовний підхід, у якому кожен етап повністю завершується перед початком наступного. Він базується на локальних середовищах розробки, ручному тестуванні, статичному конфігуруванні інфраструктури та обмежених можливостях масштабування. Такий підхід залишається ефективним у стабільних та передбачуваних умовах, але менш адаптивний до динамічних хмарних екосистем.

Окремий фокус у дослідженні приділено моделі SSDLC. На відміну від традиційного SDLC, який передбачає впровадження безпекових практик переважно на етапах тестування або після розгортання, SSDLC інтегрує безпеку на всіх стадіях створення програмного забезпечення. Це охоплює моделювання загроз під час аналізу вимог, проєктування архітектури з урахуванням контрольних точок безпеки, автоматизовану перевірку коду, динамічне й статичне сканування вразливостей, безпечні механізми розгортання та моніторинг у режимі реального часу. Такий підхід підтримує концепцію «shift-left security», при якій ризики усуваються на ранніх етапах, знижуючи вартість виправлення помилок і підвищуючи надійність системи. У рамках хмарного підходу SSDLC поєднується з DevOps-практиками, утворюючи DevSecOps-модель, що базується на автоматизації, інфраструктурі як код та безперервному контролі відповідності.

Cloud SSDLC визначався як модель, що інтегрує DevOps, безперервну інтеграцію та доставку, контейнеризацію, інфраструктуру як код (IaC), автоматизоване тестування й підхід

«Shift-Left Security». Завдяки цьому забезпечуються гнучкість, автоматизація, повторюваність середовищ і вбудований контроль безпеки на всіх етапах життєвого циклу.

Методика порівняння здійснювалася через поетапний аналіз шести фаз SDLC, для кожної з яких оцінювалися такі критерії: автоматизація, гнучкість, масштабованість, безпека, швидкість розгортання, операційні ризики та керування конфігураціями, оскільки саме ці параметри найбільш повно відображають ключові відмінності між традиційними та хмарно-орієнтованими моделями, визначають якість і ефективність виконання кожної фази життєвого циклу, а також безпосередньо впливають на здатність організацій підтримувати сучасні вимоги до надійності, безпеки та швидкості постачання програмного забезпечення.

Для систематизації результатів були сформовані узагальнені порівняльні таблиці, а аналітична частина ґрунтувалася на комплексному огляді різних типів джерел, включаючи рецензовані наукові публікації за 2018–2025 роки, технічні звіти провідних компаній (Amazon, Microsoft, Google, Red Hat, CNCF), офіційну документацію хмарних платформ AWS, Azure і GCP, галузеві стандарти та рекомендації (NIST, OWASP, ISO/IEC 27001/27017), а також аналітичні матеріали індустрії, такі як блоги розробників, whitepapers і репозиторії GitHub.

Збір джерел здійснювався у базах Google Scholar, IEEE Xplore, Scopus, ACM Digital Library та Semantic Scholar. Для пошуку використовувалися ключові слова: "cloud SDLC", "secure SDLC", "DevSecOps", "cloud-native security", "infrastructure as code", "CI/CD security", "shift-left security". До аналізу включалися роботи, що: описують методології SDLC або SSDLC; стосуються хмарної інфраструктури; містять емпіричні результати або формальні моделі. Виключалися джерела, що дублюють результати, не мають технічної конкретики або належать до застарілих технічних підходів (до 2018 року, окрім фундаментальних праць).

Подальший синтез матеріалу здійснювався шляхом класифікації джерел за етапами SDLC та типами технічних рішень (DevOps-процеси, безпека, інфраструктура, тестування), що дозволило цілісно порівняти моделі та сформулювати висновки щодо ефективності Cloud SSDLC.

Результати та Обговорення

Збір та аналіз вимог є першим і одним із найважливіших етапів *SDLC*. На цьому етапі визначаються очікування та обмеження щодо майбутньої програмної системи як з користувацької, так і з технічної перспективи. Процес передбачає тісну взаємодію між усіма зацікавленими сторонами, серед яких — бізнес-аналітики, керівники проєктів, кінцеві користувачі та команди розробників (IEEE Computer Society, 2014).

Результатом цієї роботи зазвичай стає підготовка ключової документації, такої як бізнес-вимоги, функціональні специфікації, а також описи сценаріїв використання та користувацьких історій. У традиційних, не хмарних середовищах цей процес часто має лінійний характер і базується на статичних документах та заздалегідь запланованих зустрічах. Такий підхід може призводити до затримок і непорозумінь між учасниками проєкту (Basili & Turner, 1975).

Вплив хмарних інструментів на збір вимог. Використання хмарних технологій значно змінює підхід до збору вимог. Завдяки можливостям роботи в реальному часі та спільної взаємодії, хмарні інструменти підвищують ефективність, прозорість і гнучкість цього процесу (Google Cloud, 2023).

Серед таких інструментів варто відзначити *Atlassian Jira ma Confluence*, які забезпечують ведення документації та відстеження користувацьких історій із підтримкою редагування в режимі реального часу та контролем версій. Для візуалізації процесів і спільної розробки ідей використовуються *Lucidchart* і *Miro*, а для створення й редагування текстових документів і таблиць — *Google Workspace* і *Microsoft* 365. У сфері дизайну інтерфейсу та досвіду активно застосовуються Figma і FigJam, які підтримують інтерактивне прототипування та спільні воркшопи. Крім того, платформи на кшталт Amazon Honeycode і Google AppSheet надають інструменти з низьким рівнем кодування, що дозволяє швидко створювати прототипи бізнес-процесів на основі зібраних вимог (Amazon Web Services, n.d.).

Добавлено примечание ([1]): У висновках під таблицями необхідно забезпечити, щоб інтерпретація завжди підкреслювала (кібер)безпекові наслідки змін, навіть якщо критерій стосується економіки чи швидкості.

Хмарні інструменти допомагають подолати низку типових труднощів, характерних для традиційного підходу до збору вимог. Вони покращують взаємодію між усіма зацікавленими сторонами завдяки можливості спільного редагування та коментування в реальному часі, що значно зменшує залежність від синхронних зустрічей, також автоматизоване відстеження змін і доступ до історії редагування спрощують контроль версій і забезпечують прозорість процесу (Google Cloud, 2023).

Крім цього, хмарні рішення сприяють прискоренню ітерацій, дозволяючи швидко оновлювати вимоги відповідно до змін у бізнес-середовищі. Нарешті, вони роблять процес більш інклюзивним: учасники з різних локацій і часових поясів можуть брати активну участь, що зменшує ймовірність непорозумінь і покращує узгодженість бачення продукту (Jagli & Yeddu, 2017).

Попри численні переваги хмарних інструментів, деякі проблеми залишаються актуальними. Зокрема, інтенсивна співпраця може призвести до надмірної кількості документації та користувацьких історій, що ускладнює їх пріоритезацію та подальше управління. Також існує проблема фрагментації інструментів: команди часто поєднують кілька платформ (наприклад, Jira, Miro та Google Docs), які не завжди інтегруються безшовно, що ускладнює централізоване управління вимогами.

Окрім того, організації, що працюють за традиційними або гібридними моделями розробки, можуть відчувати труднощі з адаптацією своїх процесів до більш гнучких, хмарно-орієнтованих і Agile-сумісних підходів (Jagli & Yeddu, 2017).

Нові виклики, пов'язані з хмарною інтеграцією. Варто зазначити, що впровадження хмарних платформ супроводжується появою нових викликів, які не були характерними для традиційних моделей роботи. Одним із головних питань є безпека та контроль доступу: конфіденційна проєктна й бізнес-інформація, що зберігається у хмарному середовищі, вимагає ретельного управління ідентифікацією користувачів, ефективного шифрування даних та відповідності нормативним вимогам NIST (2018).

Ще одним викликом є так звана "втома від співпраці" — постійна доступність хмарних інструментів може спричинити перевантаження сповіщеннями, емоційне виснаження працівників і, як наслідок, зниження загальної продуктивності. Крім того, ефективність роботи в хмарному середовищі значною мірою залежить від стабільного підключення до інтернету. У випадках, коли зв'язок є нестабільним або обмеженим, це може серйозно заважати проведенню ключових сесій планування та координації.

Також не слід ігнорувати культурний опір: деякі зацікавлені сторони виявляють небажання відмовлятися від формальних, документально орієнтованих практик на користь більш гнучких, ітеративних і колаборативних підходів, які передбачають використання хмарних інструментів (Sarmah & Deka, 2020).

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до збору вимог (табл. 1).

Табл. 1. Основні відмінності між традиційним і хмарним підходами до збору вимог.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Характер процесу	Лінійний, послідовний, із фіксованими етапами.	Ітеративний, гнучкий, орієнтований на постійне оновлення.
Інструменти	Документи Word/Excel, офлайн-збори, електронна пошта.	Jira, Confluence, Miro, Figma, Google Workspace, Microsoft 365.
Взаємодія учасників	Переважно синхронна (зустрічі, дзвінки).	Асинхронна та синхронна взаємодія в реальному часі через

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
		хмарні сервіси.
Контроль версій	Ручне оновлення документів, складне відстеження змін.	Автоматичне збереження історії змін і контроль версій.
Доступність	Обмежена — залежить від фізичних зустрічей або корпоративної мережі.	Доступ з будь-якої локації через інтернет.
Гнучкість у зміні вимог	Низька: зміни часто потребують переробки документації.	Висока: вимоги оновлюються безпосередньо у хмарних системах.
Основні виклики	Повільна комунікація, відсутність прозорості.	Безпека даних, фрагментація інструментів, «втома від співпраці».

Джерело: розроблено автором на основі IEEE Computer Society (2014); Basili & Turner (1975); Google Cloud (2023); Amazon Web Services (n.d.); Jagli & Yeddu (2017); Sarmah & Deka (2020); NIST (2018).

