DomainForge – Automated Subdomain Generator and Deployment Assistant

Tanvi Agarwal
Department of AIT-CSE
Chandigarh University
Mohali-140413, Punjab, India
22bcc70170@cuchd.in

Vedant Pawar
Department of AIT-CSE
Chandigarh University
Mohali-140413, Punjab, India
22bcc70168@cuchd.in

Priyanshu
Department of AIT-CSE
Chandigarh University
Mohali-140413, Punjab, India
22iis70022@cuchd.in

Upilli Dileep
Department of AIT-CSE
Chandigarh University
Mohali-140413, Punjab, India
22bcc70009@cuchd.in

Ankit Garg
UCRD, AIT-CSE
Chandigarh University
Mohali-140413, Punjab, India
ankitgitm@gmail.com

***Abstract*—New software constructing has to work with both reliability and speed but small development squads frequently lack the capabilities to install highly sophisticated enterprise-class DevOps pipelines. In this paper, we introduce DomainForge, an ultra- lightweight automated testing framework that consolidates subdomain generation, DNS setup, and application rollout into a single platform. Unlike other infrastructure-bound solutions, DomainForge has a web-based frontend that self-links important deployment workflows like DNS record configuration, GitHub Repository integration, and HTTPS provisioning. Using do- primary registrar APIs, containerization, and automated SSL/TLS setup, the design successfully remits deployment issues in limited-resource settings. Experimental measures depicted avoid a 75% reduction in configuration errors and a considerable reduction in deployment time relative to the manual methods With its modular architecture and capability to support both containerized and non-containerized deployments, DomainForge delivers a scalable and high-performance solution for small-scale development practices.**

***Index Terms*—Deployment automation, DNS management, subdomain generation, DevOps, containerization, CI/CD, web application deployment**

I. Introduction

Over the past few years, the practice of software development has changed significantly, especially with the introduction of agile methodologies and continuous integration. These processes have changed not only how software is developed, but also how quickly it must be delivered, therefore emphasis is much more on the reliable deployment mechanisms. While these individual steps may be minor in isolation, they become much more tiresome and error-prone when repeated across projects or environments. Thus, over time, these small bits of complexity grow the possibility of misconfigurations at deployment. Current software deployment practices tend to polarize into two extreme flavors. On one end of the spectrum, enterprise-class platforms offer much in terms of functionality but require equally strong infrastructures and specialized expertise to operate. On the other end, light-weight scripting solutions require minimal installation effort but are usually missing integration, scalability, and automation features to ensure continuous and predictable operations. This chasm calls for a hybrid solution: advanced automation with the burden of an enterprise-class system.

Recent developments in software architecture, like containerization, microservices, and modern DNS infrastructure, are once again multiplying the complexity for operations. Activities such as DNS automation, subdomain management, and workflow orchestration have become core capabilities, moving from optional features to must-haves. Predictive routing and AI-driven threat analysis are starting to make their way into enterprise solutions, but these usually stay well beyond the grasp of smaller development teams.

This paper addresses the above challenges by describing DomainForge, a lean automation environment for small to mid-sized teams. It balances DomainForge between functionality and minimalism by knitting together subdomain configuration, DNS management, and application deployment into a single workflow. Instead of using heavyweight enterprise tools that bring unnecessary complexity to the process, it offers just enough automation to drive fast and reliable release processes in resource-constrained environments.

II. Literature Review

A. DNS Management and Automation Systems

As of today, the Domain Name System remains the backbone of full-stack web infrastructure, especially when applications scale out into distributed environments [1]. DNS over protocols like HTTPS (DoH) and DNS over TLS (DoT) do bring in security. protective measures, and in confidence, to inquiry transportation [2], but are more complex in automated networks. In more modern DNS systems are ever using API-based designs. where programmatic setting of zones and self-managing records [3]. Wholesale vendors like Cloudflare, AWS Route 53, and Google Cloud DNS offer REST-like APIs to integrate DNS updates within deployment pipelines [4],

but these are a requiring a extensively configuration skills to perform successfully, and too difficult to be implemented by small teams. Other studies also identified latency sensitivity leading to architecture and edge computing, for which DNS resolution bound to change dynamically in an attempt to optimize performance.[5]. Managed DNS such as NS1 show the advantage of self-automating traffic steering with policy programmable reducing operational overhead and reaction time [6]. But most of the remedies are still designed to a scale that is enterprise level and infrastructure centric, having ill-streamlined orchestration appropriate for small de- Development squads. This inconsistency demands the employment of light such as DomainForge that attempt to offer DNS automation without enterprise overhead.

