BULLETIN of Cherkasy State Technological University

Journal homepage: https://bulletin-chstu.com.ua/en

Vol. 31 No. 1. 2026

UDC 004.4/.415+004.89/.9
DOI: 10.62660/bcstu/1.2026.85

ISSN: 2306-4412
E-ISSN: 2708-6070

# The impact of cloud services on secure software development

**Maksym Shishkin***
Postgraduate Student
Simon Kuznets Kharkiv National University of Economics
61165, 9-A Nauky Ave., 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 aimed to examine 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 that demonstrate the use of cloud tools and platforms were considered, including Infrastructure as Code and Continuous Integration/ Continuous Deployment, to increase productivity, agility, and early implementation of security controls (a "security shift to the left" approach). The analysis results has shown that the use of cloud practices in a secure SDLC 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 were identified. A set of best practices for implementing secure cloud SDLC workflows was 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 organisations create sustainable, efficient, and secure cloud development environments

**Keywords:** cybersecurity practices in development; infrastructure as code; continuous integration and delivery pipelines; automated vulnerability testing; configuration management; cloud responsibility models; system oversight and monitoring

## INTRODUCTION

As of 2025, software development takes place in an environment characterised by increasingly dynamic business requirements, the globalisation of teams, high infrastructure workloads, and continuous release delivery. In such an environment, traditional Software Development Life Cycle (SDLC) models, while still relevant for stable and predictable projects, often prove insufficiently flexible without the integration of cloud services. Lack of adaptability can lead to increased risks of malicious attacks, data leaks, compliance violations, and loss of control over the environment. At the same time, cloud services offer scalability, automation, and integration of security mechanisms at all stages of the SDLC, opening up new opportunities for creating resilient, secure, and flexible development processes. Therefore, in these conditions, it is necessary not to abandon traditional approaches, but to learn to make informed choices between classic and cloud-oriented models depending on the requirements, risks, and goals of a specific project.

S.M. Saleh *et al.* (2024) conducted a systematic review of 66 publications devoted to the security of Continuous Integration/Continuous Deployment (CI/CD) pipelines in a cloud environment. The study identified

**Article's History:** Received: 07.11.2025; Revised: 10.02.2026; Accepted: 16.03.2026; Published: 08.04.2026

**Suggested Citation:**
Shishkin, M. (2026). The impact of cloud services on secure software development. *Bulletin of Cherkasy State Technological University*, 31(1), 85-100. doi: 10.62660/bcstu/1.2026.85.

*Corresponding author

the main tools (e.g., Harbour, SonarQube, GitHub Actions) and key challenges (image manipulation, unauthorised access, weak authentication). The study concluded that the integration of security at the CI/CD stages significantly reduces the number of vulnerabilities in production environments. The study by M. Chauhan & S. Shiaeles (2023) examined cloud security frameworks (The National Institute of Standards and Technology, NIST Cloud; Cloud Security Alliance, etc.) and noted that none of them fully cover all security criteria, creating a "gap" in standardisation. The study proposed a unique generalised model for the assessment of the security of cloud services, combining elements of existing frameworks to reduce the gaps between standards. K. Kiashemshaki *et al.* (2025) addressed secure programming practices, including assessing the role of large language models (LLMs) in code generation, noting that despite the growth of automation, the implementation of secure code remains fragmented. The study emphasised that the use of LLMs in cloud environments can only be effective if automatic security validation systems are integrated. O. Vakhula & I. Opirskyy (2023) investigated the integration of the "Security as Code" approach in cloud environments for security automation at the Infrastructure as Code (IaC) and Development, Security, Operations (DevSecOps) levels. The study demonstrated that this approach reduces the human factor in the DevSecOps process and ensures continuous compliance of systems with standards requirements. The work of Y. Matseniuk & A. Partyka (2024) proposed an automated compliance verification method for Amazon Web Services/Google Cloud Platform (AWS/GCP) and Azure cloud accounts and emphasised the importance of configuration standardisation. The study developed a configuration control model that uses IaC policies to ensure continuous security auditing in DevOps processes. In a review paper, researcher A.S. Pawar (2025) described modern approaches to securing containerised and microservice environments, with an emphasis on security practices in cloud-native systems, such as service mesh, runtime protection, and DevSecOps. The study emphasised that the transition to cloud-native technologies requires a new way of thinking about security, where the focus shifts from perimeter protection to the security of components and development processes.

Despite significant progress in researching cloud environment security and DevSecOps tools, the direct impact of cloud services on the entire SDLC in the context of security has been analysed to a lesser extent. There has been insufficient research into how the use of cloud services changes approaches to requirements, design, testing, and deployment from a security perspective, especially in the Ukrainian or regional context. Therefore, the research relevance is determined. The study aimed to identify the main aspects of the impact of cloud services on secure software development and to outline key areas for improving security practices in this area.

## MATERIALS AND METHODS

This study is a systematic comparative review. The comparison method was used within the scope of thy study. The objects of comparison were the traditional and SSDLC, to determine their advantages, limitations, and impact on development security. Traditional SDLC was characterised as a linear and sequential approach in which each stage is completed before the next one begins. It is based on local development environments, manual testing, static infrastructure configuration, and limited scalability. This approach remains effective in stable and predictable environments but is less adaptable to dynamic cloud ecosystems. The study focused separately on the SSDLC model. The methodology was built as a step-by-step research cycle: defining comparison criteria; collecting relevant sources and systematising them; content analysis; comparative evaluation of SDLC and Cloud SSDLC models; and forming generalised conclusions.

Cloud SSDLC was defined as a model that integrates DevOps, continuous integration and delivery, containerisation, IaC, automated testing, and the Shift-Left Security approach. This ensures flexibility, automation, repeatability of environments, and built-in security controls at all stages of the lifecycle. The comparison methodology was conducted through a step-by-step analysis of the six phases of SDLC, for each of which the following criteria were evaluated: automation, flexibility, scalability, security, deployment speed, operational risks, and configuration management, as these parameters most fully reflect the key differences between traditional and cloud-oriented models, determine the quality and efficiency of each phase of the life cycle, and directly affect the ability of organisations to meet modern requirements for reliability, security, and speed of software delivery.

Content analysis was used to systematise the results, summarised comparative tables were created, and the analytical part was based on a comprehensive review of various types of sources, including peer-reviewed scientific publications for 2018-2025, technical reports from leading companies (Amazon, Microsoft, Google, Red Hat, CNCF), official documentation of cloud platforms, industry standards and recommendations (NIST, 2018; OWASP Foundation, n.d.). Industry analytical materials such as developer blogs and white papers (Amazon Web Services, n.d.b) were also used. Sources were collected from Google Scholar, IEEE Xplore, Scopus, ACM Digital Library, and Semantic Scholar databases. The following keywords were used for the search: "cloud SDLC", "secure SDLC", "DevSecOps", "cloud-native security", "infrastructure as code", "CI/CD security", "shift-left security". The analysis included works describing SDLC or SSDLC methodologies, related to cloud infrastructure, and containing empirical results or formal models. Sources that duplicate results, lack technical specifics, or refer to outdated technical approaches (before 2018, except for fundamental works) were excluded.

The quality of sources was assessed according to the following parameters: presence of peer review, date of publication, technical specifics, and consistency with modern cloud architecture models. Further synthesis of the material was conducted by classifying sources according to SDLC stages and types of technical solutions (DevOps processes, security, infrastructure, testing), which made it possible to comprehensively compare models and formulate conclusions about the effectiveness of Cloud SSDLC.

