Embracing IaC Through the DevSecOps Philosophy

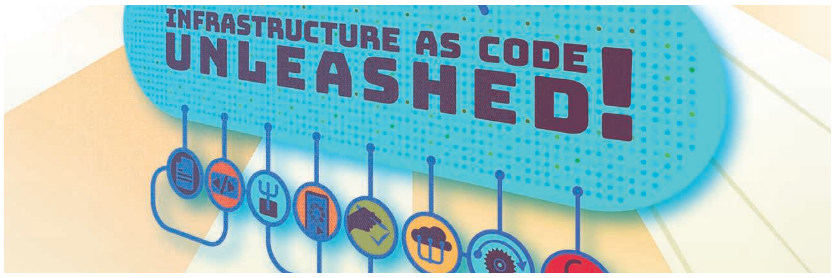
Concepts, Challenges, and a Reference Framework

Juncal Alonso, TECNALIA-Basque Research and Technology Alliance (BRTA)

Radosław Piliszek, 7bulls.com

Matija Cankar, XLAB d.o.o.

// We introduce the challenges of DevSecOps philosophy and its applicability to the development and operation of trustworthy infrastructure-as-code, and we combine the solutions into a single framework covering all crucial steps. Finally, we discuss how the proposed framework addresses the challenges and introduce an initial design for it. //



Digital Object Identifier 10.1109/MS.2022.3212194
Date of current version: 23 December 2022

INFRASTRUCTURE AS CODE (IaC)[1] is an approach enabling the automation of several deployments, configurations, and management tasks that otherwise would have to be performed manually by an operator. IaC brings many benefits in a cloud computing context as it saves time and resources when an application needs to be redeployed on a different set of resources or needs to be extended with new components or run on different cloud infrastructures. As such, IaC has represented a very important progress that has dramatically changed the work organization of many IT-intensive organizations (e.g., Netflix[2]). The main advantage of using IaC is repeatability: Once the right process is codified, it can be repeated as many times as desired in exactly the same way. IaC code, similar to any software, can be versioned and written by a collaborating team bringing together multiple expertise. In general, any benefit coming from traditional software design, management tools, and concepts is also a benefit for IaC software:[3] security by design, automation, testing, reusability, auditability, and so on. To this end, our proposal is that the application of DevOps and the extended DevSecOps[4] philosophy can enable the creation of such secure, reliable, and self-healed IaC software.

Introduction

In this article, we suggest a holistic approach and framework for the development and execution of trustworthy IaC, being it secure, integral, and self-healed. The remaining article is organized as follows: The challenges to operate trustworthy IaC are presented and discussed in the four stages of the DevSecOps process (see Figure 1). The tools and practices proposed to overcome these are detailed for each phase. The article ends up with the conclusions and the next steps to be addressed.

Challenges

The first set of general challenges affect all DevSecOps stages and are cultural. These challenges are the result of the paradigm shift, which is happening slowly and is deriving from the following:

- market fragmentation
- requirement of wide (IaC) skills
- definition of well-known IaC code patterns
- difficulty in replicating errors
- IaC languages specificities and tools heterogeneity
- security and trustworthiness
- configuration drift
- changing infrastructure requirements.

1. *Market fragmentation*:[5] Tools currently in the market are fragmented for infrastructure provisioning, configuration, deployment, and orchestration. The toolchain used to cover all DevOps phases depends on the ability of the DevSecOps teams to integrate them all. For instance, CloudFormation (https://aws.amazon.com/cloudformation/) could be used to set up the virtual machines and connect them to the network, Chef (https://www.chef.io/) to configure and secure the virtual machines, and Docker (https://www.docker.com/) to containerize the application. The fragmentation leads to the requirement of wide (IAC) skills.[6,7]
2. *Requirement of wide (IaC) skills.* Existing IaC languages and tools are based on different programming and executing paradigms. As such, using them in combination or passing from one to the other requires significant skills and reaching such levels require significant time in learning and training.