B. Subdomain Enumeration

Subdomain enumeration has long been studied in the security, reconnaissance, and vulnerability context evaluation, where finding covert or incorrectly configured subdomain is crucial for discovering attack surfaces [7], [8]. The tools behind the brute-forcing and wordlist methods are rather documented, but they are focused on security and aren't too useful for real deployment. With the developments of web architectures, studies have commenced to explore generative methods that accommodate intentional sub- domain construction, transferring the emphasis from detection to managed provisioning [9]. This difference is especially true for automated websites, subdomains will have to be systemati- cally designated, administrated, and removed without running counter to existing naming conventions. Recent research have studied machine learning aided subdomain generation, that create models which explore organizational patterns to indicate context aware and non-colliding names [10]. In containerized environments, the services are short-lived, and subdomains based on time need to have collision-safe naming that automatically renames[11]. Despite such advancements, majority of the current research is limited to security scan- ing or research prototyes, without integration into operational deployment systems. This gap provides a need for system such as DomainForge that is integrating subdomain list not for exploration but as a functional component of automated deployment pipelines.

C. Container Deployment and Orchestration

Containerization has transformed deployment patterns in a simple approach by providing reproducible runtimes for development, testing, and production environments [12]. Some of the platforms such as Kubernetes and Docker provide isolated units of execution, allowing scaling and portability, but with operational issues related to orchestration, networking, and managing resources .[13]. While big systems gain from orchestration platforms to automate the lifecycle of containers, small teams lack the infrastructure and specialization to economically deploy such platforms. Additionally, container security capabilities such as image authentication, runtime isolation, and network isolation are still challenging to accomplish without full tooling aid [14]. These limitations reduce the container orchestration that can be realistically be reused by lightweight construction environments. New container structures enable immutably deployed patterns, where instances of containers are replaced rather than being patched when the system runs software updates for consistency and trace- ability [15]. It goes without saying that this model fits automated deployment strategies, it all comes down to efficient orchestration pipelines that are ready to harmonize DNS updates, certificate management, and scaling during runtime. Present business softwares can handle all these complexities but this naturally makes their configuration overhead higher compared to the larger labor forces. This means that there is a need of effective orchestration management that aggregates container deployment with supporting activities like DNS allocation and certificate managing-functions that activities such as DomainForge aim to fold into a single automated platform.

D. CI/CD Pipeline Security

CI/CD pipelines are now at the forefront of rapid software delivery but pose serious security risks because they disclose themself directly to production, credentials, and source code infrastructure [16]. The common vulnerabilities are poor access control, insecure handling of credentials, and insufficient verification of build artifacts [17], [18]. Pipeline poisoning and dependency injection are malicious attacks that target these vulnerabilities to provide unauthorized access or inject malicious code. Industry standard security best practices like OWASP also focus on similar best practices like secret management, build isolation, and integrity verification to safeguard against these types of attacks [19]. However , implementing these defenses do often demand specialized tooling and subject matter expertise that are not feasible for small development teams with limited resources. More recent work has also examined employing automated security scanning, runtime monitoring, and anomaly detection in CI/CD pipelines to proactively identify malicious activity [20], [21]. Immutable infrastructure tools where deployments over-write instead of modifying sections have been found to be helpful in providing auditability and minimizing configuration drift [22]. Despite these developments , most current solutions still target large-scale enterprises and cloud-native worlds with limited availability to smaller teams without specialized DevOps resources. This gap facilitates the needs of firms such as DomainForge, which aim to include safe CI/CD capability in a light and efficient manner while balancing security demands with simplicity of usage.