Представлене порівняння демонструє, що традиційний підхід до збору вимог є лінійним і малогнучким, значною мірою залежить від фізичних зустрічей та ручного управління документацією, що ускладнює внесення змін, уповільнює комунікацію та підвищує ризик непослідовності версій. Хмарно-орієнтований підхід, навпаки, забезпечує ітеративність, централізований доступ і автоматизований контроль версій, що сприяє прозорості процесу, швидкому оновленню вимог та ефективній взаємодії розподілених команд. Водночас інтеграція великої кількості інструментів створює ризик фрагментації процесів, а робота з конфіденційними даними у хмарі підвищує вимоги до безпеки, зокрема до управління доступом, шифрування та контролю операцій. Динамічність хмарних сервісів також може спричинити появу нових загроз, таких як помилкові конфігурації, небажане розповсюдження даних або некоректні дозволи. Отже, хмарні сервіси істотно підвищують адаптивність і якість процесу збору вимог, однак ефективність такого підходу залежить від належної стандартизації, зрілості практик управління доступами та дотримання принципів хмарної безпеки.

Проектування системи

Після етапу збору вимог настає фаза проєктування системи, яка полягає у трансформації функціональних очікувань у структуру, придатну для технічної реалізації. Цей процес охоплює як високорівневу архітектуру, що включає компоненти служб, потоки даних і межі безпеки, так і деталізоване проєктування, зокрема схеми баз даних, опис програмного інтерфейу та моделювання поведінки інтерфейсу користувача (IEEE Computer Society, 2014). У традиційних локальних умовах проєктування зазвичай передбачає створення статичних архітектурних схем і ручне планування інфраструктури в межах обмежених ресурсів. Такий підхід часто обмежує гнучкість, масштабованість і адаптивність системи до змін у вимогах (Basili & Turner, 1975).

Вплив хмарних інструментів на проектування системи. Сучасні хмарні технології значно розширюють можливості архітектурного проєктування завдяки підтримці динамічності, масштабованості та автоматизації. Хмарні платформи забезпечують розробників засобами для оперативного створення архітектурних схем, підтримки спільної роботи над архітектурними рішеннями, програмного опису інфраструктури з можливістю її версіювання, а також формалізованої документації прийнятих рішень. Крім того, вони пропонують методологічну підтримку у вигляді фреймворків, які допомагають дотримуватись

принципів надійності, продуктивності, безпеки та оптимізації витрат у процесі проєктування. Зазначені можливості реалізуються за допомогою таких інструментів і сервісів, як AWS Architecture Diagrams, Azure Architecture Center, Google Cloud Architecture Framework, платформи Lucidchart, Draw.io, Miro, інструменти інфраструктури як код (Terraform, AWS CloudFormation, Pulumi), формати документації Architecture Decision Records (ADR), а також фреймворки Well-Architected Framework, запропоновані Amazon Web Services, Azure та Google Cloud Platform (Amazon Web Services, n.d.; Google Cloud, 2023).

Застосування хмарних інструментів дає змогу вирішити низку критичних недоліків традиційного підходу до проєктування. Вони усувають обмеження масштабованості завдяки нативній підтримці горизонтального масштабування, еластичності та мікросервісної архітектури (Jagli & Yeddu, 2017). Автоматизація через IaC дозволяє уникнути людських помилок, зменшуючи потребу у ручному налаштуванні інфраструктури. Також хмарні платформи сприяють модульному підходу до архітектури, що полегшує поступову еволюцію системи. Крім того, завдяки можливості створювати ізольовані середовища на вимогу, архітектори можуть оперативно тестувати рішення ще до початку основної реалізації, що суттєво прискорює зворотний зв'язок.

Водночас, деякі труднощі залишаються актуальними навіть у хмарному середовищі. Надмірне використання функціональності хмарних платформ може призвести до так званої надінженерії, коли архітектура стає надто складною без об'єктивної потреби (Jagli & Yeddu, 2017). Крім того, проєктна документація часто розпорошується між різними інструментами — від діаграм візуального моделювання до IaC-репозиторіїв і платформ командної співпраці, що ускладнює цілісне управління. Ще однією важливою проблемою є прогалини в знаннях: не всі команди мають відповідну експертизу в хмарних розподілених системах, особливо ті, хто звик до традиційних підходів (Sarmah & Deka, 2020).

Нові ризики визвані інтеграцією з хмарними сервісами. Інтеграція з хмарними сервісами також створює нові типи ризиків, які потребують уважного контролю. Одним із них є складність у прогнозуванні бюджету, оскільки хмарні моделі ціноутворення (такі як оплата за використання чи багаторівнева тарифікация) вимагають глибшого фінансового планування (Amazon Web Services, n.d.). Іншим аспектом є безпека, яка має бути врахована вже на етапі проєктування — від шифрування даних і управління ролями користувачів до налаштування шлюзів API (OWASP Foundation, 2023). Крім того, робота з IaC вимагає не лише технічних знань, а й постійного технічного обслуговування, оскільки несвоєчасне оновлення може призвести до накопичення технічної заборгованості (Jagli & Yeddu, 2017). Нарешті, залежність від конкретних сервісів одного постачальника (наприклад, AWS Lambda або Azure Functions) може обмежити можливість міграції на іншу платформу, що створює ризик "vendor lock-in" (Google Cloud, 2023).

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до проектування системи (табл. 2).

Табл. 2. Основні відмінності між традиційним і хмарним підходами до проектування системи.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Характер процесу проєктування	Статичне, послідовне проєктування з фіксованими архітектурними рішеннями.	Динамічне, гнучке проєктування з можливістю швидкої зміни архітектури.
Тип архітектури	Монолітна, з обмеженою масштабованістю.	Мікросервісна або хмарно-нативна, із підтримкою еластичності.
Інструменти проєктування	Локальні графічні редактори (Visio, Draw.io офлайн).	Хмарні платформи (AWS Architecture Center, Azure

		Diagrams, Lucidchart, Miro).
Опис інфраструктури	Ручне планування та документування.	Інфраструктура як код (Terraform, CloudFormation, Pulumi).
Автоматизація процесів	Мінімальна, налаштування виконуються вручну.	Висока — розгортання, тестування та моніторинг автоматизовані.
Співпраця команди	Обмежена, залежить від фізичних зустрічей або локальної мережі.	Спільна робота в режимі реального часу, хмарне зберігання артефактів.
Безпека	Додається на пізніх етапах проєкту.	Враховується на етапі архітектури (security by design).
Масштабованість	Обмежена апаратними ресурсами.	Гнучка — динамічне масштабування хмарних ресурсів.
Основні ризики	Обмежена адаптивність, повільна еволюція архітектури.	Надінженерія, vendor lock-in, потреба у високій кваліфікації команди.

Джерело: розроблено автором на основі IEEE Computer Society (2014); Basili & Turner (1975); Amazon Web Services (n.d.); Google Cloud (2023); Jagli & Yeddu (2017); Sarmah & Deka (2020); OWASP Foundation (2023).

Порівняння показує, що хмарно-орієнтований підхід до проєктування суттєво відрізняється гнучкістю, підтримкою мікросервісної архітектури та використанням інфраструктури як коду, що забезпечує автоматизацію, масштабованість і врахування безпеки вже на рівні архітектури. Традиційне проєктування лишається статичним, монолітним і залежним від ручного документування, що уповільнює розвиток системи. Хмарні інструменти на зразок AWS Architecture Center чи Terraform покращують колаборацію та пришвидшують ухвалення рішень, проте створюють і нові ризики — зокрема можливість помилок у IaC-конфігураціях, надмірних дозволів, витоку архітектурних артефактів або залежності від постачальника. Тому ефективність хмарного проєктування залежить від контрольованої складності та належного управління безпекою.

Розробка системи

На етапі розробки створений архітектурний проєкт трансформується в робоче програмне забезпечення. Розробники реалізують функціональність, пишуть код, інтегрують служби та перевіряють модулі відповідно до попередньо визначених вимог і технічних рішень. У цей період також здійснюється тестування окремих компонентів, рев'ю коду та контроль версій, що дозволяє підтримувати якість і стабільність програмного продукту (IEEE Computer Society, 2014). У традиційних підходах, особливо в локальних середовищах, процес розробки часто був обмежений обчислювальними ресурсами однієї машини, недостатньою взаємодією між членами команди та ручними сценаріями тестування і розгортання. Ці чинники сприяли появі затримок, проблем з узгодженістю середовищ і загальному зниженню продуктивності (Sarmah & Deka, 2020).

Вплив хмарних інструментів на етап розробки. Хмарні платформи забезпечують розробників широким спектром засобів, що сприяють спрощенню, стандартизації та автоматизації процесів створення програмного забезпечення. Вони дають змогу працювати в уніфікованих середовищах розробки з будь-якої точки доступу, інтегрувати механізми

безперервної інтеграції та доставки, централізовано управляти залежностями, своєчасно виявляти дефекти та уразливості, а також поетапно впроваджувати новий функціонал із можливістю контролю впливу на кінцевих користувачів. Реалізація цих можливостей здійснюється за допомогою хмарних інтегрованих середовищ розробки (наприклад, AWS Cloud9, GitHub Codespaces, JetBrains Space), інструментів CI/CD (GitHub Actions, GitLab CI/CD, AWS CodePipeline, Azure DevOps), сховищ артефактів (JFrog Artifactory, AWS CodeArtifact), засобів аналізу коду й безпеки (SonarCloud, Snyk, GitHub Dependabot) та систем керування функціональністю (LaunchDarkly, CloudBees) (OWASP Foundation, 2023).