In addition to the previously mentioned overarching challenges, we identified more specific ones based on research findings, and[6,8] and composed a complete list of IaC challenges to be addressed by DevOps teams as follows:

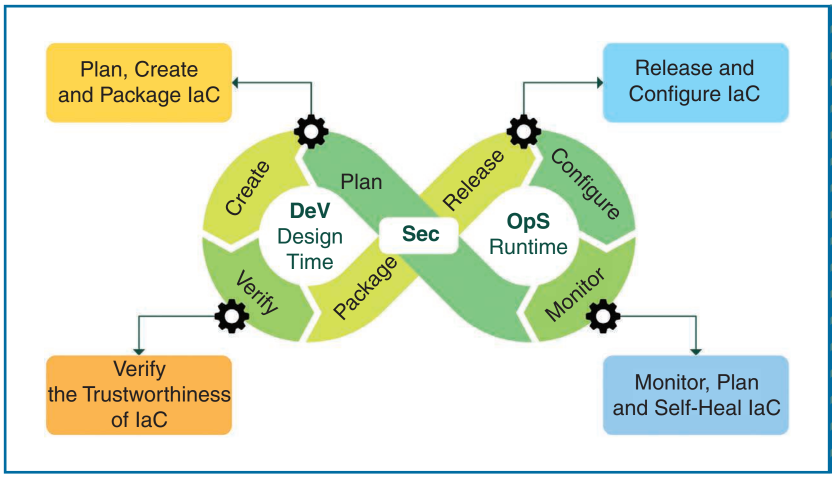
We divided the specific challenges into the four stages of the DevSecOps process presented in Figure 1.

The first DevSecOps phase, named *Plan*, *Create*, and *Package* the IaC, includes the following challenges:

3. *Definition of well-known IaC code patterns*: Patterns, techniques and architectural elements that need to be applied to improve quality objectives (e.g., caches for performance, load balancers for scaling, circuit breakers for reliability) and availability of deployments (e.g., canary releases, blue/green deployment) are well-known ones, but are not yet coded as well-defined IaC patterns.
4. *Difficulty in replicating errors*:[6] Like any other piece of code, IaC is subject to errors. Since the initial IaC is developed by humans, there is always the chance that it contains minor errors that will only produce impact after some time.
5. *The large variety of infrastructures to be provided can make a manual process quite cumbersome*:[7] Depending on the stage of the application, e.g., development, testing, integration, preproduction, or production, the infrastructure to be provisioned varies and the IaC is affected by these changes.

The second DevSecOps phase, named *Verify the trustworthiness* of IaC includes the following challenge:

6. *Security and trustworthiness*:[6] Systems created with IaC workflows are often large and complex to maintain. What seems like minor configuration change in the IaC code, can be spread out to different parts of the system (or other systems), producing issues. Due to the large number of



Plan, Create and Package IaC
Release and Configure IaC
Create
Plan
Release
Configure
DeV
Design
Time
Sec
OpS
Runtime
Verify
Package
Monitor
Verify the Trustworthiness of IaC
Monitor, Plan and Self-Heal IaC

FIGURE 1. The DevSecOps workflow with phases.

dependencies and the aim for stable environments, it is difficult to manually keep track of their potential security vulnerabilities and make updates when necessary.

The third DevSecOps phase, named *Release*, *Configure*, and *deploy IaC* includes the following challenges:

7. *Configuration drift*:[9] Once a system is created via an IaC workflow, any attempt to manually modify its configuration (even just to patch a security issue) will lead to a dangerous misalignment (config drift) between the actual system and its infrastructural code.

The last DevSecOps phase, named *Monitor*, *self-heal*, and *replan*,[5] focuses on the following:

8. *Changing infrastructure requirements*: The infrastructure requirements may change over time: for instance, one may need to move their application (or parts of it) from a private on-premises environment to a cloud/edge one or may have to change cloud provider or use more than one at the same time. This challenge also includes the maintainability of the IaC and its consistency to the changes and needs of the infrastructure.

Plan, Create, and Package

Designing Exploitable Abstractions of Execution Environments

Creating a new IaC is very complex job without tools or templates. That is why the design time (Figure 2) features an *integrated development environment (IDE)* that include a dedicated modeling language—DOML (the DevSecOps modeling language), similar to the one documented by Rahman et al.[6]—which allows describing the cloud applications and its infrastructural requirements without specificities of different IaC languages. Users can describe application layers together with an abstract infrastructural layer. With the help of integrated tools, the initial layers are augmented with a concrete infrastructure layer and optimized layer.