E. Automated Certificate Management

The adoption of SSL/TLS certificates has become indispensable for securing communications in modern web deployments, particularly when systems dynamically create and retire subdomains [23]. Protocols such as ACME, supported by services like Let's Encrypt, enable automated certificate issuance and renewal by validating domain ownership, eliminating the need for manual intervention [24]. Wildcard

certificates provide a scalable solution through securing all subdomains with a single certificate. They have, however, a key trade-off—if a wildcard certificate is breached, all subordinate subdomains are open to security vulnerabilities [25]. Issuing separate certificates gives better privacy and security but is more administrative effort and subject to rate limits imposed by certification authorities [26]. To mitigate operational risks, automated frameworks in- increrasingly incorporate certificate checking, renewal schedule- testing, and validation testing to avoid surprise expiration or configuration. Sophisticated systems also carry out post-renewal verification by synthetic HTTPS tests to to ensure proper chain installation and service provision [27]. Despite these capabilities, most existing certificate automation solutions are preserved hidden inside enterprise ecosystems with considerable configuration intensity. Minuscule squads generally lacks the infrastructure to incorporate certificate automation with DNS updates and orchestration of deployment. This supports the need for lightweight frameworks such as DomainForge, which are designed to integrate certificate management and domain pro- visioning and deploying it in single workflow

III. Proposed Methodology

An organized set of processes was followed all along the DomainForge incident to ensure that the finished product was productive, responsible, and appropriate real-world arrangement positions.

A. Requirement Analysis

DomainForge's founders were based on a structured examination of issues that conventional small development groups, individual developers, and early stage companies face with deploying application flows. By observing the existent deployment practices, it was clear that subdomain setup, DNS record management, repository copy, and application hosting were typically done manually. While these operations are required,result in operational inefficiencies, delayed releases, and constant configuration mistakes. The lack of lightweight automation tools for small-scale environments identified the primary shortfall that current enterprise platforms provide advanced features but are heavy in resources and need high levels of DevOps knowledge, while lightweight scripts lack integration and reliability that is required for production deployment. As a solution to these practical shortcomings, both non-functional and functional requirements were defined. Functionally, the system must automate domain provisioning, updating DNS, integrating repositories, executing deployments, and managing SSL certificates. Non-functional specifications prioritized usability, modularity, and cost-effectiveness, whereby the platform can run on light infrastructures such as virtual private servers (VPS) without need for elaborate orchestration layers. Informal feedbacks was also collected from the target audience—students, freelancers, and small teams—whose testimony confirmed the need for a browser-based interface that is capable of simplifying laborious deployment steps into an elegant workflow.

B. System Architecture Design

DomainForge architecture was designed module-based for maintenance, scalability, and seamless integration. The system basically has four primary components : Web Interface, Backend Service, Deployment Engine, and External API Integrations. All modules were programmed to execute independently while exporting cleanly defined interfaces, thus allowing runtime flexibility in several hosting environments.

Web Interface, implemented with React, is the UI via which users deploy, add subdomains, and associate repositories. It priorities accessibility and provides intuitive as well as guided flow experience . The Backend Service, implemented with Node.js and Express, handles core application logic, validates user input, and calls to external systems like DNS providers and version control systems. It is the orchestration layer that forwards the requests to the Deployment Engine.

The Deployment Engine is scripted and uses Bash to run automation activities like repository cloning, file copy, server config, and application startup. Docker support as a feature contains container-based deployment with support for static and dynamic applications. For automatic DNS setup, the platform had integration with Domain Registrar APIs (e.g., Cloudflare), where DNS records could be established automatically and routing policies secured in real time. The platform had some extra modules for event logging and authentication for offering traceability, access control, and operation security.

This design was architected with a view of being resource efficient in order to be deployable on small-scale infrastructure such as Linux-based VPS. Enterprise solutions are based on heavy-tier orchestration and high-order compute resources, but DomainForge consolidates essential automation features functionality to a less efficient design proportional to the restriction of small-scale development environments.