Інтеграція хмарних технологій значною мірою долає багато традиційних бар'єрів у розробці. Однією з найважливіших переваг є усунення невідповідностей між середовищами. Використання хмарних IDE або контейнеризованих середовищ (наприклад, Docker, Dev Containers) дозволяє уникнути ситуацій, коли програма працює лише на локальній машині розробника. Також значно скорочується час на розгортання завдяки автоматизованим CI/CD-пайплайнам, що покращує частоту випуску нових версій та зменшує ймовірність людських помилок (Jagli & Yeddu, 2017). Хмарні репозиторії та платформи для співпраці, такі як GitHub чи Bitbucket, покращують взаємодію між членами команди завдяки можливості спільного редагування, рев'ю коду та контролю змін у реальному часі. Крім того, завдяки інтегрованим засобам аналізу безпеки, розробники можуть виявляти потенційні вразливості ще на етапі написання коду, знижуючи ризики для готового продукту (OWASP Foundation, 2023).

Незважаючи на численні покращення, деякі виклики залишаються актуальними і в хмарній розробці. Часті ітерації та високий темп розгортання можуть спонукати команди до тимчасових рішень, що накопичують технічний борг і знижують підтримуваність коду (IEEE Computer Society, 2014). Підтримка стабільної якості коду, незважаючи на наявність сучасних інструментів, як і раніше значною мірою залежить від внутрішньої дисципліни команди та чітких стандартів розробки. Крім того, новим членам команди часто складно приєднатися до проєкту без належної документації або стандартів інтеграції, що уповільнює процес адаптації.

Нові ризики визвані інтеграцією з хмарними сервісами. Інтеграція хмари у процес розробки також породжує нові ризики. Зокрема, різноманіття інструментів та практик — таких як контейнери, безсерверна архітектура, IaC — може перевантажити команди, особливо менш досвідчені (Sarmah & Deka, 2020). Надмірна автоматизація, хоч і корисна, іноді призводить до сліпої довіри до систем, які можуть пропустити критичні дефекти або невірно спрацювати без належної перевірки. Також зростає ризик пов'язаний із неправильними конфігураціями — наприклад, помилка у файлі Terraform або YAML для CI/CD-процесу може зробити застосунок вразливим (NIST, 2018). Нарешті, використання сервісів, орієнтованих на конкретного постачальника (як-от AWS Lambda, Firebase або Azure Functions), може призвести до залежності від однієї хмарної платформи, зменшуючи гнучкість та можливість переносу рішень у майбутньому.

Добавлено примечание ([2]): Розшифруйте абревіатуру.

Добавлено примечание ([3R2]): Це не абревіатура

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до розробки системи (табл. 3).

Табл. 3. Основні відмінності між традиційним і хмарним підходами до розробки системи.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Середовище розробки	Локальне, залежить від конфігурації окремих машин розробників.	Хмарне або контейнеризоване (GitHub Codespaces, AWS Cloud9, Docker).
Інтеграція та доставка	Ручне тестування й розгортання, довгі цикли релізів.	Безперервна інтеграція та доставка (CI/CD) з автоматизацією перевірок і релізів.
Керування	Відсутність централізованого	Централізовані сховища

залежностями	контролю, часті конфлікти версій.	артефактів (JFrog, AWS CodeArtifact).
Контроль версій та співпраця	Локальні репозиторії, обмежена командна взаємодія.	Хмарні платформи Git (GitHub, GitLab, Bitbucket) із можливістю спільного редагування та рев'ю коду.
Забезпечення якості	Тестування виконується після основної розробки, часто вручну.	Автоматизоване тестування, статичний аналіз коду (SonarCloud, Snyk).
Безпека	Впроваджується після етапу реалізації.	Інтегрована в процес розробки (security-as-code, shift-left security).
Гнучкість та масштабованість	Залежить від апаратних ресурсів розробника.	Динамічне масштабування, гнучкі середовища за запитом.
Типові інструменти	Локальні IDE (IntelliJ, Eclipse, Visual Studio).	Хмарні IDE (GitHub Codespaces, AWS Cloud9, JetBrains Space).
Основні переваги	Простота контролю локального процесу.	Висока швидкість розгортання, командна синхронізація, автоматизація.
Основні ризики	Несумісність середовищ, ручні помилки, затримки у випусках.	Перевантаження інструментами, конфігураційні помилки, vendor lock-in.

Джерело: розроблено автором на основі IEEE Computer Society (2014); Sarmah & Deka (2020); OWASP Foundation (2023); Jagli & Yeddu (2017); NIST (2018).

Порівняння демонструє, що перехід від локальних, ізольованих і ручних процесів до хмарно-орієнтованих моделей розробки не лише прискорює цикли доставки, а й суттєво змінює профіль безпеки. У традиційних середовищах ручні операції, відсутність централізованого контролю версій і залежність від локальної інфраструктури підвищують ризики людських помилок, дрейфу конфігурацій і неконсистентності середовищ. Хмарний підхід, навпаки, інтегрує безпеку в сам процес розробки через практики shift-left, security-as-code, автоматизований статичний і динамічний аналіз коду, що дає змогу виявляти вразливості значно раніше. Використання контейнеризованих середовищ, централізованих сховищ залежностей та CI/CD-пайплайнів забезпечує контрольованість, трасованість і меншу ймовірність помилок конфігурації. Проте посилена автоматизація й велика кількість сервісів водночас створюють нові загрози: зростає ризик конфігураційних вразливостей у хмарних ресурсах, залежності від постачальника, компрометації секретів у пайплайнах та некоректного налаштування прав доступу. Таким чином, хоча хмарний підхід значно підвищує рівень проактивної безпеки, його ефективність залежить від зрілого управління інфраструктурою, безперервного моніторингу та дотримання принципів мінімальних привілеїв і контрольованої автоматизації.

Тестування системи

Етап тестування є критично важливою частиною життєвого циклу розробки програмного забезпечення, оскільки забезпечує відповідність реалізованого функціоналу визначеним вимогам та стандартам якості. Тестування охоплює різні рівні перевірки, включаючи модульне, інтеграційне, системне, навантажувальне та безпекове тестування

(Google Cloud, 2023). Його метою є виявлення помилок, запобігання регресіям і забезпечення відповідності програмного продукту очікуванням користувачів та бізнесу ще до розгортання у виробничому середовищі. У традиційних моделях SDLC тестування часто відкладалося на завершальний етап циклу розробки та виконувалося вручну, що призводило до затримок, накопичення дефектів і зростання вартості виправлень (Microsoft, 2022). До того ж ізольованість тестових середовищ ускладнювала відтворення реальних умов, у яких буде працювати застосунок.

Вплив хмарних інструментів на тестування. Хмарні технології кардинально трансформували процес тестування, зробивши його більш автоматизованим, масштабованим та інтегрованим у загальну інфраструктуру розробки. Завдяки цьому стало можливим безперервне виконання перевірок при кожній зміні коду, забезпечення кросбраузерної та кросплатформної сумісності без локального налаштування середовища, імітація навантаження в умовах, наближених до виробничих, виявлення вразливостей ще на етапі розробки, а також тестування взаємодії з недоступними або нестабільними зовнішніми сервісами. Реалізація зазначених можливостей підтримується за допомогою інтеграції тестів у CI/CD-конвеєри (GitHub Actions, GitLab CI, Jenkins у хмарі), платформ для кросбраузерного тестування (BrowserStack, Sauce Labs), інструментів навантажувального тестування (Apache JMeter в AWS, Gatling Cloud, служби Azure Load Testing), засобів безпекового аналізу (Snyk, Checkmarx, OWASP ZAP) і сервісів віртуалізації API (WireMock Cloud, Mountebank) (Microsoft, 2022).

Використання хмарних рішень усуває цілу низку проблем, що були характерні для традиційного підходу до тестування. Зокрема, масштабованість хмарних ресурсів дозволяє виконувати масове паралельне тестування без обмежень, пов'язаних із локальними потужностями. Кросплатформне тестування за допомогою ферм пристроїв забезпечує ширше покриття сумісності, тоді як хмарне навантажувальне тестування дозволяє точно симулювати реальні сценарії використання. Крім того, CI/CD-конвеєри забезпечують миттєвий зворотний зв'язок після змін у коді, що прискорює виявлення та усунення дефектів (Jagli & Yeddu, 2017). Уніфіковані хмарні середовища для тестування також набагато краще імітують виробничі умови, ніж локальні конфігурації, що знижує ризик несподіваної поведінки програмного забезпечення після розгортання.

Попри значний прогрес, деякі проблеми залишаються актуальними навіть у хмарному контексті. Однією з них є нестабільність автоматизованих тестів: їхні результати можуть бути непослідовними та хибно позитивними через тимчасові залежності, фонові процеси або недостатню ізоляцію даних (IEEE Computer Society, 2014). Також з часом зростають витрати на підтримку автоматизованих тестів — у міру розширення функціональності додатку необхідно постійно оновлювати й адаптувати тести. І навіть у добре побудованих автоматизованих конвеєрах існує ризик неповного покриття — через часові обмеження або обмеження самих інструментів деякі критично важливі сценарії можуть залишитися недостатньо перевіреними.

Нові ризики визвані інтеграцією з хмарними сервісами. Перехід до хмарного тестування, хоч і відкриває нові можливості, водночас створює і нові ризики. Перш за все, зростає потреба в уважному поводженні з даними, щоб уникнути витоку конфіденційної інформації та дотримання законодавчих норм (наприклад, GDPR) NIST (2018). Також масштабні тести, особливо ті, що пов'язані з продуктивністю або наскрізною перевіркою (end-to-end), можуть вимагати значних обчислювальних ресурсів, що призводить до зростання вартості. Безпека тестових середовищ та CI/CD-систем є критично важливою, оскільки ці середовища містять як код, так і секрети, необхідні для тестування NIST (2018). Нарешті, складність інтеграції різноманітних хмарних інструментів у єдиний пайплайн може потребувати розробки спеціальних скриптів, конфігурацій або навіть оркестраційних систем, що збільшує технічну складність проекту (Sarmah & Deka, 2020).