The framework, including *DOML Editor*, offers tools to analyze the DOML model finalized by users with respect to security, best practices, and validity constraints as well as static security testing tools to run checks against the resulting DOML. The latter step is covered by a *Model checker*: a verification tool that inspects the DOML application infrastructure blueprint and detects inconsistencies in the model and the application deployment template. The checker detects missing parts, dependency cycles, and various issues that can be detected from the high-level language perspective. After a successful check, the application blueprint is translated in the one or more target IaC languages like Terraform, OASIS TOSCA, and/or Ansible using an *IaC Generator*.

The IDE integrates all DOML related tools including the syntax coloring for DOML. When the IaC for a particular target language is created, the IDE also provides the packaging tool, to combine the content in a single file (CSAR—cloud service archive or Zip).

Optimization of the Deployment Configuration

The *infrastructure optimizer* or IaC optimization platform (IOP) is a tool designed to work directly with the DOML model to ensure that it describes an optimal infrastructure so that the created IaC is optimal. The IOP decisions are able to combine both resource-provider-declared as well as self-gathered historic data, which are kept and maintained by an internal service which is accessible also from the outer through an application programming interface (API) and a GUI. IOP uses this data to propose the optimal DOML that satisfies the user-provided constraints.

Verify Trustworthiness

To strengthen the infrastructure automation revolution, DevSecOps engineers deserve the same kind of tools and development environment as any other software developer. Our main goal is to allow the

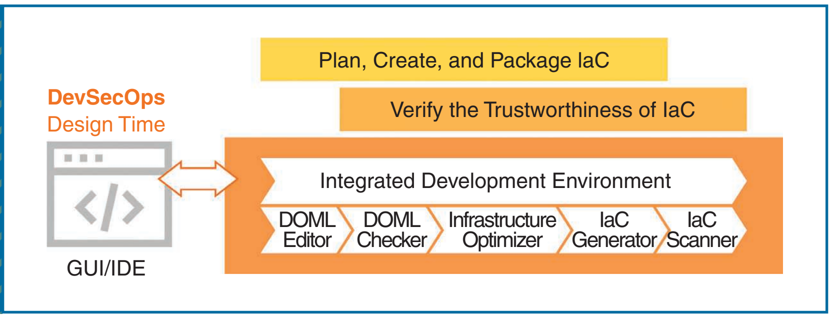


FIGURE 2. The design phase of DevSecOps.

DevSecOps team to work with infrastructural code as they do with a traditional application code, starting from the definition of requirements for the infrastructure—such requirements are expressed in terms of technical capabilities the application-level software should offer—to the design, implementation, verification, deployment, testing, operation, and monitoring of such infrastructural code. Treating the *infrastructure* specification as *code* instantly brings the code processing benefits such as modeling approaches, the usage of templates, repetitiveness, and automation. Moreover, we can apply code inspection tools that can check the code's validity and trustworthiness at the design time and inside the continuous integration/continuous delivery process.

DevSec: Static IaC Inspection

The static application security testing (SAST) checks guarantee us to shift security left as much as possible. SAST is a modern approach to test the application for known vulnerabilities, caused by type errors, misconfigurations, or software errors. One source of known vulnerabilities is OWASP (https://owasp.org/www-community/vulnerabilities/). In our case, the SAST is applied through three-fold verification and inspection of the infrastructure as a code as follows:

1. *Model checker*: Already presented in the "Plan, Create, and Package" section, this corresponds to a first step of security inspection executed over a high-level module and verifies a cohesiveness of module with the basic functional and non-functional requirements expressed from the user.
2. *IaC scanner (security inspector)*: The IaC scanner takes the IaC translated from the DOML and runs multiple linter and security tools to inspect the IaC. The goal is to find inconsistencies, typos, IaC-language-specific model errors, and run other checks available in open-source community or among proprietary services.
3. *IaC scanner (component security inspector)*: The IaC can depend on multiple components provided by community or IaC language developers. Errors and misconfigurations in IaC templates, IaC libraries, and collections (e.g., Ansible collections) expand attack surface and number of potential issues for our application. Each dependency should be verified (md5 check), inspected to detect misuse, inappropriate configuration, and outdated libraries suffering from known vulnerabilities. This component inspector is powered by the active knowledge database, where a set of services is gathering information of known components/dependencies, their release changes and security fixes.

To strengthen the infrastructure automation revolution, DevSecOps engineers deserve the same kind of tools and development environment as any other software developer.