## RESULTS AND DISCUSSION

The software development life cycle describes the complete sequence of stages in the creation of a software product, from requirements analysis to operational support. The main stages include requirements analysis, design, development, testing, deployment, and support. This structure ensures consistency, predictability, and quality control at all stages of development, which can be used for standardisation of the process and comparison of different approaches to its implementation. In contrast to the traditional SDLC, which involves the implementation of security practices mainly during the testing or post-deployment stages, SSDLC integrates security at all stages of software development. This includes threat modelling during requirements analysis, architecture design with security checkpoints, automated code review, dynamic and static vulnerability scanning, secure deployment mechanisms, and real-time monitoring. This approach supports the concept of "shift-left security", where risks are eliminated early, reducing the cost of fixing bugs and increasing system reliability. Within the cloud approach, SSDLC is combined with DevOps practices to form a DevSecOps model based on automation, infrastructure as code, and continuous compliance monitoring.

Requirements gathering and analysis are the first and one of the most relevant stages of SDLC. At this stage, expectations and constraints for the future software system are defined from both a user and technical perspective. The process involves close interaction between all stakeholders, including business analysts, project managers, end users, and development teams (IEEE Computer Society, 2024). This usually results in the compilation of key documentation, such as business requirements, functional specifications, and descriptions of use cases and user stories. In traditional, non-cloud environments, this process is often linear and based on static documents and pre-scheduled meetings. This approach can cause delays and misunderstandings between project participants (Basili & Turner, 1975). The use of cloud technologies significantly changes the approach to requirements gathering. Thanks to real-time collaboration and interaction capabilities, cloud tools increase the efficiency, transparency, and flexibility of this process (Google Cloud, n.d.). Notable examples include Atlassian Jira and Confluence, which provide documentation and user story tracking with real-time editing and version control. Lucidchart and Miro are used for visualising processes and collaborating on ideas, while Google Workspace and Microsoft 365 are used for creating and editing text documents and spreadsheets. In the field of interface and experience design, Figma and FigJam are actively used to support interactive prototyping and collaborative workshops. In addition, platforms such as Amazon Honeycode and Google AppSheet provide low-code tools that can be used to quickly create prototypes of business processes based on collected requirements (Amazon Web Services, n.d.a).

Cloud-based tools help overcome a range of typical challenges associated with the traditional approach to requirements gathering. They improve collaboration between all stakeholders through real-time co-editing and commenting, which significantly reduces reliance on synchronous meetings. Automated change tracking and access to edit history simplify version control and ensure process transparency. In addition, cloud solutions help accelerate iterations by quickly updating requirements in line with changes in the business environment. Lastly, they render the process more inclusive: participants from different locations and time zones can actively participate, reducing the probability of misunderstandings and improving the alignment of the product vision (Jagli & Yeddu, 2017).

Despite the numerous advantages of cloud tools, some problems remain relevant. In particular, intensive collaboration can lead to an excessive amount of documentation and user stories, making it difficult to prioritise and manage them. There is also the issue of tool fragmentation: teams often combine multiple platforms (e.g., Jira, Miro, and Google Docs) that do not always integrate seamlessly, making centralised requirements management difficult. In addition, organisations operating under traditional or hybrid development models may find it difficult to adapt their processes to more flexible, cloud-oriented and Agile-compatible approaches. New challenges associated with cloud integration. Notably, the introduction of cloud platforms is accompanied by new challenges that were not characteristic of traditional operating models. One of the main issues is security and access control: confidential project and business information stored in a cloud environment requires careful user identity management, effective data encryption, and compliance with NIST (2018) regulatory requirements.

Another challenge is the "collaboration fatigue". The constant availability of cloud tools can cause notification overload, emotional exhaustion among employees, and, as a result, a decline in overall productivity. In addition, the effectiveness of working in a cloud environment depends largely on a stable internet connection. In cases where connectivity is unstable or limited, this can seriously interfere with key planning and coordination sessions. Cultural resistance should not be ignored either: some stakeholders are reluctant to abandon formal,

document-oriented practices in favour of more flexible, iterative, and collaborative approaches that involve the use of cloud tools. In addition to the advantages and challenges described above, it is worth summarising the main differences between traditional and cloud-based approaches to requirements gathering (Table 1).

**Table 1.** Key differences between traditional and cloud-based approaches to requirements gathering

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Process characteristic | Linear, sequential, with fixed stages | Iterative, flexible, with priority on continuous improvement |
| Tools | Word/Excel, offline meetings, e-mail | Jira, Confluence, Miro, Figma, Google Workspace, Microsoft 365 |
| Participant interaction | Mostly synchronous (meetings, calls) | Asynchronous and synchronous real-time interaction via cloud services |
| Version control | Manual document updates, difficult change tracking | Automatic saving of change history and version control |
| Accessibility | Limited – depends on physical meetings or corporate network | Access from any location via the Internet |
| Flexibility in requirement changes | Low – changes often require reworking of documentation | High: requirements are updated directly in cloud systems |
| Key challenges | Slow communication, lack of transparency | Data security, tool fragmentation, "collaboration fatigue" |

**Source:** compiled by the author based on V.R. Basili & A.J. Turner (1975), D. Jagli & S. Yeddu (2017), NIST (2018), IEEE Computer Society (2024), Google Cloud (n.d.), Amazon Web Services (n.d.a; n.d.b)

The comparison presented demonstrates that the traditional approach to requirements gathering is linear and inflexible, relying heavily on physical meetings and manual document management, which complicates changes, slows down communication, and increases the risk of version inconsistencies. The cloud-oriented approach, on the other hand, provides iterativeness, centralised access and automated version control, which promotes process transparency, rapid requirements updates and effective collaboration between distributed teams. At the same time, the integration of a large number of tools creates the risk of process fragmentation, and working with confidential data in the cloud increases security requirements, particularly for access management, encryption, and operational control. The dynamic nature of cloud services can also create new threats, such as misconfigurations, unwanted data distribution, or incorrect permissions. Therefore, cloud services significantly improve the adaptability and quality of the requirements gathering process, but the effectiveness of this approach depends on proper standardisation, maturity of access management practices, and adherence to cloud security principles. After the requirements gathering stage comes the system design phase, which involves transforming functional expectations into a structure suitable for technical implementation. This process encompasses both high-level architectures, including service components, data flows, and security boundaries, and detailed design, including database schemas, software interface descriptions, and user interface behaviour modelling (IEEE Computer Society, 2024). In traditional local environments, design typically involves creating static architectural diagrams and manually planning infrastructure within limited resources. This approach often limits the flexibility, scalability, and adaptability of the system to changing requirements (Basili & Turner, 1975).

The impact of cloud tools on system design. Modern cloud technologies significantly expand the possibilities of architectural design by supporting dynamism, scalability, and automation. Cloud platforms provide developers with the means to quickly create architectural diagrams, support collaboration on architectural solutions, programmatically describe infrastructure with the ability to version it, and formalise documentation of decisions made. In addition, they offer methodological support in the form of frameworks that help adhere to the principles of reliability, performance, security, and cost optimisation in the design process. These capabilities are implemented using tools and services such as AWS Architecture Diagrams, Azure Architecture Centre, Google Cloud Architecture Framework, Lucidchart, Draw.io, Miro platforms, infrastructure as code tools, Architecture Decision Records (ADR) documentation formats, and Well-Architected Frameworks offered by Amazon Web Services, Azure, and Google Cloud Platform (Amazon Web Services, n.d.a; Google Cloud, n.d.).

The use of cloud tools addresses a range of critical shortcomings of the traditional approach to design. They eliminate scalability limitations through native support for horizontal scaling, elasticity, and microservice architecture (Jagli & Yeddu, 2017). Automation through IaC avoids human error by reducing the need for manual infrastructure configuration. Cloud platforms also promote a modular approach to architecture, facilitating the gradual evolution of the system. In addition, the ability to create isolated environments on demand can be used to quickly test solutions before the main implementation begins, which significantly accelerates feedback.