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до тестування системи (табл. 4).

Табл. 4. Основні відмінності між традиційним і хмарним підходами до тестування системи.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Середовище тестування	Локальні машини, обмежені ресурси	Хмарні середовища, динамічне масштабування
Автоматизація	Тестування виконується переважно вручну	Інтеграція автоматизованих тестів у CI/CD
Навантажувальне тестування	Обмежене локальними ресурсами	Масштабоване, реалістичне навантаження через хмарні ресурси
Кросплатформність	Потребує ручної перевірки або емуляторів	Хмарні ферми пристроїв (BrowserStack, Sauce Labs)
Безпекове тестування	Виконується на пізніх етапах	Інтегроване у пайплайни CI/CD (Snyk, Checkmarx)
Керування даними	Тестові дані зберігаються локально	Централізоване керування в хмарі
Інтеграція інструментів	Вимагає ручного налаштування	Підтримується платформами (GitHub Actions, Jenkins Cloud)

Джерело: розроблено автором на основі Google Cloud (2023); Microsoft (2022); Jagli & Yeddu (2017); IEEE Computer Society (2014); NIST (2018); Sarmah & Deka (2020).

Порівняльний аналіз свідчить, що традиційне тестування, засноване на локальних ресурсах і ручних операціях, створює суттєві безпекові ризики: обмежена автоматизація ускладнює відтворюваність перевірок, затримує виявлення вразливостей і зміщує безпекове тестування на кінцеві етапи SDLC, коли виправлення є затратними й потенційно небезпечними для стабільності системи. Хмарно-орієнтована модель усуває ці недоліки завдяки інтеграції автоматизованих безпекових перевірок у CI/CD-процеси, можливості масштабувати середовища для моделювання реальних атак і використанню спеціалізованих інструментів Static Application Security Testing, Dynamic Application Security Testing, Software Composition Analysis та аналізу секретів, що значно підвищує ефективність виявлення загроз. Крім того, централізоване керування тестовими даними, логами й артефактами посилює контроль доступу та спрощує аудит, зменшуючи ризики несанкціонованого використання або витоку конфіденційної інформації. Водночас залежність від хмарних платформ підвищує вимоги до правильного налаштування прав доступу, сегментації середовищ і захисту CI/CD-пайплайнів, оскільки неправильні конфігурації або компрометація хмарних акаунтів можуть створити нові вектори атак. Таким чином, хмарне тестування значно підвищує рівень проактивної безпеки, але потребує вищої дисципліни у керуванні конфігураціями та зрілих DevSecOps-практик.

Розгортання

Розгортання — це один з завершальних етапів життєвого циклу розробки програмного забезпечення, під час якого продукт випускається у виробниче середовище та стає доступним для кінцевих користувачів. У традиційних моделях SDLC процес розгортання часто здійснювався вручну, епізодично, із попередньо запланованими простоями системи. Такий підхід був схильним до помилок, мав високий ризик збоїв і супроводжувався значними витратами на підтримку (NIST, 2018). З появою хмарних інструментів та практик DevOps стратегії розгортання зазнали суттєвих змін. Вони стали автоматизованими, швидкими та

більш безперервними, що дозволяє уникати простоїв, зменшити ризики та забезпечити високу доступність застосунків (Jagli & Yeddu, 2017).

Вплив хмарних інструментів на етап розгортання. Сучасні хмарні платформи відкривають широкі можливості для автоматизації та централізованого управління процесами розгортання програмного забезпечення в різних середовищах. Інфраструктура розгортання інтегрується у життєвий цикл розробки, забезпечуючи автоматичне збирання, тестування та доставку застосунків на кожному етапі, що сприяє зниженню кількості помилок, прискоренню виходу нових версій і покращенню надійності поставки. Підтримка різнорідних середовищ — від девелоперських до продукційних — реалізується шляхом конфігурації пайплайнів з урахуванням специфіки кожного оточення. Контейнеризація забезпечує уніфікацію процесу доставки, дозволяючи масштабувати обчислювальні ресурси, балансувати навантаження та автоматично відновлювати сервіси у випадку збоїв. Оркестраційні системи, зокрема ті, що базуються на Kubernetes, дозволяють централізовано керувати життєвим циклом контейнерів, автоматизувати розгортання, оновлення та моніторинг, зберігаючи стабільність і контроль над інфраструктурою. Реалізація зазначених можливостей здійснюється за допомогою сервісів безперервної інтеграції та доставки (CI/CD), таких як AWS CodeDeploy, Azure Pipelines, Google Cloud Deploy і GitHub Actions, а також систем оркестрації контейнерів, включаючи Kubernetes-платформи Azure Kubernetes Service, Amazon Elastic Kubernetes Service і Google Kubernetes Engine, та сервіс Amazon ECS.

Окрему нішу займають рішення для безсерверного розгортання, яке забезпечує автоматичне масштабування, подієво-орієнтовану активацію функцій та сплату лише за фактичне використання ресурсів. Це дає змогу зосередитись на бізнес-логіці, не витрачаючи ресурси на підтримку серверного середовища.

Інший ключовий компонент — інфраструктура як код (Infrastructure as Code, IaC), яка передбачає програмне описання всієї інфраструктури, включаючи мережеві компоненти, ресурси обчислення, служби зберігання даних та політики безпеки. Цей підхід забезпечує відтворюваність, контроль версій, аудит змін і автоматизацію процесів розгортання, що є особливо важливим для командної роботи та DevOps-практик. Зазначені можливості реалізуються за допомогою інструментів безсерверного розгортання, таких як AWS Lambda, Azure Functions і Google Cloud Functions, а також рішень IaC, включаючи Terraform, AWS CloudFormation та Pulumi.

Крім того, розгортання в хмарі підтримує сучасні стратегії доставки, такі як синьо-зелене (blue-green), канаркове (canary) і накатувальне (rolling) розгортання. Для їх реалізації застосовуються спеціалізовані засоби на кшталт Argo CD та Spinnaker, які дозволяють здійснювати поступове або паралельне впровадження змін без негативного впливу на користувачів.

Використання хмарних інструментів значною мірою усуває проблеми, притаманні традиційному підходу до розгортання. Зокрема, автоматизація процесів знижує ймовірність людських помилок і забезпечує послідовність дій у всіх середовищах (Sarmah & Deka, 2020). Завдяки інфраструктура-як-код (IaC) усувається проблема «дрейфу середовищ», коли середовища розробки, тестування та продакшн відрізняються між собою, що часто призводить до неочікуваної поведінки застосунків. Конвеєри CI/CD дають змогу пришвидшити випуск релізів, забезпечуючи їхню регулярність та відповідність вимогам гнучких методологій розробки.

Крім того, стратегічне планування розгортання дозволяє мінімізувати або повністю уникати простоїв під час оновлень. Використання методів поступового впровадження, таких як canary або blue-green, допомагає зменшити ризики, пов’язані із впровадженням змін у продуктивне середовище.

Попри значні переваги, деякі виклики залишаються актуальними навіть у контексті хмарного розгортання. Однією з таких проблем є складність відкоту змін у разі виявлення критичних помилок після релізу. Якщо не розроблено відповідний план дій або не створено резервних копій, відновлення працездатності може бути ускладнене. Крім того, збій у збірці

або неправильна конфігурація конвеєра CI/CD може заблокувати реліз і затримати вихід продукту на ринок (Sarmah & Deka, 2020).

Іншим серйозним викликом є питання безпеки. Зокрема, жорстко закодовані облікові дані або неналежне керування секретами у скриптах розгортання можуть призвести до витоку конфіденційної інформації або створити вразливості в системі.

Нові ризики визвані інтеграцією з хмарними сервісами. Інтеграція розгортання з хмарною інфраструктурою також створює нові ризики, які не були характерними для традиційних систем. Один із них пов'язаний із моніторингом витрат. Часте розгортання може призводити до утворення тимчасових середовищ або надмірного використання обчислювальних ресурсів, що потребує уважного контролю за бюджетом.

Автоматизація процесів та відкриті кінцеві точки API також збільшують поверхню атак. Якщо CI/CD-конвеєри або середовища не захищені належними засобами, такими як контроль доступу, аудит подій чи обмеження привілеїв, це може створити серйозні вразливості (Jagli & Yeddu, 2017). Ще одним викликом є розростання ланцюгів інструментів: для повноцінного функціонування автоматизованого розгортання потрібно інтегрувати велику кількість сервісів — керування кодом, тестування, моніторинг, інфраструктура — що може призвести до зростання складності підтримки.

Нарешті, організаційний аспект також не варто недооцінювати. Перехід до безперервної доставки вимагає не лише технічних змін, а й зміни культури всередині команди. Зокрема, підвищується відповідальність розробників за якість коду та релізів, а також необхідність довіри до автоматизованих процесів і практик DevOps.

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до розгортання системи (табл. 5).

Табл. 5. Основні відмінності між традиційним і хмарним підходами до розгортання системи.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Процес розгортання	Ручне, періодичне, з простоєм системи	Автоматизоване, безперервне (CI/CD)
Інфраструктура	Налаштовується вручну на фізичних або віртуальних серверах	Описується як код (IaC — Terraform, CloudFormation)
Середовище виконання	Фіксовані сервери, складність масштабування	Контейнери та оркестрація (Docker, Kubernetes)
Стратегії релізів	Одноразове оновлення з ризиком збоїв	Blue-green, canary, rolling оновлення
Безсерверні рішення	Відсутні або потребують серверного управління	AWS Lambda, Azure Functions, GCP Functions
Безпека	Ручне керування обліковими даними	Автоматизоване керування секретами, політики доступу
Моніторинг і відкат	Здійснюється після збою	Автоматизовані дашборди, алерти, rollback-процеси
Організаційна культура	Розділення обов'язків між Dev та Ops	DevOps — спільна відповідальність за релізи

Матеріал розроблено автором на основі таких джерел: Google Cloud (2023); Microsoft (2022); Jagli & Yeddu (2017); IEEE Computer Society (2014); NIST (2018); Sarmah & Deka (2020).