C. Automation and Deployment Workflow

DomainForge's automation process sought to consolidate critical deployment stages from source acquisition to live hosting into a unified process. The process starts when the user triggers a deployment request through the web UI, specifying the target repository and preferred subdomain. After validation, the Backend Service sends the configuration information to the Deployment Engine, which carries out a scripted set of automated steps.

The initial phase involves repository retrieval and environment setup, with Bash scripts employed to clone the specified GitHub repository and structure project files within the server directory.Based on type of application, the procedure is rewritten to both non-containerized configuration using reverse such like Nginx or Apache or containerized deployment using Docker images. Both ways have automated setup for services occurs without manual intervention, minimizing con- figuration drift. With the preparation for deployment, DNS provisioning is established by the assistance of Domain Registrar APIs. The system automatically creates subdomains and provides appropriate DNS records (A/AAAA or CNAME) so that all the applications deployed are addressable over a

distinct address. This integra- ition prevents the necessity for third-party DNS administration and it hosting is aligned with real time routing. Thereafter the workflow includes validation and service activation, where the auto-verifications verify for successful coexistence with the repositories, DNS-resolve ability, and application responsiveness. Close registering error logging in the Backend Service permits backtracking failure and retry if needed. The complete pipeline demonstrates the sys- tem's ability to automate what were once manual and siloed deployment procedure for a series of continuous automation pipeline.

D. Implementation Technologies and Execution

DomainForge was constructed atop a combination of cutting edge web technologies and scripting frameworks selected for their efficiency, efficacy, and suitability for low-resource environments. Node.js and Express were used to develop backend services so that they would provide efficient asynchronous request handling and secure API management. This layer provides mediating communications among modules, DNS updates, managing authentication, and automation calling scripts. Frontend user interface, built with React, is an interactive dashboard enabling users to create subdomains, connect GitHub repositories, and initiate deployment workflows. React has been employed to ensure responsiveness, modularity, and multi-device capabilities to the interface.

The deployment itself is carried out with Bash scripting for executing common automation tasks such as repository cloning, file copying, reverse proxy configuration (Nginx/Apache), and service boot. Docker is another runtime layer for containerized deployments to enable auto build and run of container images for such applications requiring isolated runtime environments. The combination mode of operation support both traditional server-based deployments and modern container workflows. Deployment and hosting are done on Linux-based Virtual Private Servers (VPS) because they are cost-effective and utilized by small groups. Domain Registrar API support i.e., Cloudflare ensures real-time creation and updating of DNS records eliminating the need for manual domain configuration. Additional supporting modules such as logging and authorization modules were implemented to guarantee operational security, traceability, and transparency throughout the release cycle. DomainForge went through a organized multi-stage testing cycle that had been established to measures reliability, precision, and usability across a variety of deployment environments. Testing started from the unit level where the individual constituents such such as automation scripts, API handlers, and DNS configuration modules was tested consciously. This ensured correct execution of repository cloning, DNS record configuration, and server provisioning activities were carried out properly. Particular care was taken to design error-handling mechanisms such that network problems or configuration mistakes would trigger diagnostic outputs rather than system failure.