At the same time, some challenges remain relevant even in a cloud environment. Overuse of cloud platform functionality can lead to so-called over-engineering, where architecture becomes overly complex without an objective need (Jagli & Yeddu, 2017). In addition, project documentation is often scattered across different tools, from visual modelling diagrams to IaC repositories and team collaboration platforms, making holistic management difficult. Another important issue is knowledge gaps: not all teams have the relevant expertise in cloud distributed systems, especially those accustomed to traditional approaches. New risks are caused by integration with cloud services. Integration with cloud services also creates new types of risks that require careful monitoring. One of them is the complexity of budget forecasting, as cloud pricing models (such as pay-per-use or multi-tier pricing) require more in-depth financial planning (Amazon Web Services, n.d.a). Another aspect is security, which must be considered at the design stage, from data encryption and user role management to Application Programming Interface (API) gateway configuration (OWASP Foundation, n.d.). In addition, working with IaC requires not only technical knowledge but also ongoing maintenance, as delayed updates can lead to technical debt accumulation (Jagli & Yeddu, 2017). Lastly, dependence on specific services from a single provider (e.g., AWS Lambda or Azure Functions) may limit the ability to migrate to another platform, creating the risk of vendor lock-in (Google Cloud, n.d.). In addition to the advantages and challenges described above, it is necessary to summarise the main differences between traditional and cloud-based approaches to system design (Table 2).

**Table 2.** Key differences between traditional and cloud-based approaches to system design

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| The nature of the design process | Static, sequential design with fixed architectural solutions | Dynamic, flexible design with the ability to quickly change the architecture |
| Architecture type | Monolithic, with limited scalability | Microservice or cloud-native, with elasticity support |
| Project design tools | Local graphic redactors (Visio, offline Drawio) | Cloud platforms (AWS Architecture Centre, Azure Diagrams, Lucidchart, Miro) |
| Infrastructure description | Manual planning and documentation | Infrastructure as code (Terraform, CloudFormation, Pulumi) |
| Process automation | Minimal configuration is manual | High – deployment, testing and monitoring are automated |
| Team collaboration | Limited, dependent on physical meetings or local network | Real-time collaboration, cloud storage of artefacts |
| Security | Added in the later stages of the project | Addressed at the architecture stage (security by design) |
| Scalability | Limited by hardware | Flexible – dynamic scaling of cloud resources |
| Main risks | Limited adaptability, slow architectural evolution | Over-engineering, vendor lock-in, and the need for a highly skilled team |

**Source:** compiled by the author based on V.R. Basili & A.J. Turner (1975), D. Jagli & S. Yeddu (2017), IEEE Computer Society (2024), Google Cloud (n.d.), OWASP Foundation (n.d.), Amazon Web Services (n.d.a)

The comparison showed that the cloud-oriented approach to design differs significantly in terms of flexibility, support for microservice architecture, and the use of infrastructure as code, which provides automation, scalability, and security considerations at the architectural level. Traditional design remains static, monolithic, and dependent on manual documentation, which slows down system development. Cloud tools such as AWS Architecture Centre or Terraform improve collaboration and accelerate decision-making, but they also create new risks, including the possibility of errors in IaC configurations, excessive permissions, leakage of architectural artefacts, or vendor lock-in. Therefore, the effectiveness of cloud design depends on controlled complexity and proper security management.

During the development stage, the architectural design is transformed into working software. Developers implement functionality, write code, integrate services, and test modules in accordance with predefined requirements and technical solutions. During this period, individual components are also tested, code reviews are conducted, and version control is performed to maintain the quality and stability of the software product (IEEE Computer Society, 2024). In traditional approaches, especially in local environments, the development process was often limited by the computing resources of a single machine, insufficient interaction between team members, and manual testing and deployment scenarios. These factors contributed to delays, environmental consistency issues, and overall productivity declines.

Cloud platforms provide developers with a wide range of tools that simplify, standardise and automate software development processes. They create unified development environments from any access point, integrate

continuous integration and delivery mechanisms, centrally manage dependencies, detect defects and vulnerabilities in a timely manner, and gradually implement new functionality with the ability to control the impact on end users. These capabilities are implemented using cloud-based integrated development environments (e.g., AWS Cloud9), CI/CD tools (GitHub Actions, GitLab CI/CD, AWS CodePipeline), artefact repositories (JFrog Artifactory, AWS CodeArtifact), code and security analysis tools (SonarCloud, Snyk, GitHub Dependabot), and feature control systems (LaunchDarkly, CloudBees) (OWASP Foundation, n.d.).

The integration of cloud technologies largely overcomes many traditional barriers to development. One of the most relevant advantages is the elimination of inconsistencies between environments. The use of cloud IDEs or containerised environments (e.g., Docker, Dev Containers) avoids situations where an application only works on the developer's local machine. Deployment time is also significantly reduced thanks to automated CI/CD pipelines, which improves release frequency and reduces the probability of human error (Jagli & Yeddu, 2017). Cloud repositories and collaboration platforms such as GitHub or Bitbucket improve interaction between team members by enabling collaborative editing, code review, and real-time change control. In addition, integrated security analysis tools can be used to identify potential vulnerabilities at the code writing stage, reducing risks for the finished product (OWASP Foundation, n.d.). Despite numerous improvements, some challenges remain relevant in cloud development. Frequent iterations and rapid deployment can push teams to make temporary decisions that accumulate technical debt and reduce code maintainability (IEEE Computer Society, 2024). Maintaining consistent code quality, despite the availability of modern tools, still depends largely on the internal discipline of the team and clear development standards. In addition, it is often difficult for new team members to join a project without proper documentation or integration standards, which slows down the adaptation process.

New risks are caused by integration with cloud services. Integrating the cloud into the development process also creates new risks. In particular, the diversity of tools and practices such as containers, serverless architecture, and IaC can overwhelm teams, especially less experienced ones. Excessive automation, while useful, sometimes leads to blind trust in systems that may miss critical defects or malfunction without proper verification. There is also an increased risk associated with incorrect configurations; for example, an error in a Terraform or YAML file for a CI/CD process can make an application vulnerable (NIST, 2018). Lastly, the use of vendor-specific services (such as AWS Lambda, Firebase, or Azure Functions) can lead to dependence on a single cloud platform, reducing flexibility and the ability to migrate solutions in the future. In addition to the advantages and challenges described above, it is necessary to summarise the main differences between traditional and cloud-based approaches to system development (Table 3).

**Table 3.** Key differences between traditional and cloud-based approaches to system development

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Development environment | Local, depends on the configuration of individual developers' machines | Cloud-based or containerised (GitHub Codespaces, AWS Cloud9, Docker) |
| Integration and delivery | Manual testing and deployment, long release cycles | Continuous integration and delivery (CI/CD) with automated testing and releases |
| Dependency management | Lack of centralised control, frequent conflicts between versions | Centralised artefact repositories (JFrog, AWS CodeArtifact) |
| Version control and collaboration | Local repositories, limited team collaboration | Cloud-based Git platforms (GitHub, GitLab, Bitbucket) with collaborative editing and code review capabilities |
| Quality control | Testing is performed after the main development, often manually. | Automated testing, static code analysis (SonarCloud, Snyk) |
| Security | Implemented after the implementation stage | Integrated into the development process (security-as-code, shift-left security) |
| Flexibility and scalability | Depends on the developer's hardware resources | Dynamic scaling, flexible environments on demand |
| Typical tools | Local IDE (IntelliJ, Eclipse, Visual Studio) | Cloud IDE (GitHub Codespaces, AWS Cloud9, JetBrains Space) |
| Main benefits | Simplicity of local process control | High deployment speed, team synchronisation, and automation |
| Main risks | Environment incompatibility, manual errors, and release delays | Tool overload, configuration errors, vendor lock-in |

**Source:** compiled by the author based on D. Jagli & S. Yeddu (2017), NIST (2018), IEEE Computer Society (2024), OWASP Foundation (n.d.)