Порівняльний аналіз свідчить, що традиційне розгортання, виконуване вручну та на статично налаштованих серверах, створює значні безпекові ризики: ручне керування обліковими даними, відсутність контрольованого середовища для оновлень і одноетапні релізи підвищують імовірність людських помилок, витоку секретів та збоїв, які важко швидко локалізувати й відкотити. У таких моделях безпекові перевірки відбуваються після розгортання, а моніторинг є реактивним, що збільшує час виявлення та реагування на інциденти. Натомість хмарно-орієнтоване розгортання інтегрує безпеку безпосередньо в пайплайни CI/CD через автоматизоване керування секретами, політики найменших привілеїв, IaC з можливістю перевірки конфігурацій та механізми контролю цілісності. Використання контейнеризації та оркестрації забезпечує ізоляцію середовищ, а такі стратегії, як canary або blue-green, дозволяють мінімізувати ризики шляхом поступового введення змін і швидкої локалізації потенційних вразливостей. Автоматизовані дашборди, алерти та rollback-процедури скорочують «вікно атаки» та забезпечують оперативне реагування. Водночас збільшення кількості інтегрованих сервісів підвищує вимоги до правильного керування конфігураціями, захисту CI/CD-конвеєрів і перевірки IaC, оскільки помилка в конфігурації або компрометація хмарного облікового запису може масштабуватися на всю інфраструктуру.

Підтримка та моніторинг

Після завершення розгортання програмного забезпечення починається етап обслуговування та моніторингу, який триває протягом усього життєвого циклу системи. Основною метою цього етапу є підтримання безперебійної роботи програмного забезпечення, своєчасне усунення помилок, оптимізація продуктивності, забезпечення оновлень і відповідності змінним потребам користувачів. Це включає постійний моніторинг працездатності, аналіз логів і контроль за використанням ресурсів, що дозволяє підтримувати відповідність програмного забезпечення сучасним вимогам безпеки та технічним стандартам (IEEE Computer Society, 2014).

У традиційних підходах обслуговування зазвичай здійснювалося вручну — спеціалісти аналізували журнали подій, вирішували проблеми по мірі їх виникнення та періодично виконували оновлення. Однак такі реактивні методи часто виявлялися неефективними: затримки з реагуванням на інциденти призводили до тривалих простоїв, а зниження продуктивності могло залишатися непоміченим протягом тривалого часу (Amazon Web Services, n.d.).

Вплив хмарних інструментів на етап підтримки. Сучасні хмарні середовища створюють розширені можливості для автоматизації технічного обслуговування, спостереження за станом систем та забезпечення їх високої доступності. Завдяки вбудованим механізмам моніторингу й управління інцидентами, інженери можуть у реальному часі відстежувати ключові метрики, виявляти відхилення від нормативних значень, проводити аналіз причин збоїв і автоматизувати реагування на критичні події. Централізоване логування, збирання телеметричних даних, сповіщення, автоматичне масштабування і відновлення, а також керування патчами й конфігураціями дозволяють підтримувати стабільну роботу систем навіть у складних динамічних середовищах. Для реалізації зазначених можливостей використовуються спеціалізовані сервіси, зокрема Amazon CloudWatch, Azure Monitor, Google Cloud Operations Suite, Datadog, Splunk, PagerDuty, AWS Systems Manager, Azure Automation та AWS Patch Manager.

Завдяки хмарним рішенням вдалося подолати низку обмежень, характерних для традиційного обслуговування. Однією з головних переваг є усунення обмеженої видимості — централізовані інформаційні панелі та системи логування дозволяють отримувати цілісну картину стану системи. Моніторинг у реальному часі дозволяє оперативно виявляти потенційні збої ще до того, як вони вплинуть на користувачів, а автоматичне масштабування і механізми самовідновлення значно зменшують потребу в постійному ручному втручанні. Крім

того, автоматизація рутинних завдань, як-от оновлення конфігурацій або застосування патчів, робить технічне обслуговування більш стабільним і передбачуваним.

Попри очевидні переваги, використання хмарних інструментів не позбавляє команд певних викликів. Однією з поширених проблем є велика кількість помилкових сповіщень: неякісно налаштовані правила моніторингу можуть генерувати надмірну кількість некритичних повідомлень, що відволікає інженерів і призводить до "алертної втоми" (Sarmah & Deka, 2020). Аналіз першопричин збоїв також залишається складним завданням, особливо в умовах розподілених систем, де інциденти можуть мати непрямі й непередбачувані наслідки. Додатково, постійний моніторинг і чергування на виклик можуть призвести до вигорання та зниження ефективності команд підтримки.

Нові ризики визвані інтеграцією з хмарними сервісами. Використання хмарних сервісів у сфері обслуговування й моніторингу створює нові загрози, з якими необхідно рахуватись. Однією з них є перевантаження інструментами — надмірна кількість сервісів і платформ ускладнює їх інтеграцію та управління, а також створює ризик дублювання функцій. Безпека операційних даних також набуває особливої актуальності: журнали, телеметрія та аналітичні звіти можуть містити конфіденційну інформацію, яка потребує належного захисту відповідно до політик доступу та норм захисту даних (OWASP Foundation, 2023). Крім того, залежність від стабільності та доступності хмарних сервісів створює додаткову вразливість — у разі збоїв провайдера функціональність моніторингу або реагування може бути порушена. Окремо варто згадати динамічну природу хмарного середовища: ефемерні інстанси, як-от контейнери або безсерверні функції, ускладнюють традиційні методи трасування, що потребує використання нових підходів до налагодження й спостереження.

Окрім описаних переваг і викликів, доцільно узагальнити основні відмінності між традиційним і хмарним підходами до підтримки та моніторінгу (табл. 6).

Табл. 6. Основні відмінності між традиційним і хмарним підходами до підтримки та моніторінгу системи.

Критерій	Традиційний підхід	Хмарно-орієнтований підхід
Характер підтримки	Реактивна — усунення проблем після їх виникнення.	Проактивна — автоматичне виявлення та попередження інцидентів.
Моніторинг системи	Ручний аналіз логів, періодичні перевірки.	Безперервний моніторинг у реальному часі через сервіси (CloudWatch, Azure Monitor, Datadog).
Керування інцидентами	Виконується вручну, залежить від чергових спеціалістів.	Автоматизоване сповіщення, аналіз причин і оркестрація реакцій через PagerDuty, Splunk тощо.
Логування та телеметрія	Децентралізоване зберігання логів, складнощі з аналітикою.	Централізоване логування, збір телеметрії та інтегрована аналітика у хмарі.
Масштабованість і доступність	Обмежена ресурсами локальної інфраструктури.	Автоматичне масштабування, механізми самовідновлення та балансування навантаження.
Оновлення та патчі	Застосовуються вручну, часто під час простоїв.	Автоматичне або планове оновлення через сервіси на

		кшталт AWS Systems Manager, Azure Automation.
Видимість системи	Часткова, залежить від доступу до окремих компонентів.	Повна завдяки інтегрованим дашбордам і спільним консолям спостереження.
Безпека операційних даних	Мінімальні політики доступу, ручне шифрування.	Розмежування доступів, контроль секретів, шифрування даних на рівні сервісу.
Основні виклики	Затримки у виявленні помилок, тривалі простої, ручна діагностика.	Перевантаження сповіщеннями, складність інтеграції інструментів, ризики доступності сервісів провайдера.

Джерело: матеріал розроблено автором на основі таких джерел: IEEE Computer Society (2014); Amazon Web Services (n.d.); Sarmah & Deka (2020); OWASP Foundation (2023).

Порівняльний аналіз показує, що традиційні моделі підтримки та моніторингу характеризуються низьким рівнем безпеки через реактивний підхід: інциденти виявляються лише після їх прояву, логування здійснюється локально й фрагментовано, а ручне застосування патчів створює вікна вразливості та ризик помилок конфігурації. Обмежена видимість системи ускладнює вчасне виявлення загроз, а відсутність централізованого контролю над доступом підвищує ймовірність несанкціонованих змін. Хмарно-орієнтований підхід, навпаки, вбудовує безпеку у сам процес підтримки: безперервний моніторинг у реальному часі, централізоване логування та збір телеметрії дозволяють значно швидше виявляти аномалії, підозрілу активність та потенційні атаки. Автоматизоване керування інцидентами, механізми самовідновлення, контроль секретів і політики найменших привілеїв зменшують людський фактор і підвищують загальну стійкість системи. Водночас хмарна модель породжує свої безпекові ризики — зокрема, залежність від доступності сервісів провайдера, збільшення поверхні атаки через численні інтеграції та ризик витоку операційних даних за умови неправильного налаштування доступів. Отже, хоча хмара суттєво підсилює можливості безпечної експлуатації систем, її ефективність залежить від грамотної конфігурації, ретельного управління правами та впровадження узгодженої політики захисту операційних даних.

Безпека в хмарному *SDLC*

Безпека є критично важливим аспектом на всіх етапах життєвого циклу розробки програмного забезпечення (SDLC), і її значущість ще більше посилюється в контексті хмарних обчислень. Це зумовлено розширенням площі потенційного впливу (attack surface), розподіленими архітектурами, моделлю спільної відповідальності та високою динамічністю інфраструктури (Amazon Web Services, n.d.). У зв'язку з цим відбувається еволюція класичного SDLC у більш комплексну модель — Secure Software Development Life Cycle (SSDLC), яка передбачає інтеграцію безпекових вимог, перевірок і контролів на кожному етапі розробки.