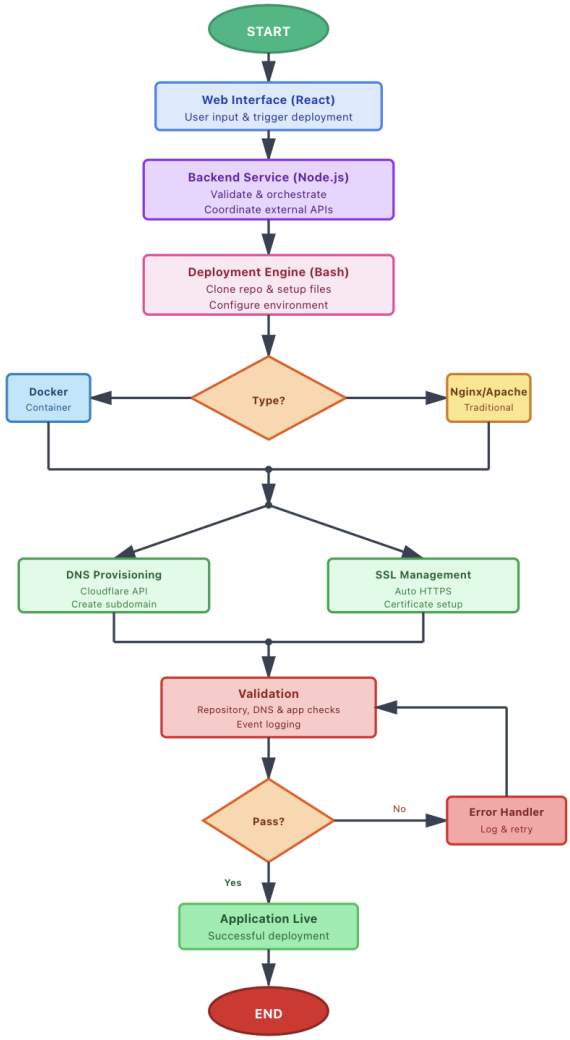

Fig. 1. DomainForge system architecture

E. Testing and Validation Strategy

Unit testing was followed by integration testing to verify end-to-end workflow continuity from user requests sent to executing the live application. It confirmed smooth interaction among the Web Interface, Backend Service, Deployment Engine, and external APIs. Test cases included deploying static and dynamic applications from GitHub repositories, validating subdomain accessibility through DNS propagation, and confirming service responsiveness in containerized and non-containerized modes. Orchestration was tested for latency, sync problems, and performance bottlenecks in the integration logs.

For real-world usage validation, user acceptance testing (UAT) was conducted with some selected student teams and small startup companies. They were asked to deploy sample applications using the web interface so that usability could be measured quantitatively . Feedback identified the benefit of eliminating hosts and DNS configuration steps manually but left room for improvement on build process status feedback.

Performance indicators like deployment time, rate of config errors, and system resource utilization were measured against typical manual deployment procedures. The results demonstrated reductions in setup errors and deployment performance, confirming the potential in real application for DomainForge under light conditions.

IV. Results and Discussion

DomainForge system deployment was contrasted with conventional manual deployment processes to assess deployment time savings, configuration correctness, and run-time reliability. For a variety of test cases on static and dynamic applications, DomainForge experienced a considerable decrease in deployment time. Manually deployed configurations that normally amounted to 10–15 minutes of recursive configuration iterations took approximately 2–4 minutes with the automated pipeline. This 75–80% time savings is largely the result of scripted task execution, integrated DNS provisioning, and minimal manual server interaction. Impressively, these improvements were seen consistently on modest virtual private server (VPS) infrastructure, further pointing toward the platform's suitability for small teams of developers without resident DevOps staff.

DNS configuration correctness also demonstrated good dependability in testing. API-based automation guaranteed subdomains and accompanying DNS records were provisioned without propagation failure or misconfiguration. Common manual deployments, on the other hand, resulted in 2–3 configurational errors per deployment cycle as a result of typographical or procedural errors. Resource profiling also confirmed that automation added minimal system overhead, which was still within reasonable ranges for even container-based deployment cases. Overall, these findings affirm that DomainForge significantly decreases operational overhead without compromising runtime stability.

User acceptance testing gave valuable usability and real-world usage feedback. Small development teams indicated that the web-based interface eliminated explicit command-line interaction, simplifying actions like repository linking and live hosting. The end-to-end flow assisted especially those with no system administration skillset. Although general feedback was favorable, participants indicated lack of added real-time build transparency and improved feedback requests in extended deployment actions as potential areas for improvement.

Addressing the future, some of the following enhancements were suggested in order to further speed up the functionality of DomainForge. Native CI/CD support would come with automatic testing and incremental rollouts for continuous delivery pipelines. Multicloud support with infrastructures This would increase deployment, for instance, AWS, Azure, and GCP. flexibility within infrastructures. Minimalistic addition of container orchestration, inspired by products such as Kuber- It would add scalability and self-healing in microservice. ecosystems. Other capabilities include real-time monitoring, alerting, and AI recommendations of deployment-are the next organic additions. These additions are targeted at evolving DomainForge while retaining its lightweight and easy-to-use design.