The comparison shows that the transition from local, isolated, and manual processes to cloud-oriented development models not only accelerates delivery cycles but also significantly changes the security profile. In traditional environments, manual operations, lack of centralised version control, and dependence on local infrastructure increase the risks of human error, configuration drift, and environment inconsistency. The cloud approach, on the other hand, integrates security into the development process itself through shift-left practices, security-as-code, and automated static and dynamic code analysis, enabling vulnerabilities to be detected much earlier. The use of containerised environments, centralised dependency repositories, and CI/CD pipelines provides controllability, traceability, and a lower probability of configuration errors. However, increased automation and a large number of services also create new threats: there is a growing risk of configuration vulnerabilities in cloud resources, vendor lock-in, compromise of secrets in pipelines, and incorrect configuration of access rights. Thus, although the cloud approach significantly increases the level of proactive security, its effectiveness depends on mature infrastructure management, continuous monitoring, and adherence to the principles of least privilege and controlled automation.

The testing phase is a critical part of the software development life cycle, as it ensures that the implemented functionality complies with the specified requirements and quality standards. Testing covers various levels of verification, including unit, integration, system, load, and security testing (Google Cloud, n.d.). Its purpose is to identify errors, prevent regressions, and ensure that the software product meets user and business expectations before deployment in a production environment. In traditional SDLC models, testing was often postponed to the final stage of the development cycle and performed manually, which led to delays, accumulation of defects, and increased cost of fixes (Microsoft, n.d.). In addition, the isolation of test environments made it difficult to reproduce the real conditions in which the application would operate.

The impact of cloud tools on testing. Cloud technologies have radically transformed the testing process, making it more automated, scalable, and integrated into the overall development infrastructure. This has made it possible to continuously perform checks with every code change, ensure cross-browser and cross-platform compatibility without local environment configuration, simulate load in conditions close to production, detect vulnerabilities at the development stage, and test interaction with unavailable or unstable external services. These capabilities are implemented through the integration of tests into CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins in the cloud), cross-browser testing platforms (BrowserStack, Sauce Labs), load testing tools (Apache JMeter in AWS, Gatling Cloud, Azure Load Testing services), security analysis tools (Snyk, Checkmarx, OWASP ZAP) and API virtualisation services (WireMock Cloud, Mountebank) (Microsoft, n.d.).

The use of cloud solutions eliminates a range of problems that were characteristic of the traditional approach to testing. In particular, the scalability of cloud resources can be used for mass parallel testing without the limitations associated with local capacities. Cross-platform testing using device farms provides broader compatibility coverage, while cloud load testing can ensure accurate simulation of real-world usage scenarios. In addition, CI/CD pipelines provide instant feedback after code changes, which accelerates the detection and elimination of defects (Jagli & Yeddu, 2017). Unified cloud testing environments also mimic production conditions with significantly higher efficiency than on-premises configurations, reducing the risk of unexpected software behaviour after deployment. Despite significant progress, some problems remain relevant even in the cloud context. One of them is the instability of automated tests: their results can be inconsistent and false positives due to temporary dependencies, background processes, or insufficient data isolation (IEEE Computer Society, 2024). The cost of maintaining automated tests also increases over time. As the functionality of the application expands, tests must be constantly updated and adapted. Even in well-built automated pipelines, there is a risk of incomplete coverage. Due to time constraints or limitations of the tools, some critical scenarios may remain insufficiently tested.

New risks are caused by integration with cloud services. The transition to cloud testing, while creating new opportunities, also creates new risks. First, there is a growing need for careful handling of data to avoid leaks of confidential information and compliance with legal regulations (e.g., GDPR), NIST (2018). Furthermore, large-scale tests, especially those related to performance or end-to-end testing, can require significant computing resources, leading to increased costs. The security of test environments and CI/CD systems is critical, as these environments contain both code and secrets necessary for testing NIST, 2018). Lastly, the complexity of integrating various cloud tools into a single pipeline may require the development of special scripts, configurations, or even orchestration systems, which increases the technical complexity of the project. In addition to the advantages and challenges described above, it is worth summarising the main differences between traditional and cloud-based approaches to system testing (Table 4).

**Table 4.** Key differences between traditional and cloud-based approaches to system testing

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Testing environment | Local machines, limited resources | Cloud environments, dynamic scaling |
| Automation | Testing is predominantly manual | Integration of automated tests into CI/CD |

*Continued Table 4.*

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Load testing | Limited by local hardware | Scalable, realistic load testing through cloud resources |
| Cross-platform compatibility | Requires manual verification or emulators | Cloud device farms (BrowserStack, Sauce Labs) |
| Security testing | Implemented in later stages | Integrated into CI/CD pipelines (Snyk, Checkmarx) |
| Data management | Test data is stored locally | Centralised management in the cloud |
| Tool integration | Requires manual configuration | Supported by platforms (GitHub Actions, Jenkins Cloud) |

**Source:** compiled by the author based on NIST (2018), D. Jagli & S. Yeddu (2017), IEEE Computer Society (2024), Microsoft (n.d.), Google Cloud (n.d.)

Comparative analysis shows that traditional testing based on local resources and manual operations creates significant security risks: limited automation complicates the reproducibility of checks, delays the detection of vulnerabilities, and shifts security testing to the final stages of the SDLC, when fixes are costly and potentially dangerous for system stability. The cloud-oriented model eliminates these shortcomings by integrating automated security checks into CI/CD processes, enabling the scaling of environments to simulate real attacks, and using specialised tools such as Static Application Security Testing, Dynamic Application Security Testing, Software Composition Analysis, and secrets analysis, which significantly improve threat detection efficiency. In addition, centralised management of test data, logs, and artefacts strengthens access control and simplifies auditing, reducing the risks of unauthorised use or leakage of confidential information. At the same time, dependence on cloud platforms increases the requirements for proper configuration of access rights, segmentation of environments, and protection of CI/CD pipelines, as incorrect configurations or compromise of cloud accounts can create new attack vectors. Thus, cloud testing significantly increases the level of proactive security but requires greater discipline in configuration management and mature DevSecOps practices. Deployment is one of the final stages of the software development life cycle, during which the product is released into the production environment and becomes available to end users. In traditional SDLC models, the deployment process was often performed manually, sporadically, with pre-planned system downtime. This approach was prone to errors, had a high risk of failure, and was accompanied by significant support costs (NIST, 2018). With the advent of cloud tools and DevOps practices, deployment strategies have undergone significant changes. They have become automated, fast, and more continuous, avoiding downtime, reducing risks, and ensuring high application availability (Jagli & Yeddu, 2017).

The impact of cloud tools on the deployment stage. Modern cloud platforms create vast opportunities for automation and centralised management of software deployment processes in different environments. The deployment infrastructure is integrated into the development lifecycle, ensuring automatic collection, testing, and delivery of applications at each stage, which helps reduce errors, accelerate the release of new versions, and improve delivery reliability. Support for heterogeneous environments, from development to production, is achieved by configuring pipelines to suit the specifics of each environment. Containerisation unifies the delivery process, which can be used to scale computing resources, balance loads, and automatically restore services in the event of failures. Orchestration systems, particularly those based on Kubernetes, can be used for centralised management of the container lifecycle, automating deployment, updates and monitoring while maintaining stability and control over the infrastructure. These capabilities are implemented using continuous integration and delivery (CI/CD) services such as AWS CodeDeploy, Azure Pipelines, Google Cloud Deploy, and GitHub Actions, as well as container orchestration systems including the Kubernetes platforms Azure Kubernetes Service, Amazon Elastic Kubernetes Service, Google Kubernetes Engine, and Amazon ECS service.

Serverless deployment solutions occupy a separate niche, providing automatic scaling, event-driven function activation, and pay-as-you-go pricing. This can be used to prioritise business logic without wasting resources on maintaining a server environment. Another key component is Infrastructure as Code (IaC), which involves a software description of the entire infrastructure, including network components, computing resources, data storage services, and security policies. This approach provides reproducibility, version control, change auditing, and automation of deployment processes, which is critical for teamwork and DevOps practices. These capabilities are implemented using serverless deployment tools such as AWS Lambda, Azure Functions, and Google Cloud Functions, as well as IaC solutions including Terraform, AWS CloudFormation, and Pulumi. In addition, cloud deployment supports modern delivery strategies such as blue-green, canary, and rolling deployments. Specialised tools such as Argo CD and Spinnaker are used to implement these strategies, which can be used for gradual or parallel implementation of changes without negatively impacting users.