Одним з фундаментальних принципів SSDLC є концепція Shift Left Security, яка полягає у зміщенні фокусу безпеки ліворуч на часовій шкалі SDLC — тобто на більш ранні етапи проєктування, планування та розробки. Замість реактивного виявлення вразливостей на пізніх етапах (тестування або пост-фактум після розгортання), даний підхід орієнтований на превентивне виявлення та усунення ризиків ще на фазі написання коду. Це досягається шляхом впровадження статичного аналізу безпеки (SAST), перевірки залежностей (SCA), інструментів перевірки конфігурацій та політик доступу, а також навчання розробників безпечним практикам кодування. Наприклад, використання таких сервісів, як Snyk,

Checkmarx, GitHub Dependabot, Trivy або Bridgecrew, дозволяє інтегрувати безпекові перевірки безпосередньо в CI/CD-пайплайни, забезпечуючи автоматичне виявлення вразливостей ще до того, як код потрапить у production-середовище (Aziz & Ahmad, 2016).

Таким чином, безпека стає не ізольованою відповідальністю окремих команд, а невід'ємною частиною усієї інженерної культури — від розробника до DevOps-інженера. Shift Left не лише знижує загальні витрати на усунення дефектів, але й підвищує рівень відповідності регуляторним вимогам, покращує якість продукту та мінімізує бізнес-ризики.

У цьому розділі наведено поетапний розподіл міркувань безпеки, інструментів і проблем у контексті хмарного SDLC.

Безпека на етапах SDLC

Збір і аналіз вимог. На етапі збирання та аналізу вимог у межах SSDLC особливу увагу приділяють визначенню вимог до безпеки, дотриманню нормативно-правових актів, а також моделюванню потенційних загроз. У хмарному контексті цей процес доповнюється використанням спеціалізованих стандартів і інструментів. Зокрема, як основу для формування базового рівня вимог безпеки застосовуються рекомендації NIST (National Institute of Standards and Technology, 2018) та CIS Benchmarks, які визначають найкращі практики з урахуванням специфіки хмарної інфраструктури. Для моделювання загроз можуть використовуватись хмарні інструменти, як-от Microsoft Threat Modeling Tool, що дозволяє систематично ідентифікувати можливі вектори атак. Такий підхід сприяє зменшенню ризиків, пов'язаних з нечітким безпековим покриттям або невідповідністю нормативним вимогам. Утім, у хмарному середовищі виникають і нові виклики, зокрема потреба транслювати угоди про рівень обслуговування (SLA) та модель спільної відповідальності між замовником і хмарним провайдером у конкретні вимоги до розробки програмного забезпечення (Google Cloud, 2023). Додатково постає завдання досягти узгодженого розуміння хмароспецифічних загроз серед усіх залучених команд.

Проектування системи. На етапі проєктування системи ключовим акцентом є створення архітектури з урахуванням безпекових протоколів, принципу мінімальних привілеїв (least-privilege) та ефективних механізмів захисту даних. В хмарному середовищі це передбачає ретельне планування управління ідентифікацією та доступом (IAM), що дозволяє контролювати та обмежувати права користувачів і сервісів відповідно до їхніх потреб (Microsoft, 2022) Крім того, важливо впроваджувати надійні стратегії шифрування як для збереження, так і для передачі даних, що забезпечує їх конфіденційність і цілісність. Для забезпечення безпечної мережевої архітектури застосовуються віртуальні приватні мережі (VPC), брандмауери та інші засоби ізоляції трафіку, які знижують ризик несанкціонованого доступу. Такий підхід дозволяє мінімізувати ризики, пов'язані з надмірними правами доступу або використанням слабких стандартів шифрування. Водночас у хмарних системах виникають нові виклики, зокрема управління безпекою у багаторегіональних та багатохмарних середовищах, а також забезпечення комплексного захисту гібридних архітектур, які поєднують локальні та хмарні ресурси.

Розробка системи. На етапі написання коду забезпечується створення безпечного програмного забезпечення шляхом запобігання вразливостям та ефективного керування секретами. Для цього застосовуються можливості автоматичного статичного аналізу коду, що дозволяє виявляти потенційні уразливості на ранніх стадіях розробки, а також рішення для безпечного зберігання і управління конфіденційними даними, які виключають необхідність жорсткого кодування облікових даних. Крім того, активне використання інструментів для сканування вразливостей у залежностях забезпечує своєчасне виявлення та усунення ризиків, пов'язаних із застосуванням сторонніх бібліотек. До прикладів таких інструментів належать Static Application Security Testing (SAST) системи, зокрема SonarCloud і Checkmarx, сервіси для керування секретами, як AWS Secrets Manager і HashiCorp Vault, а також рішення для моніторингу вразливостей залежностей, зокрема Snyk та Dependabot. Запровадження цих практик сприяє зниженню ймовірності помилок при впровадженні коду та виключенню

жорстко закодованих облікових даних, проте водночас зростає складність управління ризиками, пов'язаними з масштабним використанням відкритого програмного забезпечення, що потребує системного та комплексного підходу (OWASP Foundation, 2023).

Тестування системи. Процес перевірки засобів контролю безпеки в хмарних середовищах передбачає використання можливостей динамічного тестування безпеки додатків, що дозволяє виявляти вразливості під час виконання програмного забезпечення. Крім того, застосовуються методи хмарного сканування вразливостей, які забезпечують автоматичний аналіз налаштувань і конфігурацій інфраструктури з метою виявлення потенційних загроз. Використання fuzz-тестування та sandbox-середовищ дозволяє імітувати атаки у контрольованому ізольованому середовищі, що підвищує точність оцінки безпеки. Ці підходи сприяють зниженню ризиків, пов'язаних із неправильними налаштуваннями середовища виконання, а також відкритими API чи ендпоінтами. Водночас виникають нові виклики, такі як необхідність симуляції реалістичних атак у ефемерних інфраструктурах та забезпечення повного тестового покриття змінних хмарних ресурсів. До інструментів, що реалізують зазначені можливості, належать Dynamic Application Security Testing (DAST) системи, хмарні сканери вразливостей на кшталт AWS Inspector, а також рішення для fuzz-тестування і управління sandbox-середовищами.

Розгортання. Безпечне управління конвеєрами розгортання передбачає інтеграцію механізмів контролю доступу та безпекових перевірок у процес налаштування інфраструктури, що здійснюється за допомогою підходу Infrastructure as Code (IaC). Завдяки автоматизованим перевіркам конфігурацій IaC можна своєчасно виявляти та усувати помилки, що запобігає виникненню конфігураційних дрейфів і неавторизованому розгортанню коду. Крім того, застосування практик ротації секретів у CI/CD конвеєрах підвищує безпеку зберігання та використання конфіденційних даних. Впровадження концепції незмінної інфраструктури (immutable infrastructure) сприяє підтримці стабільності та повторюваності середовищ розгортання. Водночас, у процесі посилення безпеки конвеєрів CI/CD виникають нові виклики, зокрема пов'язані з потенційними помилками у налаштуванні шаблонів IaC, які можуть призводити до вразливостей. Для реалізації цих можливостей застосовуються спеціалізовані інструменти безпекових перевірок IaC, такі як tfsec і Checkov, а також механізми автоматизованої ротації секретів у CI/CD.

Підтримка та моніторинг. Постійний контроль вразливостей, виявлення вторгнень та аудит відповідності до нормативних вимог забезпечують комплексний підхід до підтримки безпеки хмарної інфраструктури. Автоматизовані рішення для управління інформацією та безпековими подіями (SIEM) дозволяють збирати, аналізувати та корелювати дані в режимі реального часу, що сприяє оперативному виявленню загроз і мінімізації часу перебування зловмисників у системі. Крім того, автоматизація процесів керування виправленнями допомагає своєчасно усувати виявлені вразливості, знижуючи ризики експлуатації систем. Постійний моніторинг відповідності, що реалізується через інструменти контролю конфігурацій та політик безпеки, забезпечує відповідність нормативним вимогам і внутрішнім стандартам. Проте, активне використання таких систем створює нові виклики, серед яких — ризик інформаційної перевантаженості через надмірні попередження та телеметрію, а також необхідність швидко адаптуватися до змін у хмарних сервісах і нових вразливостей. Для підтримки цих можливостей використовуються хмарні SIEM-системи, такі як AWS GuardDuty і Azure Sentinel, інструменти автоматизації керування виправленнями, а також сервіси для постійного моніторингу відповідності, наприклад AWS Config і Prisma Cloud.

Принципи хмарної безпеки на практиці. У практичному застосуванні принципи хмарної безпеки вимагають комплексного підходу, що враховує особливості хмарних платформ, динамічність інфраструктури та масштабованість систем. Ефективна безпека в хмарі потребує інтеграції багаторівневих заходів, починаючи від контролю доступу і автентифікації, закінчуючи моніторингом і автоматизацією управління інцидентами. Важливим аспектом є адаптація традиційних методів захисту до особливостей хмарних сервісів, де ресурси динамічно створюються, змінюються та знищуються. Відповідно,

реалізація безпекових практик повинна бути гнучкою та автоматизованою, щоб мінімізувати людський фактор і забезпечити швидке реагування на загрози. Ключовими складовими є застосування принципів модульності, стандартизації та інфраструктури як коду, що дозволяє створювати повторювані, контрольовані і безпечні середовища розгортання.