Security inspection is a continuous task. Even though SAST approaches were designed to be used at the design time, they should also be scheduled later, to track the component updates, new discovered vulnerabilities, and zero-day exploits. A preliminary development of the DevSecOps engine for executing various IaC SAST checks is available on GitHub (https://github.com/xlab-si/iac-scan-runner/) effectively combining the B and C components described previously.

Release, Configure, Check, and Deploy

Git Repository: The Getaway to Runtime

After the solution is modeled, converted, and verified statically, the time has come to test the deployment and involve the runtime of the proposed framework (Figure 3). The way from design time to runtime goes via Git tooling, e.g., GitHub, GitLab, and Gerrit. The assumption is that the code is controlled and executed in a GitOps manner—this way the running IaC is known and kept archived, and the access control is kept like for any other kind of code. With a strict config, it is possible to avoid creating config drifts which again improves the overall security posture of the deployed solution. However, configuration drift can happen also during the runtime, which needs to be detected by monitoring and fixed with self-healing.

The Environment for Predeployment of IaC

On the boundary of design time and runtime, there is a tooling for IaC predeployment, the Canary Sandbox Environment (CSE) which allows testing of dynamic properties of the generated IaC. The CSE tooling allows to create opinionated local environments, such as OpenStack, and to mock APIs of public cloud providers. This is an optional but recommended step that proves useful to weed out issues in the IaC before they reach production. The issues caught at this stage are not normally possible to be caught at the static analysis step as they depend on the in-runtime application of the IaC. They include issues related to, e.g., assumptions about existing resources, interdependencies, and race conditions. The dynamic analysis involves running the IaC in vivo so any shortcomings of the IaC and its supporting tooling are thus exposed more easily.

IaC Execution and Orchestration

From the git repository, the IaC is handled by the runtime controller. The controller orchestrates the rest of the runtime tooling to achieve coherency in the runtime—the IaC execution manager (IEM) receives the IaC to deploy, and infrastructure advisor is configured to monitor the new deployment after the monitoring agent are ready. The target of the IaC can be any supported public cloud provider or a local cloud based on OpenStack, VMware, or any analogous solution. All changes to the infrastructure go through the runtime controller and the IEM, thus both tools are always aware of the current state of the deployment. Indeed, the self-healing mechanisms rely heavily on the robustness of this runtime stack.

Monitor, Self-Heal, Replan

SecOps: Dynamic (Runtime) Inspection

Live applications on the virtual infrastructure are constantly in threat of various incidents that affect the application performance, availability, integrity, and data safety. The infrastructure monitoring component tracks events with sensors and process them. The security incidents during the runtime are traced by applying dynamic application security testing (DAST) approaches with live security monitoring. The first step of threat detection, integrity checking, incident response, and compliance in our solution is tackled by the Wazuh tool (https://wazuh.com/) where the user can configure tests and track the application security state and test results through a GUI.

From a huge pool of AIOps methods[10] our next step of security inspection was selected. It relies on the power of the natural language processing (NLP) for system and