The use of cloud tools largely eliminates the problems inherent in the traditional approach to deployment. In particular, process automation reduces the probability of human error and ensures consistency across all environments. IaC eliminates the problem of "environment drift", where development, testing, and production environments differ from each other, often leading to unexpected application behaviour. CI/CD pipelines accelerate releases, ensuring their regularity and compliance with agile development methodologies. In addition, strategic deployment planning minimises or completely avoids downtime during updates. The use of gradual implementation methods, such as canary or blue-green, helps to reduce the risks associated with implementing changes in a production environment. Despite the significant advantages, some challenges remain relevant even in the context of cloud deployment. One such problem is the complexity of rolling back changes if critical errors are detected after release. If no appropriate action plan has been developed or no backups have been created, recovery can be difficult. In addition, a build failure or misconfiguration of the CI/CD pipeline can block a release and delay the product's time to market. Another substantial challenge is security. In particular, hard-coded credentials or improper secret management in deployment scripts can lead to confidential information leaks or create vulnerabilities in the system.

New risks are caused by integration with cloud services. Integrating deployments with cloud infrastructure also creates new risks that were not characteristic of traditional systems. One of them is related to cost monitoring. Frequent deployments can lead to the creation of temporary environments or excessive use of computing resources, requiring careful budget control. Process automation and open API endpoints also increase the attack surface. If CI/CD pipelines or environments are not protected by appropriate measures such as access control, event auditing, or privilege restrictions, this can create serious vulnerabilities. Another challenge is the proliferation of tool chains: for automated deployment to function properly, a large number of code management, testing, monitoring, and infrastructure services need to be integrated, which can lead to increased maintenance complexity. Lastly, the organisational aspect should not be underestimated either. The transition to continuous delivery requires not only technical changes, but also a change in culture within the team. In particular, responsibility of developers for code and release quality increases, as does the requirement for trust in automated processes and DevOps practices. In addition to the advantages and challenges described above, it is worth summarising the main differences between traditional and cloud-based approaches to system deployment (Table 5).

**Table 5.** Key differences between traditional and cloud approaches to system deployment

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Deployment process | Manual, periodic, with system downtime | Automated, continuous (CI/CD) |
| Infrastructure | Configured manually on physical or virtual servers | Described as code (IaC – Terraform, CloudFormation) |
| Execution environment | Fixed servers, scaling complexity | Containers and orchestration (Docker, Kubernetes) |
| Release strategies | One-time update with risk of issues | Blue-green, canary, rolling updates |
| Continuous solutions | None or require server control | AWS Lambda, Azure Functions, GCP Functions |
| Security | Manual control with accounts | Automated secret and security access policy control |
| Monitoring and roll-back | Conducted after a failure | Automated dashboards, alerts, and rollback processes |
| Organisational culture | Distribution of responsibilities between Dev and Ops | DevOps – joint responsibility for releases |

**Source:** compiled by the author based on D. Jagli & S. Yeddu (2017), NIST (2018), IEEE Computer Society (2024), Microsoft (n.d.), Google Cloud (n.d.)

Comparative analysis demonstrates that traditional deployment, performed manually and on statically configured servers, creates significant security risks: manual management of credentials, lack of a controlled environment for updates, and single-stage releases increase the likelihood of human error, leaks of secrets, and failures that are difficult to quickly localise and roll back. In such models, security checks occur after deployment, and monitoring is reactive, increasing the time to detect and respond to incidents. In contrast, cloud-native deployment integrates security directly into CI/CD pipelines through automated secret management, least privilege policies, IaC with configurable configurations, and integrity control mechanisms. The use of containerisation and orchestration ensures environment isolation, while strategies such as canary or blue-green deployments minimise risk by gradually introducing changes and quickly localising potential vulnerabilities. Automated dashboards, alerts, and rollback procedures reduce the "attack window" and ensure rapid response. At the same time, the increase in the number of integrated services raises the bar for proper configuration

management, CI/CD pipeline security, and IaC verification, as a configuration error or cloud account compromise can scale across the entire infrastructure.

Once software deployment is complete, the maintenance and monitoring phase begins, which continues throughout the entire system lifecycle. The main goal of this phase is to maintain uninterrupted software operation, promptly eliminate errors, optimise performance, provide updates, and ensure compliance with changing user needs. This includes continuous performance monitoring, log analysis, and resource usage control, which ensures compliance of the software with modern security requirements and technical standards (IEEE Computer Society, 2024). In traditional approaches, maintenance was usually performed manually: specialists analysed event logs, solved problems as they arose, and performed periodic updates. However, such reactive methods often proved ineffective: delays in responding to incidents led to prolonged downtime, and performance degradation could go unnoticed for a long time (Amazon Web Services, n.d.a).

Modern cloud environments create advanced opportunities for automating maintenance, monitoring system status, and ensuring high availability. With built-in monitoring and incident management mechanisms, engineers can track key metrics in real time, identify deviations from normal values, analyse the causes of failures, and automate responses to critical events. Centralised logging, telemetry data collection, alerts, automatic scaling and recovery, as well as patch and configuration management, ensure stable system operation even in complex dynamic environments. Specialised services are used to implement these capabilities, including Amazon CloudWatch, Azure Monitor, Google Cloud Operations Suite, Datadog, Splunk, etc.

Cloud solutions can address a range of limitations characteristic of traditional services. One of the main advantages is the elimination of limited visibility: centralised dashboards and logging systems provide a comprehensive picture of the system's status. Real-time monitoring can quickly detect potential failures before they affect users, and automatic scaling and self-healing mechanisms significantly reduce the need for constant manual intervention. In addition, automating routine tasks such as configuration updates or patching makes maintenance more stable and predictable. Despite these advantages, the use of cloud tools does not eliminate certain challenges for teams. One common problem is the high number of false alerts: poorly configured monitoring rules can generate an excessive number of non-critical messages, distracting engineers and leading to "alert fatigue". Analysing the root causes of failures also remains a difficult task, especially in distributed systems where incidents can have indirect and unpredictable consequences. In addition, constant monitoring and on-call duty can lead to burnout and reduced efficiency of support teams.

New risks are caused by integration with cloud services. The use of cloud services in the field of maintenance and monitoring creates new threats that must be addressed. One of them is tool overload: an excessive number of services and platforms complicates their integration and management, and also creates the risk of duplication of functions. The security of operational data is also becoming particularly relevant: logs, telemetry, and analytical reports may contain confidential information that requires adequate protection in accordance with access policies and data protection regulations (OWASP Foundation, n.d.). In addition, dependence on the stability and availability of cloud services creates additional vulnerability: in the event of provider failures, monitoring or response functionality may be compromised. The dynamic nature of the cloud environment is also worth mentioning: ephemeral instances, such as containers or serverless functions, complicate traditional tracing methods, requiring new approaches to debugging and monitoring. In addition to the advantages and challenges described above, it is worth summarising the main differences between traditional and cloud approaches to support and monitoring (Table 6).