Модель спільної відповідальності (Shared Responsibility Model) (Amazon Web Services, n.d.) є фундаментальним підходом до розподілу обов'язків між хмарним провайдером і користувачем у сфері безпеки. Хмарний провайдер відповідає за захист інфраструктури, яка включає фізичні сервери, мережеве обладнання, центри обробки даних, а також за базовий рівень безпеки сервісів, які він надає. Зі свого боку, користувач або команда розробки відповідає за безпеку застосунків, даних, конфігурацій доступу і налаштувань у межах орендованих ресурсів. Чітке розуміння цього розподілу обов'язків є критично важливим для уникнення прогалин у безпеці, адже неправильно інтерпретовані межі відповідальності можуть призвести до вразливостей і компрометації систем. Відтак, команди повинні активно впроваджувати політики безпеки, налаштовувати контроль доступу і відповідально ставитися до захисту своїх компонентів, враховуючи динамічну природу хмарних платформ.

Архітектура нульової довіри (Zero Trust Architecture) (National Institute of Standards and Technology, 2020) передбачає радикальне перегляд традиційних підходів до безпеки, які базувалися на довірі до внутрішньої мережі. У Zero Trust кожен запит, незалежно від його джерела, вважається потенційно небезпечним і проходить сувору автентифікацію та авторизацію. Ця архітектура ґрунтується на принципі «least privilege» (Saltzer & Schroeder, 1975), що обмежує користувачів і сервіси лише необхідними правами для виконання їхніх завдань. Впровадження Zero Trust в хмарних середовищах включає багатофакторну автентифікацію (MFA), постійний моніторинг поведінки користувачів і пристроїв, а також сегментацію мережі для мінімізації потенційних точок доступу зловмисників. Такі заходи підвищують стійкість систем до внутрішніх і зовнішніх загроз, знижують ризик несанкціонованого доступу та поширення атак всередині інфраструктури.

Підхід «безпека як код» (Security as Code) (Das & Chu, 2023) полягає у застосуванні принципів розробки програмного забезпечення до процесів управління безпекою, зокрема конфігурацій, політик і процедур. Це означає, що всі безпекові налаштування і політики формулюються у вигляді конфігураційних файлів, які зберігаються у системах контролю версій, тестуються автоматизованими засобами і розгортаються за допомогою інструментів автоматизації. Такий підхід забезпечує прозорість, відтворюваність і контроль змін, а також дозволяє швидко виявляти помилки і несумісності у безпекових налаштуваннях до їхнього застосування у продуктивних середовищах. Безпека як код є ключовим елементом сучасних DevSecOps (Kim, Behr, & Spafford, 2013) практик, що інтегрують безпеку безпосередньо у процес розробки і доставки програмного забезпечення, підвищуючи загальний рівень захищеності систем.

Підсумок. Безпека в хмарному SDLC — це не окремий крок, а постійна інтегрована практика. У той час як хмарні інструменти та автоматизація покращують видимість, швидкість реакції та відповідність вимогам, вони також створюють нові загрози та вимагають глибокого розуміння конфігурацій конкретної платформи. Проактивний, вбудований підхід до безпеки має важливе значення для забезпечення стійкості, конфіденційності та довіри до хмарних додатків (Aziz & Ahmad, 2016).

У ході дослідження було встановлено, що використання хмарних сервісів суттєво змінює підходи до забезпечення безпеки на різних етапах життєвого циклу розробки ПЗ, зокрема, підвищується автоматизація, гнучкість, інтеграція безпеки на ранніх етапах, але одночасно з'являються нові ризики (наприклад, залежність від постачальника, конфігураційні вразливості, інтеграційні складнощі).

Feio et al. (2024) досліджували фреймворк DevSecOps під кутом безперервного тестування безпеки в CI/CD пайплайнах. Вони встановили, що застосування цього підходу дозволяє раннє виявлення вразливостей в реальних проектах. Згідно з їхнім дослідженням, організації, які впровадили безперервне тестування, значно скоротили кількість критичних

Добавлено примечание ([4]): Обговорення виділяється в окремий розділ тільки при дотриманні умови мінімального обсягу >1000 слів. В зв'язку з цим Ваше Обговорення об'єднано з результатами.

помилок. Це підтверджує одну з тез про те, що хмарні практики, інтегровані на етапі розробки й тестування, підвищують безпеку. Однак Feio et al. одразу вказують на те, що інтеграція інструментів має бути ретельною, що підтверрезультатах підкреслювали, що недостатня інтеграція та фрагментація інструментів можуть негативно впливати на безпеку.

W. Joseph (2025) досліджували вплив автоматизації безпеки та CI у хмарно-нативних архітектурах. Автор звертає увагу на те, що перехід до мікросервісів, контейнеризації та оркестрації істотно збільшує складність захисту, але правильне впровадження DevSecOps дозволяє цю складність контролювати. У висновках зазначено, що хмарні сервіси дають переваги, але супроводжуються новими ризиками — такими як неправильна конфігурація контейнерів чи оркестраторів. Таким чином, результати Joseph підтверджують наші спостереження.

Verdet et al. (2023) аналізували практики безпеки у скриптах IaC-конфігурації (Terraform та ін.). Вони виявили, що хоча політики доступу є найчастіше реалізовані, шифрування «at rest» значно відстає. У висновках підкреслено, що хмарні інструменти дозволяють автоматизувати інфраструктуру як код, але існує проблема неправильної конфігурації — це прямо корелює з висновком Verdet et al. Тобто їхнє дослідження служить емпіричним підкріпленням нашої тези про ризики провадження IaC.

William (2023) розглядали, як DevSecOps сприяє створенню стійких хмарно-нативних систем через раннє виявлення вразливостей та безперервний захист. Вони вказують, що саме інтеграція безпеки «зліва» (shift-left) і «справа» (secure-right) є ключовою. У дослідженні аналогічно відзначено, що безпека має бути закладена ще на етапі проєктування та інтегрована безперервно. Таким чином, висновки William доповнюють наші результати.

Myrbakken et al. (2025) здійснили багатомасштабний огляд практик DevSecOps та інструментів, виявивши 39 ключових практик і 114 інструментів, які асоціюються із ними. Вони акцентують увагу, що хоча інструменти доступні, організації часто не досягають зрілості у практиці безпеки. Результати показують аналогічну проблему: ми виявили, що організаційна культура і навчання команд є ще одним бар'єром на шляху до ефективного застосування хмарних безпечних SDLC-процедур. Отже, висновки Myrbakken et al. узгоджуються.

Leshchenko et al. (2024) запропонували розширену модель DevSecOps, яка містить етапи безпечного виведення з експлуатації, безпекового управління, частого аудиту та інноваційну безпеку. Вони наголошують, що існує розрив у типових моделях між фазами розробки та експлуатації. У висновках також зазначено, що хмарні моделі потребують глибшої інтеграції між розробкою, тестуванням, розгортанням і підтримкою. Тобто, модель Leshchenko et al. підсилює тезу про цілісність процесу безпеки.

N. Gadani, (2024) охоплює аналіз викликів безпеки у хмарній розробці (misconfigurations, недостатній контроль API, слабка ідентифікація). Ці виклики перегукуються з висновками — зазначено, що конфігураційні помилки, залежність від постачальника, распорошеність інструментів є значущими проблемами. Таким чином, дослідження підтверджує висновки.

W. Umeugo (2023) (для малих / середніх підприємств) досліджували впровадження SSDLC і вказували, що саме організаційна зрілість і культура відіграють ключову роль. У висновках відзначено, що недостатня кваліфікація команд або опір змінам (особливо в традиційних підходах) є бар'єром — це узгоджується з результатом даної праці.

M. Pranav (2025a) аналізують автоматизацію як «силу-двигун» наступного покоління DevSecOps. Вони пришвидшують впровадження безпеки, але вказують, що надмірна автоматизація може призвести до «слепої» довіри до систем. У цій роботі також зафіксовано схожу проблему — «алертна втома», надмірні сповіщення, а також ризик автоматизації без критичного контролю. Отже, це підтверджує цю думку.

M. Pranav, (2023b) пропонує модель адаптивної безпеки, яка базується на циклі MAPE-K і інтегрує безпеку в SDLC, постійно адаптуючись до змін. Ця модель доповнює висновки про те, що хмарні середовища потребують не просто інтеграції безпеки, але й здатності до

адаптації. Тому ця точку зору вважається слушною, і вона дає напрямок подальших досліджень.

Можна зробити висновок, що більшість сучасних публікацій підтверджують висновки про позитивний вплив хмарних сервісів на безпечну розробку ПЗ — вони визнають важливість DevSecOps, автоматизації, інтеграції безпеки, використання IaC та хмарних архітектур. Водночас деякі роботи розширюють ці тези: наприклад, Leshchenko et al. наголошують на фазі виведення з експлуатації, Alidoost Nia — на адаптивності, Myrbakken et al. — на інструментальній базі. Це дослідження узагальнює ці аспекти і додає фокус на специфічні ризики хмарних моделей (vendor lock-in, фрагментація інструментів, «алертна втома») у контексті SSDLC.

Однак варто зазначити, що в деяких роботах акцент зроблений скоріше на загальні моделі DevSecOps, ніж на специфіку хмарних сервісів у контексті SDLC. Наприклад, дослідження SSDLC для SME (2023) орієнтоване на організаційну культуру, а не на хмарну архітектуру. У той же час це дослідження має більш вузький фокус саме на хмарному середовищі та його впливі на етапи SDLC з точки зору безпеки — це дає певну унікальність. Також помітно, що кількість емпіричних досліджень у цьому напрямку (особливо із вимірюваннями у виробничих середовищах) все ще обмежена — вирізняються роботи Feio et al. (2024) чи William (2023), але більшість є оглядовими чи концептуальними. Це підтверджує наш висновок про необхідність подальших досліджень й практичних кейсів.

На основі порівняння з літературою можна зробити висновок, що впровадження хмарно-орієнтованого SSDLC із акцентом на безпеку має значний потенціал.

Висновки

Хмарні платформи забезпечують високий рівень масштабованості та еластичності, що дозволяє динамічно виділяти ресурси відповідно до потреб програмного забезпечення. Це допомагає створювати стабільні, високопродуктивні середовища для розробки та тестування, усуваючи інфраструктурні вузькі місця та забезпечуючи плавну роботу навіть під час пікових навантажень.