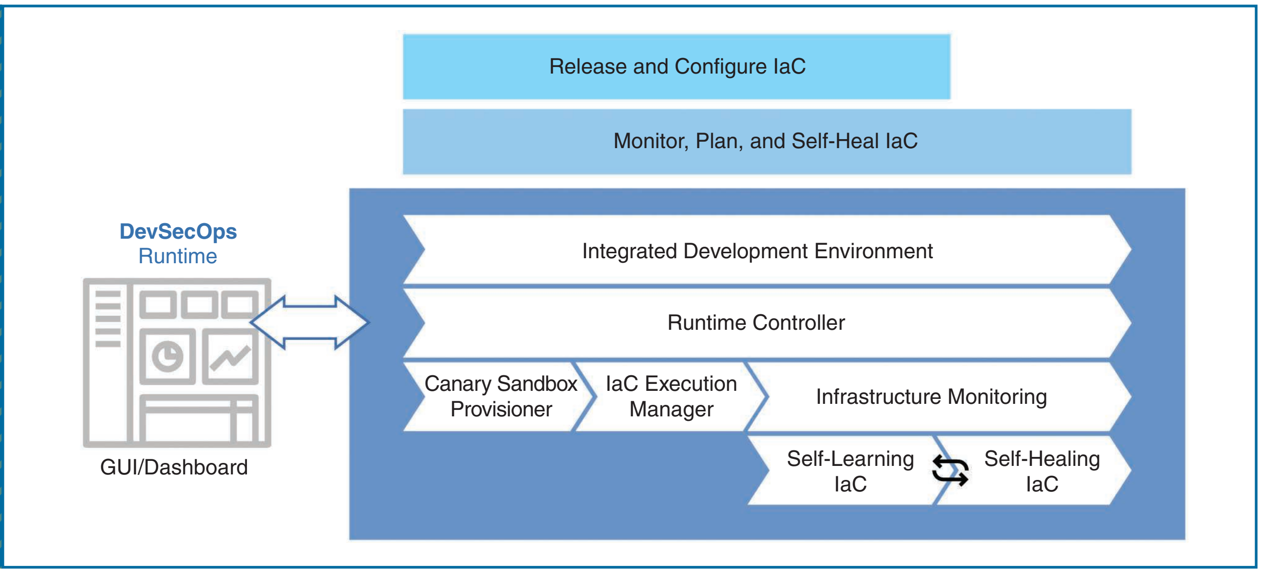

Release and Configure IaC
Monitor, Plan, and Self-Heal IaC
DevSecOps
Runtime
GUI/Dashboard
Integrated Development Environment
Runtime Controller
Canary Sandbox Provisioner
IaC Execution Manager
Infrastructure Monitoring
Self-Learning IaC
Self-Healing IaC

FIGURE 3. The runtime phase of DevSecOps.

application logs. The AI/ML-powered log monitoring inspector automatically analyses messages from historical logs, their severity, frequency, and format, and, with the power of deep learning techniques designed for NLP, understands the normal log flows in such a detail, that abnormalities are efficiently spotted. Based on self-learned knowledge, the system can perform unsupervised anomaly detection on the log data and issue notifications when application is behaving unexpectedly. Presented DAST approaches use triggers to interact with other runtime services that together maintain the application lifecycle and initiate *self-learning* and *self-healing* approaches to put the application back in order.

Table 1. The relationship of the challenges for trustworthy IaC development and the component addressing it in the proposed solution.

IaC challenge	DevSecOps framework component
1. Market fragmentation	DevSecOps framework (all components)
2. Requirement of wide (IaC) skills	DevSecOps framework (all components)
3. Definition of well-known IaC code patterns	IaC generator, IaC execution manager, infrastructure monitoring, self-learning IaC, self-healed IaC
4. Difficulty in replicating errors	IaC scanner, canary sandbox environment, infrastructure monitoring, self-learning IaC, self-healed IaC
5. IaC languages specificities and tools heterogeneity	DOML editor, IaC generator, IaC execution manager
6. Security and trustworthiness	Model checker, IaC scanner, infrastructure monitoring
7. Configuration drift	IDE, IaC generator
8. Changing infrastructure requirements	DOML editor, infrastructure optimizer, infrastructure monitoring, self-learning IaC, self-healed IaC

Secured and Self-Healed IaC

The infrastructure monitoring component receives the configuration request from the *runtime controller*. It then acts upon all its subcomponents (monitoring agents, time series data bases, self-learning, and self-healing) to set them up to work with the new deployment. There are two branches of the monitoring solution. One involves performance and general availability metrics, and the other involves dynamic security testing. In addition to "merely" monitoring the deployment, the DevSecOps framework offers self-learning, which applies machine learning to gain new knowledge from the metrics and events data—detect patterns and draw conclusions. Both raw monitoring as well as self-learning store the information so it can be exploited by the optimizer discussed previously. They can also signal severe abnormalities to the self-healing subsystem which creates an emergency plan and utilizes a runtime controller to enact it. The comprehensiveness of these components ensures that operators do not forget about the so-often-forgotten A in the CIA triad, nor do they have to worry about composing their own solution to monitoring of the different aspects of their infrastructure. (CIA Triad is an information security model, based on three principles—confidentiality, integrity, and availability.)

As discussed throughout this article, DevOps teams in general and IaC developers in particular, are facing a set of unsolved challenges in the management of IaC which prevent organizations from fully embracing this paradigm. Therefore, we proposed a DevSecOps framework to address the identified challenges in developing and maintaining *secure, integral,* and *self-healed* applications designed with IaC (see the overview in Table 1). The envisioned framework and novel concept of DevSecOps combined provide a solution that improves the consequences of the market fragmentation of the tools for infrastructure provisioning, configuration, deployment, and orchestration. The result minimizes the potential interoperability problems through the IaC tool chain and reduces the required skill set and time for mastering the DevSecOps approach. The proposed approach automatizes the process of trustworthy IaC creation from the design to the runtime. In all stages, the user has a guidance and control to achieve the best results with IaC creation and suggestions for integration of monitoring and self-healing mechanisms in the target application. The holistic management of the IaC tool chain increases the interoperability of the phases and decreases the learning curve.