**Table 6.** Key differences between traditional and cloud-based approaches to system support and monitoring

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| Support type | Reactive – troubleshooting | Proactive – automatic detection and incident prevention |
| System monitoring | Manual log analysis, periodic verifications | Continuous real-time monitoring through services (CloudWatch, Azure Monitor, Datadog) |
| Incident management | Performed manually, depending on the specialists on duty | Automated notification, root cause analysis, and response orchestration via PagerDuty, Splunk, etc. |
| Logging and telemetry | Decentralised log storage, difficulties with analytics | Centralised logging, telemetry collection and integrated analytics in the cloud |
| Scalability and accessibility | Limited by local infrastructure resources | Automatic scaling, self-healing mechanisms, and load balancing |
| Updates and patches | Applied manually, usually during downtimes | Automatic or scheduled updates via services such as AWS Systems Manager, Azure Automation |

*Continued Table 6.*

| Criteria | Traditional aspect | Cloud-oriented approach |
|---|---|---|
| System visibility | Partial, depending on access to individual components | Complete due to integrated dashboards and shared monitoring consoles |
| Operational data security | Minimum access policies, manual encryption | Access control, secrecy control, and data encryption at the service level |
| Key challenges | Delays in detecting errors, prolonged downtime, and manual diagnostics | Overload of notifications, complexity of tool integration, risks of provider service availability |

**Source:** compiled by the author based on IEEE Computer Society (2024), OWASP Foundation (n.d.), Amazon Web Services (n.d.a; n.d.b)

A comparative analysis shows that traditional support and monitoring models are characterised by low security due to a reactive approach: incidents are only detected after they occur, logging is performed locally and in a fragmented manner, and manual patching creates windows of vulnerability and the risk of configuration errors. Limited system visibility makes it difficult to detect threats in a timely manner, and the lack of centralised access control increases the probability of unauthorised changes. A cloud-oriented approach, on the other hand, embeds security into the support process: continuous real-time monitoring, centralised logging, and telemetry collection can be used for significantly faster detection of anomalies, suspicious activity, and potential attacks. Automated incident management, self-healing mechanisms, secrecy control, and least privilege policies reduce the human factor and increase the overall resilience of the system. At the same time, the cloud model creates unique security risks, including dependence on the availability of provider services, an increased attack surface due to numerous integrations, and the risk of operational data leaks if access is configured incorrectly. Therefore, although the cloud significantly enhances the security of system operation, its effectiveness depends on competent configuration, careful rights management, and the implementation of a coordinated operational data protection policy.

Security in cloud SDLC. Security is a critical aspect at all stages of the SDLC lifecycle, and its importance is further amplified in the context of cloud computing. This is due to the expansion of the attack surface, distributed architectures, the shared responsibility model, and the high dynamism of the infrastructure (Amazon Web Services, n.d.a). In this regard, the classic SDLC is evolving into a more complex SSDLC model, which involves the integration of security requirements, checks, and controls at every stage of development. One of the fundamental principles of SSDLC is the concept of Shift Left Security, which involves shifting the focus of security to the left on the SDLC timeline, i.e. to the earlier stages of design, planning and development. Instead of reactive vulnerability detection at later stages (testing or post-factum after deployment), this approach prioritises preventive detection and elimination of risks as early as the code writing phase. This is achieved through the implementation of static security analysis (SAST), dependency checks, configuration and access policy verification tools, and training developers in secure coding practices. For example, using services such as Snyk, Checkmarx, GitHub Dependabot, Trivy, or Bridgecrew integrates security checks into CI/CD pipelines, ensuring automatic detection of vulnerabilities before code enters the production environment (Davis, 2005).

Thus, security becomes not an isolated responsibility of individual teams, but an integral part of the entire engineering culture, from developers to DevOps engineers. Shift Left not only reduces the overall cost of defect elimination but also increases regulatory compliance, improves product quality, and minimises business risks. Security in the SDLC stages, requirements gathering and analysis. At the requirements gathering and analysis stage within the SSDLC, special attention is paid to defining security requirements, regulatory compliance, and modelling potential threats. In a cloud context, this process is complemented using specialised standards and tools. In particular, NIST (2018) and CIS Benchmarks recommendations, which define best practices incorporating the specifics of cloud infrastructure, are used as the basis for forming the basic level of security requirements. Cloud tools such as Microsoft Threat Modelling Tool can be used to model threats, which can be used for the systematic identification of possible attack vectors. This approach helps reduce risks associated with unclear security coverage or non-compliance with regulatory requirements. However, the cloud environment also presents new challenges, including the need to translate service level agreements and shared responsibility models between the customer and the cloud provider into specific software development requirements (Google Cloud, n.d.). Additionally, there is the challenge of achieving a common notion of cloud-specific threats among all teams involved.

At the system design stage, the key focus is on creating an architecture that incorporates security protocols, the principle of least privilege, and effective data protection mechanisms. In a cloud environment, this involves careful planning of identity and access management, which can be used to control and restrict the rights of users and services according to their needs (Microsoft, n.d.). In addition, it is worth implementing robust encryption strategies for both data storage and

transmission to ensure confidentiality and integrity. To ensure a secure network architecture, virtual private networks, firewalls, and other traffic isolation measures are used to reduce the risk of unauthorised access. This approach minimises the risks associated with excessive access rights or the use of weak encryption standards. At the same time, cloud systems pose new challenges, including security management in multi-regional and multi-cloud environments, as well as ensuring comprehensive protection for hybrid architectures that combine local and cloud resources.

During the code-writing stage, secure software is created by preventing vulnerabilities and effectively managing secrets. This is achieved using automated static code analysis, which can detect potential vulnerabilities early in the development process, as well as solutions for secure storage and management of sensitive data, which eliminate the need for hard-coding credentials. In addition, active use of tools for scanning vulnerabilities in dependencies ensures timely detection and elimination of risks associated with the use of third-party libraries. Examples of such tools include SAST systems, such as SonarCloud and Checkmarx, secret management services, such as AWS Secrets Manager and HashiCorp Vault, and dependency vulnerability monitoring solutions, such as Snyk and Dependabot. The introduction of these practices helps reduce the probability of errors in code implementation and eliminates hard-coded credentials, but at the same time increases the complexity of managing risks associated with the large-scale use of open-source software, which requires a systematic and comprehensive approach (OWASP Foundation, n.d.).

The process of testing security controls in cloud environments involves the use of dynamic application security testing capabilities, which can detect vulnerabilities during software execution. In addition, cloud vulnerability scanning methods are used to automatically analyse infrastructure settings and configurations to identify potential threats. The use of fuzz testing and sandbox environments can simulate attacks in a controlled, isolated environment, which increases the accuracy of security assessments. These approaches help reduce the risks associated with incorrect runtime environment settings, as well as open APIs or endpoints. At the same time, new challenges arise, such as the need to simulate realistic attacks in ephemeral infrastructures and ensure complete test coverage of variable cloud resources. Tools that implement these capabilities include Dynamic Application Security Testing systems, cloud vulnerability scanners such as AWS Inspector, and solutions for fuzz testing and sandbox environment management.

Secure deployment pipeline management involves integrating access control and security checks into the infrastructure configuration process using an IaC approach. Automated checks of IaC configurations can be used to quickly detect and eliminate errors, preventing configuration drift and unauthorised code deployment. In addition, the use of secret rotation practices in CI/CD pipelines increases the security of confidential data storage and use. The implementation of the immutable infrastructure concept helps maintain the stability and repeatability of deployment environments. At the same time, the process of strengthening the security of CI/CD pipelines poses new challenges, particularly those related to potential errors in the configuration of IaC templates, which can lead to vulnerabilities. To realise these opportunities, specialised IaC security checking tools such as tfsec and Checkov are used, as well as automated secret rotation mechanisms in CI/CD.

Continuous vulnerability monitoring, intrusion detection, and regulatory compliance auditing provide a comprehensive approach to maintaining cloud infrastructure security. Automated solutions for information and security event management (SIEM) provide real-time data collection, analysis, and correlation, facilitating rapid threat detection and minimising the time attackers spend in the system. In addition, automation of patch management processes helps to promptly eliminate identified vulnerabilities, reducing the risks of system exploitation. Continuous compliance monitoring, implemented through configuration and security policy control tools, ensures compliance with regulatory requirements and internal standards. However, the active use of such systems creates new challenges, including the risk of information overload due to excessive alerts and telemetry, as well as the need to quickly adapt to changes in cloud services and new vulnerabilities. To support these capabilities, cloud SIEM systems such as AWS GuardDuty and Azure Sentinel, remediation automation tools, and continuous compliance monitoring services such as AWS Config and Prisma Cloud are used.