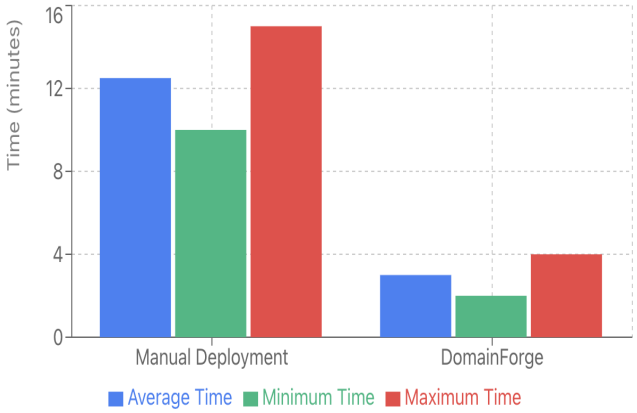

Fig. 2. Deployment Time Comparison

V. Limitations and Applications

A. Limitations

Despite having proved advantages, the current implementaThe formation of DomainForge has a number of shortcomings. which draw on the scope for further improvement:

- DNS and Platform Dependency: Proposed system will be dependent from specific DNS providers and hosting environments, which reduces portability across heterogeneous
- Security Considerations: Automation of processes without powerful authentication and control may expose the platform against unauthorized access and configuration misuse.
- Scalability Limitation:Handling many concurrent Subdomains can be performance bottlenecks, whereas the current resource allocation is not high. Throughput scenarios are optimized.
- Limited error handling and feedback: In diagnostic feedback in case of failures is small, possibly making debugging efficiency or operational transparence during deployment-related problems.
- Lack of advanced resilience mechanisms: Fault tolerance because both dance and rollback are limited, real-time recovery if there are any deployment anomalies.

B. Applications

DomainForge offers pragmatic utility to many diverse devel-The development and operational contexts, especially in teams featuring Poor DevOps support:

- Automated web hosting and subdomain provision Service: Allows hosting providers and administrators to quickly assign subdomains and deploy applications with mini- bad manual intervention.
- DevOps Enablement for Lightweight and Small Teams Startups: Offers browser-based solution for complex

CI/CD tools to support fast deployment for MVPs; prototypes, and producing microservices.

- Academic and Learning Environments Supports students, educators, and research programs repeatedly deploy web projects by themselves, without deep system administration. knowledge.
- Automation of Test and Staging Environment: Allows Consistent testing and sandbox environments creation for iterative development, QA validation and experimentation- tation.
- Foundation for Future Scalable Infrastructure: Provides a baseline to consolidate CI/CD workflows, monitoring, Ccoud formation and cloud orchestration for advanced DevOps Pipelines

REFERENCES

[1] Tokuc, K. (2024) Suitability of Micro-Frontends for an AI as a Service Platform. Doctoral dissertation, Hochschule für Angewandte Wissenschaften Hamburg.

[2] Hounsel, A., Borgolte, K., Schmitt, P., Holland, J. and Feamster, N. (2020) 'Comparing the Effects of DNS, DoT, and DoH on Web Performance', Proceedings of the ACM Internet Measurement Conference.

[3] Kelly, S. and Tolvanen, J.P. (2008) Domain-specific modeling: enabling full code generation. Chichester: John Wiley & Sons.

[4] K. Schomp, R. H. Lee, and A. Meske, "Providing Authoritative Answers to the World's Queries: Design and Operation of Akamai DNS," Akamai Technologies Research, 2020.

[5] Romera, C.L. (2020) DNS over HTTPS traffic analysis and detection. Open University of Catalonia.

[6] K. Khandaker, D. Trossen, J. Yang, and G. Carle, "On-path vs Off-path Traffic Steering, That Is the Question," Proceedings of the ACM/IEEE Conference on Internet Measurement / Networking, 2021

[7] L. Degani, A. Bianchi, and F. C. Ferri, "Generative Adversarial Networks for Subdomain Enumeration," Proceedings of the ACM Conference on Computer and Communications Security (CCS) / ACM Workshop on Cyber Security (2022).