ABOUT THE AUTHORS

JUNCAL ALONSO is a senior researcher at TECNALIA, Derio E-48160, Spain. Her research interests include software engineering for the cloud continuum, secure cloud federation, and multicloud-enabled software applications. Alonso received her Ph.D in cloud computing federation from the University of the Basque Country. Contact her at juncal.alonso@tecnalia.com.

RADOSŁAW PILISZEK is an IT solutions architect at 7bulls.com, Warsaw 00-582, Poland. His research interests include DevSecOps, pursuing agility and simplicity in the infrastructure as code movement, and hybrid and multicloud solutions, all using open source tooling, automation, and containerization wherever possible. Piliszek received his M.Sc. in informatics from the University of Bialystok. Contact him at rpiliszek@7bulls.com.

MATIJA CANKAR is a researcher and project manager at XLAB d.o.o., Ljubljana 1000, Slovenia. His current research interests include cloud automation, orchestration, infrastructure as code inspection, applying open standards (e.g., OASIS TOSCA), as well as open source technologies and delivering high-end deployment automatization solutions to end users. Cankar received his Ph.D. from the University of Ljubljana for his work on the efficient resource allocation in grid and cloud computing systems. Contact him at matija.cankar@xlab.si.

The first proofs of concept of the solution have been released in the end of 2021 as part of the PIACERE H2020 project (https://www.piacere-project.eu/). The initial evaluation of the proposed solution is taking place into industrial pilots addressing three different business domains (i.e., public administrations, critical maritime infrastructures, and public safety on IoT in 5G) that guide soliton development with concrete requirements and challenges from the field work. The final fully functional and validated solution is planned to be released in November 2023.

Acknowledgment

This work was supported in part by the European project PIACERE (Horizon 2020 research and innovation Program, under grant 101000162).

References

1. K. Morris, *Infrastructure as Code.* Sebastopol, CA, USA: O'Reilly Media, 2016.
2. "How we build code at Netflix." Netflix Tech Blog. [Online]. Available: https://netflixtechblog.com/how-we-build-code-at-netflix-c5d9bd727f15
3. M. Guerriero, M. Garriga, D. A.Tamburri, and F. Palomba. "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," in *Proc. 2019 IEEE Int. Conf. Softw. Maintenance Evolution (ICSME)*, pp. 580–589, doi: 10.1109/ICSME.2019.00092.
4. L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Reading, MA, USA: Addison-Wesley, 2015.
5. C. Siebra, R. Lacerda, J. Peixoto, I. Cerqueira, F. Da Silva, and A. Medeiros, "From theory to practice: The challenges of a DevOps infrastructure as code implementation," in *Proc. ICSOFT 2018 - 13th Int. Conf. Softw. Technol.*, pp. 1–10, doi: 10.5220/0006826104610470.
6. A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Inf. Softw. Technol.*, vol. 108, pp. 65–77, Apr. 2019, doi: 10.1016/j.infsof.2018.12.004.
7. M. Wurster et al., "The essential deployment metamodel: A systematic review of deployment automation technologies," *SICS Softw.-Inensive Cyber-Phys. Syst.*, vol. 35, nos. 1–2, pp. 63–75, 2020, doi: 10.1007/s00450-019-00412-x.
8. K. Indika et al., "The do's and don'ts of infrastructure code: A systematic grey literature review," *Inf. Softw. Technol.*, vol. 137, p. 106,593, Sep. 2021, doi: 10.1016/j.infsof.2021.106593.
9. G. Falazi et al., "On unifying the compliance management of applications based on IaC automation," in *Proc. 2022 IEEE 19th Int. Conf. Softw. Architecture Companion (ICSA-C)*, pp. 226–229, doi: 10.1109/ICSA-C54293.2022.00050.
10. P. Notaro, J. Cardoso, and M. Gerndt, "A survey of AIOps methods for failure management," *ACM Trans. Intell. Syst. Technol.*, vol. 12, no. 6, p. 45, 2021, doi: 10.1145/348342.