In practical application, cloud security principles require a comprehensive approach that incorporates the characteristics of cloud platforms, the dynamic nature of infrastructure, and the scalability of systems. Effective cloud security requires the integration of multi-layered measures, ranging from access control and authentication to monitoring and automated incident management. A substantial aspect is the adaptation of traditional security methods to the specifics of cloud services, where resources are dynamically created, changed, and destroyed. Accordingly, the implementation of security practices must be flexible and automated to minimise the human factor and ensure a rapid response to threats. Key components include the application of the principles of modularity, standardisation and infrastructure as code, which can be used for the creation of repeatable, controllable and secure deployment environments.

The Shared Responsibility Model (Amazon Web Services, n.d.b) is a fundamental approach to the distribution of responsibilities between the cloud provider

and the user in the field of security. The cloud provider is responsible for protecting the infrastructure, which includes physical servers, network equipment, data centres, and the basic level of security for the services it provides. For their part, the user or development team is responsible for the security of applications, data, access configurations, and settings within the leased resources. A set division of responsibilities is critical to the prevention of security gaps, as misinterpreted boundaries of responsibility can lead to vulnerabilities and system compromises. Therefore, teams must actively implement security policies, configure access controls, and take responsibility for protecting their components, given the dynamic nature of cloud platforms.

Zero Trust Architecture (NIST, 2018) involves a radical rethinking of traditional approaches to security, which were based on trust in the internal network. In Zero Trust, every request, regardless of its source, is considered potentially dangerous and undergoes strict authentication and authorisation. This architecture is based on the principle of "least privilege" (Saltzer & Schroeder, 1975), which limits users and services to only the rights necessary to perform their tasks. The implementation of Zero Trust in cloud environments includes multi-factor authentication, continuous monitoring of user and device behaviour, and network segmentation to minimise potential points of access for attackers. These measures increase the resilience of systems to internal and external threats, reduce the risk of unauthorised access and the spread of attacks within the infrastructure.

The Security as Code approach (Das & Chu, 2023) involves applying software development principles to security management processes, including configurations, policies, and procedures. This means that all security settings and policies are formulated in the form of configuration files that are stored in version control systems, tested by automated means, and deployed using automation tools. This approach ensures transparency, reproducibility, and change control, and can be used for quick detection of errors and incompatibilities in security settings before they are applied in production environments. Security as code is a key element of modern DevSecOps (Kim *et al.*, 2013) practices, which integrate security directly into the software development and delivery process, increasing the overall security level of systems.

Security in the cloud SDLC is not a separate step, but a continuous integrated practice. While cloud tools and automation improve visibility, responsiveness, and compliance, they also create new threats and require a mastery of specific platform configurations. A proactive, built-in approach to security is essential to ensuring the resilience, confidentiality, and trustworthiness of cloud applications (Davis, 2005). The study found that the use of cloud services significantly changes security approaches at different stages of the software development lifecycle, in particular, increasing automation, flexibility, and early security integration, but at the same time, new risks emerge (e.g., vendor lock-in, configuration vulnerabilities, integration challenges).

R. Feio *et al.* (2024) studied the DevSecOps framework from the perspective of continuous security testing in CI/CD pipelines. The study determined that the application of this approach can be used for the early detection of vulnerabilities in real projects. According to their research, organisations that have implemented continuous testing have significantly reduced the number of critical errors. This confirms one of the theses that cloud practices integrated at the development and testing stage increase security. However, the study noted that the integration of tools must be thorough and that insufficient integration and fragmentation of tools can negatively affect security, which confirms the results of this study. W. Joseph (2025) investigated the impact of security automation and CI in cloud-native architectures. The author noted that the transition to microservices, containerisation, and orchestration significantly increases the complexity of security, but proper implementation of DevSecOps can be used to control this complexity. The conclusions state that cloud services offer advantages but are accompanied by new risks, such as incorrect configuration of containers or orchestrators. Thus, the author's results confirm the observations made in this work.

A. Verdet *et al.* (2023) analysed security practices in IaC configuration scripts. The study determined that although access policies are most commonly implemented, encryption "at rest" is substantially less popular. The conclusions emphasise that cloud tools can automate infrastructure as code, but there is a problem with misconfiguration, which directly correlates with the conclusion of A. Verdet *et al.* In other words, their research serves as empirical support for the statement made in this study regarding the risks of IaC implementation. T. William (2023) examined how DevSecOps contributes to the creation of resilient cloud-native systems through early vulnerability detection and continuous protection. The study highlighted that shift-left and secure-right security integration is key. The study similarly notes that security must be built in at the design stage and integrated continuously. Thus, the author's conclusions complement the presented results.

H. Myrbakken & L. Colomo-Palacios (2025) conducted a large-scale review of DevSecOps practices and tools, identifying 39 key practices and 114 tools associated with them. They emphasised that although the tools are available, organisations often fail to achieve maturity in security practices. The results show a similar problem: organisational culture and team training were found to be another barrier to the effective application of cloud-secure SDLC procedures. Thus, the conclusions are consistent with the conclusions of this work. I. Leshchenko *et al.* (2024) proposed an extended DevSecOps

model that includes the stages of secure decommissioning, security management, frequent auditing, and innovative security. The study emphasised a gap in typical models between the development and operation phases. The conclusions also note that cloud models require deeper integration between development, testing, deployment, and support. In other words, the authors' model reinforces the thesis of security process integrity.

N. Gadani (2024) covered the analysis of security challenges in cloud development (misconfigurations, insufficient API control, weak identification). These challenges echo the findings that configuration errors, vendor dependency, and tool fragmentation are significant problems. Thus, the presented study confirms the findings. W. Umeugo (2023) investigated the implementation of SSDLC for small/medium-sized enterprises and highlighted that organisational maturity and culture are central. The conclusions note that insufficient team qualifications or resistance to change (especially in traditional approaches) are barriers, which is consistent with the results of this study. M. Pranav *et al*. (2025a) analysed automation as the "driving force" of the next generation of DevSecOps. They accelerated the implementation of security but highlighted that excessive automation can lead to "blind" trust in systems. This study also identified a similar problem of "alert fatigue", excessive notifications, and the risk of automation without critical control. This confirms the findings of the presented study. M. Pranav *et al*. (2023b) proposed an adaptive security model based on the MAPE-K cycle that integrates security into the SDLC, constantly adapting to changes. This model complements the findings that cloud environments require not only security integration but also adaptability. Therefore, this point of view is considered valid and provides direction for further research.

It is possible to conclude that modern publications confirm the conclusions regarding the positive impact of cloud services on secure software development; they recognise the importance of DevSecOps, automation, security integration, the use of IaC and cloud architectures. At the same time, some works expand on these theses: they emphasise the decommissioning phase, adaptability, and the tool base. This study summarises these aspects and adds a focus on the specific risks of cloud models (vendor lock-in, tool fragmentation, 'alert fatigue') in the context of SSDLC. However, it is worth noting that some works focus more on general DevSecOps models than on the specifics of cloud services in the context of SDLC. At the same time, this study has a narrower focus on the cloud environment and its impact on the SDLC stages from a security perspective, which gives it a certain uniqueness. It is also noticeable that the number of empirical studies in this area (especially with measurements in production environments) is still limited, with most being review or conceptual in nature. This confirms the current conclusion that further research and practical case studies are needed. Based on a comparison with the literature, it is possible to conclude that the implementation of a cloud-oriented SSDLC with an emphasis on security has significant potential.