Автоматизація процесів і підтримка CI/CD у хмарних середовищах дозволяють автоматично виконувати збирання, тестування й розгортання застосунків, скорочуючи цикл доставки та зменшуючи кількість помилок. Разом із вбудованими механізмами безпеки — такими як керування ідентифікацією, зберігання секретів, контроль відповідності та захист під час виконання — це забезпечує можливість інтегрувати безпеку на ранніх етапах розробки, знижуючи ризик критичних уразливостей. Крім того, хмарні платформи покращують співпрацю завдяки централізованим системам контролю версій і відстеженню задач у реальному часі, що полегшує взаємодію віддалених команд. Інструменти моніторингу та журналювання в реальному часі підсилюють ці переваги, забезпечуючи раннє виявлення інцидентів і проактивне реагування на проблеми продуктивності.

Економічна ефективність також є вагомим аргументом на користь хмарного SDLC. Моделі оплати за фактичне використання знижують стартові витрати та дозволяють гнучко розподіляти бюджет відповідно до реальних потреб проєкту.

Разом із тим, хмарно-орієнтований SDLC супроводжується низкою викликів. Одним із них є vendor lock-in, коли залежність від окремих сервісів ускладнює подальшу міграцію. Складність конфігурацій, нечітке розуміння моделі спільної відповідальності та динамічність хмарних середовищ створюють додаткові ризики. Якщо не здійснювати регулярного контролю ресурсів, витрати можуть зростати непередбачувано. Крім того, необхідність інтегрувати та підтримувати велику кількість інструментів вимагає підвищеної кваліфікації та адаптації команд. Перехід до хмарних методів розробки часто потребує змін у культурі, ролях і процесах.

Хмарний SSDLC вирізняється модульністю: організації можуть поступово впроваджувати окремі сервіси, не змінюючи весь життєвий цикл одночасно. Наприклад, можна використовувати хмарні IDE для розробки, зберігаючи локальні конвеєри; застосовувати хмарне тестування без перенесення репозиторіїв; впроваджувати IaC лише для

окремих підсистем. Такий підхід мінімізує ризики та дозволяє коригувати темп модернізації відповідно до бізнес-потреб.

Узагальнюючи, дослідження показує, що перехід до хмарного SSDLC є не лише технологічною міграцією, а стратегічною трансформацією процесів розробки. Хмарні інструменти забезпечують команди гнучкістю, стійкістю й розширеними засобами безпеки, але одночасно вимагають еволюції практик, навичок і підходів. Успіх залежить від того, наскільки ефективно організації зможуть поєднати інновації з контролем, модульність — з інтегрованістю, а швидкість — із безпекою. Хмара залишається потужним інструментом, але її ефективність визначається тим, як саме вона використовується. У подальших дослідженнях доцільно зосередитись на кількісному вимірюванні впливу (наприклад, зменшення кількості вразливостей, скорочення часу релізу, зниження витрат на помилки) у різних хмарних сценаріях, а також на вивченні культури й навчання команд у контексті хмарних безпечних розробок.

References/Список використаних джерел

1. IEEE Computer Society. (2014). *Guide to the software engineering body of knowledge (SWEBOK V3.0).* IEEE Computer Society Press. https://www.computer.org/education/bodies-of-knowledge/software-engineering
2. Basili, V. R., & Turner, A. J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering, SE-1*(4), 390–396.
3. Google Cloud. (2023). *Best practices for secure software supply chains in Google Cloud.* https://cloud.google.com/architecture/secure-software-supply-chain
4. Amazon Web Services. (2022). *AWS Well-Architected Framework – Security Pillar.* https://docs.aws.amazon.com/wellarchitected/latest/security-pillar
5. National Institute of Standards and Technology. (2018). *Framework for improving critical infrastructure cybersecurity* (Version 1.1). https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf
6. OWASP Foundation. (2023). *OWASP secure SDLC cheat sheet.* https://cheatsheetseries.owasp.org/cheatsheets/Secure_SDLC_Cheat_Sheet.html
7. Sarmah, A., & Deka, G. C. (2020). Cloud computing and DevOps: A perfect pair for cloud-native application development. In G. C. Deka (Ed.), *Applications of cloud computing* (pp. 129–144). Springer. https://doi.org/10.1007/978-981-15-1312-0_7
8. Jagli, D., & Yeddu, S. (2017). CloudSDLC: Cloud software development life cycle. International Journal of Computer Applications, 168(8), 6–10. https://doi.org/10.5120/ijca2017914468
9. Microsoft. (2022). *Zero Trust security model.* https://www.microsoft.com/en-us/security/business/zero-trust
10. Aziz, H., & Ahmad, A. (2016). Secure software development life cycle (SSDLC): A systematic review. *International Journal of Advanced Computer Science and Applications, 7*(9), 405–414. https://doi.org/10.14569/IJACSA.2016.070954
11. Amazon Web Services. (n.d.). *AWS shared responsibility model.* https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/shared-responsibility.htm.
12. National Institute of Standards and Technology. (2020). *NIST special publication 800-207: Zero Trust architecture.* https://doi.org/10.6028/NIST.SP.800-207.
13. Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE, 63*(9), 1278–1308.
14. Das, B. S., & Chu, V. (2023). Security as code. Boston, MA: O'Reilly Media.
15. Kim, G., Behr, K., & Spafford, G. (2013). *The Phoenix project: A novel about IT, DevOps, and helping your business win*. IT Revolution Press.

16. Chauhan, M., & Shiaeles, S. (2023). An analysis of cloud security frameworks, problems and proposed solutions. *Network, 3*(3), 422–450. https://doi.org/10.3390/network3030018

17. Saleh, S. M., Madhavji, N., & Steinbacher, J. (2024). *A systematic literature review on continuous integration and deployment (CI/CD) for secure cloud computing*. University of Western Ontario; IBM Canada Lab.

18. Kiashemshaki, K., Torkamani, M. J., & Mahmoudi, N. (2025). *Secure coding for web applications: Frameworks, challenges, and the role of LLMs*. https://doi.org/10.48550/arXiv.2507.22223

19. Vakhula, O., & Opirskyy, I. (2023). *Research on security issues in cloud environments and solutions using the "security as code" approach*. Ukrainian Scientific Journal of Information Security, 25(3). https://doi.org/10.18372/2410-7840.25.17936

20. Matseniuk, Y., & Partyka, A. (2024). *The concept of automated compliance verification as the foundation of a fundamental cloud security model*. Computer Systems and Networks, 6(1), 108–123. https://doi.org/10.23939/csn2024.01.108

21. Pawar, A. S. (2025). *Cloud-native security: A review of modern approaches*. *International Journal of Scientific Research & Engineering Trends*. https://ijsret.com/wp-content/uploads/2025/03/IJSRET_V11_issue2_576.pdf

22. Feio, R., Martins, R., & Ferreira, P. (2024). *An empirical study of DevSecOps focused on continuous security testing*. IEEE European Symposium on Security and Privacy Workshops, 610-617. https://doi.org/10.1109/EuroSPW61312.2024.00074

23. Joseph, W. (2025). *DevSecOps in the cloud-native era: Automation, security, and continuous integration*. University of Toronto. https://www.researchgate.net/publication/391077724_DevSecOps_in_the_Cloud-Native_Era_Automation_Security_and_Continuous_Integration

24. Verdet, L., Clark, M., & Peters, J. (2023). *Exploring security practices in infrastructure as code: An empirical study*. Ecole Polytechnique, Montreal (Canada) ProQuest Dissertations & Theses, 2023, 31889262. https://doi.org/10.48550/arXiv.2308.03952

25. William, T. (2023). *Building resilient cloud-native systems: The role of DevSecOps in early detection and ongoing protection*. https://www.researchgate.net/publication/386076263_Building_Resilient_Cloud-Native_Systems_The_Role_of_DevSecOps_in_Early_Detection_and_Ongoing_Protection

26. Myrbakken, H., Colomo-Palacios, R. (2017). *DevSecOps practices and tools: A multivocal literature review*. Software Process Improvement and Capability Determination. SPICE 2017. Communications in Computer and Information Science, vol 770.. https://doi.org/10.1007/978-3-319-67383-7_2

27. Leshchenko, I., Horbachova, N., & Bielov, A. (2024). *Integrating DevSecOps into the software development lifecycle: A comprehensive model for securing containerized and cloud-native environments*. Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II. https://ceur-ws.org/Vol-3826

28. Gadani, N. (2024). Security challenges in cloud-based software development: A DevSecOps perspective. The Journal of Scientific and Engineering Research. https://www.researchgate.net/publication/383982252_Security_Challenges_in_Cloud-Based_Software_Development_A_DevSecOps_Perspective

29. Umeugo, W. (2023). Secure software development lifecycle: A case for adoption in software SMEs. International Journal of Advanced Research in Computer Science. https://doi.org/10.26483/ijarcs.v14i1.6949.

30. Pranav, M., Madhesh, I., Lenin, J., & Sasikumar, R. (2025a). Advances in DevSecOps and the future of cybersecurity using automation. Proceedings of the 4th International Conference on Information Technology, Civil Innovation, Science, and Management, ICITSM 2025, 28-29 April 2025, Tiruchengode. http://dx.doi.org/10.4108/eai.28-4-2025.2357955.

31. Pranav, M., Madhesh, I., Lenin, J., & Sasikumar, R. (2025b). An introduction to adaptive software security. https://doi.org/10.48550/arXiv.2312.17358.

Добавлено примечание ([5]): У цих статей є журнали/збірки конференцій, в яких їх було опубліковано. Згідно APA 6th вони мають бути зазначені.

Добавлено примечание ([6R5]): Деякі випущені як окремі наукові статті