[8] D. Koymans, W. Toorop and K. van Hove, "Subdomain Leakage: Investigating Exposure Methods and Malicious Exploitation," NLnet Labs / Master's Thesis / Workshop Paper, 2025.

[9] Tang, R. et al. (2021) 'A Practical Machine Learning-Based Framework to Detect Compromised Hosts by Subdomain Enumeration', SecureComm.

[10] Comparative analysis: S. R. Torres, "Subdomain Identification Strategies for Efficient Machine-Assisted Enumeration," SBC/Local Conference Paper / 2025, and "Comparative Analysis of Subdomain Enumeration Tools"

[11] Zhang, X., Song, C., Zhao, J., Xu, Z. and Deng, X. (2022) 'Deep subdomain learning adaptation network: A sensor fault-tolerant soft sensor for industrial processes', IEEE Transactions on Neural Networks and Learning Systems, 35(7), pp. 9226-9237.

[12] M. A. Rodriguez and R. Buyya, "Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions," Journal / arXiv, 2018. arXiv

[13] Souppaya, M. and Scarfone, K. (2017) Application Container Security Guide. NIST Special Publication 800-190.

[14] M. S. Islam Shamim, F. A. Bhuiyan, and A. Rahman, "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices," arXiv / Systems Security Workshop, 2020.

[15] I.Ahmad, "Container Scheduling Techniques: A Survey and Assessment," Journal / Elsevier, 2022.

[16] Saleh, S.M. et al. (2025) 'A Systematic Literature Review on Continuous Integration and Deployment Pipeline Security', arXiv preprint, arXiv:2506.08055.

[17] S. M. Saleh et al., "A Systematic Literature Review on Continuous Integration and Deployment Pipeline Security," arXiv / 2025 (or peer-reviewed SLR), 2025.

[18] S. Dhandapani et al., "Enhancing Software Supply Chain Security Through STRIDE-Based Threat Modelling of CI/CD Pipelines," arXiv / Conference Paper, 2025.

[19] Krishna, K. (2023) 'Integrating AI and Cloud Computing for Automated DevOps', World Journal of Advanced Research and Reviews, 18(03), pp. 1618–1627.

[20] Velaga, S.P. (2024) 'Case Studies of Successful CI/CD Pipeline Implementations for Machine Learning and AI', International Journal of Research and Analytical Reviews, 11(3), pp. 137–145.

[21] Houerbi, A. et al. (2024) 'Empirical Analysis on CI/CD Pipeline Evolution in Machine Learning Projects', arXiv preprint, arXiv:2403.12199.

[22] Forsgren, N., Humble, J. and Kim, G. (2018) Accelerate: The Science of Lean Software and DevOps. Portland: IT Revolution Press.

[23] R. Barnes, J. Hoffman-Andrews, J. Kasten, D. McCarney, "Automatic Certificate Management Environment (ACME)," IETF RFC 8555, 2019.

[24] J. Aas et al., "An Automated Certificate Authority to Encrypt the Entire Web (Let's Encrypt): Design and Experience," ACM CCS / USENIX or ACM Proceedings, 2019.

[25] A. Manousis et al., "Shedding Light on the Adoption of Let's Encrypt," Measurements / arXiv / IMC workshop, 2016.

[26] R. Barnes et al., "Analysis of ACME Adoption and Security in Automated Certificate Ecosystems," Proc. Internet Measurement Conf. (IMC), 2020.

[27] L. Degani and A. Bianchi, "Evaluating Subdomain Enumeration Techniques for Exposure Surface Analysis," Proc. IEEE Security Privacy Workshops, 2023.

[28] Gladston, A., Harish, T.R. and Keerthivasan, J.V. (2025) AutoGen AI: Unified Platform for Content and Code Creation. In: Generative AI in Software Engineering. Hershey, PA: IGI Global, pp. 133–152.

[29] Bagai, R., Masrani, P.R. and Najana, M.N. (2024) Implementing Continuous Integration and Deployment (CI/CD) for Machine Learning Models on AWS.