## CONCLUSIONS

Cloud platforms provide a high level of scalability and elasticity, which can be used for dynamic resource allocation based on software requirements. This helps create stable, high-performance environments for development and testing, eliminating infrastructure bottlenecks and ensuring smooth operation even during peak loads. Process automation and CI/CD support in cloud environments provide automatic application build, test, and deployment, shortening the delivery cycle and reducing errors. Together with built-in security mechanisms such as identity management, secret storage, compliance control, and runtime protection, this ensures security to be integrated early in the development process, reducing the risk of critical vulnerabilities. In addition, cloud platforms improve collaboration through centralised version control and real-time task tracking, facilitating interaction between remote teams. Real-time monitoring and logging tools enhance these benefits by enabling early detection of incidents and proactive response to performance issues.

Economic efficiency is also a compelling argument in favour of cloud SDLC. Pay-as-you-go models reduce start-up costs and provide flexibility in allocating budgets according to actual project needs. However, cloud-oriented SDLC comes with a range of challenges. One of them is vendor lock-in, where dependence on individual services complicates further migration. The complexity of configurations, unclear determination of the shared responsibility model, and the dynamic nature of cloud environments create additional risks. Without regular resource monitoring, costs can rise unpredictably. In addition, the need to integrate and maintain a large number of tools requires increased skills and adaptation of teams. The transition to cloud-based development methods often requires changes in culture, roles, and processes. Cloud SSDLC is modular: organisations can gradually implement individual services without changing the entire lifecycle at once. For example, cloud IDEs can be used for development while maintaining local pipelines; cloud testing can be applied without transferring repositories; IaC can be implemented only for individual subsystems. This approach minimises risks and can be used for the adjustment of the pace of modernisation in line with business needs.

In summary, the study showed that the transition to cloud SSDLC is not just a technological migration, but a strategic transformation of development processes. Cloud tools provide teams with flexibility, resilience, and enhanced security, but at the same time require the evolution of practices, skills, and approaches. Success depends on how effectively organisations

can combine innovation with control, modularity with integration, and speed with security. The cloud remains a powerful tool, but its effectiveness is determined by how it is used. Further studies should quantify the impact (e.g., reduction in vulnerabilities, release time, error costs) in different cloud scenarios and address the culture and training of teams in the context of cloud-secure development.

## ACKNOWLEDGEMENTS

None.

## FUNDING

None.

## CONFLICT OF INTEREST

None.

## REFERENCES

[1] Amazon Web Services. (n.d.a). *AWS well-architected framework – security pillar*. Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/security-pillar.
[2] Amazon Web Services. (n.d.b). *AWS shared responsibility model*. Retrieved from https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/shared-responsibility.htm.
[3] 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. doi: 10.1109/TSE.1975.6312870.
[4] Chauhan, M., & Shiaeles, S. (2023). An analysis of cloud security frameworks, problems and proposed solutions. *Network*, 3(3), 422-450. doi: 10.3390/network3030018.
[5] Das, B.S., & Chu, V. (2023). *Security as code*. Boston: O'Reilly Media Inc.
[6] Davis, N. (2005). *Secure software development life cycle processes: A technology scouting report*. Retrieved from https://api.semanticscholar.org/CorpusID:110809329.
[7] Feio, R., Santos, N., Escravana, N., & Pacheco, B. (2024). An empirical study of DevSecOps focused on continuous security testing. In *2024 IEEE European symposium on security and privacy workshops (EuroS&PW)* (pp. 610-617). doi: 10.1109/EuroSPW61312.2024.00074.
[8] Gadani, N.N. (2024). Security challenges in cloud-based software development: A DevSecOps perspective. *The Journal of Scientific and Engineering Research*, 11(5), 287-294.
[9] Google Cloud. (n.d.). Retrieved from https://cloud.google.com/security/solutions/software-supply-chain-security?hl=uk.
[10] IEEE Computer Society. (2024). *Guide to the software engineering body of knowledge (SWEBOK V3.0)*. Retrieved from https://www.computer.org/education/bodies-of-knowledge/software-engineering.
[11] Jagli, D., & Yeddu, S. (2017). CloudSDLC: Cloud software development life cycle. *International Journal of Computer Applications*, 168(8), 6-10. doi: 10.5120/ijca2017914468.
[12] Joseph, W. (2025). *DevSecOps in the cloud-native era: Automation, security, and continuous integration*. Retrieved from https://www.researchgate.net/publication/391077724_DevSecOps_in_the_Cloud-Native_Era_Automation_Security_and_Continuous_Integration.
[13] Kiashemshaki, K., Torkamani, M.J., & Mahmoudi, N. (2025). Secure coding for web applications: Frameworks, challenges, and the role of LLMs. *ArXiv*. doi: 10.48550/arXiv.2507.22223.
[14] Kim, G., Behr, K., & Spafford, G. (2013). *The Phoenix project: A novel about IT, DevOps, and helping your business win*. Portland: IT Revolution Press.
[15] Leshchenko, I., Horbachova, N., & Bielov, A. (2024). Integrating DevSecOps into the software development lifecycle: A comprehensive model for securing containerized and cloud-native environments. In *Proceedings of the cybersecurity providing in information and telecommunication systems II* (pp. 153-161). Aachen: CEUR.
[16] 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. doi: 10.23939/csn2024.01.108.
[17] Microsoft. (n.d.). *Zero Trust security model*. https://www.microsoft.com/en-us/security/business/zero-trust.
[18] Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps practices and tools: A multivocal literature review. In *Software process improvement and capability determination. SPICE 2017* (pp. 17-29). Cham: Springer. doi: 10.1007/978-3-319-67383-7_2.
[19] NIST. (2018). *Framework for improving critical infrastructure cybersecurity*. Retrieved from https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.
[20] OWASP Foundation. (n.d.). *OWASP secure SDLC cheat sheet*. Retrieved from https://owasp.org/www-project-cheat-sheets/.
[21] Pawar, A.S. (2025). Cloud-native security: A review of modern approaches. *International Journal of Scientific Research & Engineering Trends*, 11(2), 1703-1706.
[22] Pranav, M., Madhesh, I., Lenin, J., & Sasikumar, R. (2025a). Advances in DevSecOps and the future of cybersecurity using automation. In *Proceedings of the 4th international conference on information technology, civil innovation, science, and management* (pp. 1-18). Tiruchengode: EAI. doi: 10.4108/eai.28-4-2025.2357955.

[23] Pranav, M., Madhesh, I., Lenin, J., & Sasikumar, R. (2025b). An introduction to adaptive software security. *ArXiv*. doi: 10.48550/arXiv.2312.17358.
[24] Saleh, S.M., Madhavji, N., & Steinbacher, J. (2024). A systematic literature review on continuous integration and deployment (CI/CD) for secure cloud computing. In *Proceedings of the 20th international conference on web information systems and technologies WEBIST* (pp. 331-342). Porto: SciTePress. doi: 10.5220/0013018500003825.
[25] Saltzer, J.H., & Schroeder, M.D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278-1308. doi: 10.1109/PROC.1975.9939.
[26] Umeugo, W.C. (2023). Secure software development lifecycle: A case for adoption in software SMEs. *International Journal of Advanced Research in Computer Science*, 14(1), 5-12. doi: 10.26483/ijarcs.v14i1.6949.
[27] 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), 113-122. doi: 10.18372/2410-7840.25.17936.

## Вплив хмарних сервісів на безпечну розробку програмного забезпечення

**Максим Шишкін**
Аспірант
Харківський національний економічний університет імені Семена Кузнеця
61165, просп. Науки, 9-А, м. 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 було проведення аналізу інтеграції хмарних технологій у всі етапи життєвого циклу розробки програмного забезпечення (SDLC) з акцентом на систематичне впровадження та застосування практик безпеки протягом усього процесу розробки. Для досягнення цілей дослідження використано порівняльний та контент-аналіз, що охопив 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 та Безперервної інтеграції/Безперервного розгортання, для підвищення продуктивності, гнучкості та раннього впровадження механізмів контролю безпеки (підхід «зміщення безпеки вліво»). Результати аналізу показали, що застосування хмарних практик у безпечному SDLC 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