Quality Assurance in Infrastructure as Code: Issues, Approaches, and Open Challenges

Haitam El Hayani[a], Jolan Philippe[b], Stéphanie Challita[a], Olivier Barais[a], Benoît Combemale[a]

[a] *INRIA, IRISA, Université de Rennes, Rennes, France*
[b] *Université d'Orléans, INSA CVL, LIFO, UR 4022, Orléans, France*

Abstract

Infrastructure as Code (IaC) has emerged as a cornerstone of modern DevOps practices, enabling automated, repeatable, and scalable management of cloud and software infrastructure. However, the growing complexity of IaC scripts introduces quality issues, including defects, smells, and anti-patterns that can compromise system reliability, maintainability, and performance. To address these challenges, researchers have proposed various support mechanisms, such as linters or other static analysis tools, to assist DevOps engineers in producing higher-quality IaC scripts. However, existing knowledge remains fragmented across technologies, quality issues, and developed support approaches.

This Systematic Literature Review synthesizes the state of the art by investigating the characteristics of quality issues, and the methods developed to mitigate them. Our findings reveal the community's focus on security and operational issues, dominance of technology-specific studies, and a predominance of static analysis, with techniques that are mostly rule- and graph-based. We consolidate these insights in a feature model-based taxonomy and identify major gaps, including limited attention to performance, dynamic, and run-time behavior of infrastructure. These results imply a need for a holistic, technology-agnostic, and lifecycle-aware IaC quality assurance methods that reflect the complexity of modern cloud infrastructures.

Keywords: Infrastructure as Code, Quality Assurance, DevOps, Cloud Computing

1. Introduction

DevOps has emerged as a practice to unite development and operations teams for improved Software Delivery and Operational (SDO) performance Forsgren et al. (2019). Achieving such performance improvements requires the automation of tasks traditionally performed manually across the software lifecycle. Automating the management of *infrastructure resources* is a key step of the automation process. In this paper, we adopt a broad view of *infrastructure* as encompassing not only computing (e.g., virtual machines, containers, servers), storage (e.g., databases, file systems), and networking resources (e.g., user profiles, IP addresses) Morris (2020), but also the software, platforms, and hardware that deliver or deploy applications to production Wang (2022).

Infrastructure as Code (IaC) has emerged as the de facto approach for automating infrastructure management by enabling the definition, provisioning, and configuration of deployment environments through machine-readable source code Humble and Farley (2010). Like traditional software code, IaC scripts can become large, evolve frequently, and involve contributions from multiple developers Jiang and Adams (2015). Consequently, similarly to software code, they are prone to **quality issues** such as defects, bugs, smells, and anti-patterns that can compromise system reliability, security, and performance Begoug et al. (2023); Chiari et al. (2022); Hasan et al. (2020); Jiang and Adams (2015); Rahman et al. (2019b).

Recent outages at major cloud providers underscore the severe consequences of infrastructure issues. For instance, Amazon Web Services (AWS) suffered a significant failure due to a DNS failure resolution impacting millions of users, and companies across more than 60 countries.[1] After this incident, Microsoft Azure had an outage linked to a configuration error in Azure Front Door impacting critical services like Microsoft 365 and authentication systems.[2] More recently, a major Cloudflare incident was caused by a configuration file, automatically generated by an internal system, that exceeded its size limit and triggered a cascading failure across their global network, affecting access to widely used applications and websites across the web.[3] These recent incidents highlight how even leading cloud and infrastructure providers remain vulnerable to internal defects and configuration failures that can cascade into large-scale service disruptions.

Recognizing these risks in misconfigured or defective infrastructure definitions, researchers have proposed a variety of approaches to support IaC quality assurance. Prior work spans detection of code smells, development anti-patterns, and security and maintenance defects across different IaC tools. For example, frameworks such as GLITCH enable polyglot smell detection across different IaC technologies, while other studies systematically identify security smells or development anti-patterns across repositories Saavedra and Ferreira (2023); Rahman et al. (2021b, 2020b). However, despite the diversity of these contributions, the literature remains fragmented across the IaC technologies, the types of quality issues, and the detection

[1] `https://www.infoq.com/news/2025/11/aws-disruption-cloud/`
[2] `https://azure.status.microsoft/en-gb/status/history/`
[3] `https://cybernews.com/news/cloudflare-outage-internet-down/`

approaches.

The organizational reality of DevOps practice further amplifies this fragmentation. López-Fernández et al. (2022) highlight that modern DevOps adoption forces organizations to merge isolated roles, such as development, operations, security, and quality assurance, into complex cross-functional product and platform teams. Because these practitioners bring differing functional perspectives and rely on diverse toolchains, they often lack a unified vocabulary.

In such environments, both researchers seeking to position new work, or identify existing gaps, and practitioners building multi-layered IaC (e.g., Terraform for provisioning and Kubernetes for orchestration) QA strategies must synthesize findings scattered across technology-specific studies, reconcile incompatible quality terminologies, and navigate technique classifications with no consolidated view of what has been studied, what works, and what remains unaddressed.

To address this gap, we conduct a technology-agnostic Systematic Literature Review (SLR) that treats IaC as a unified discipline rather than a collection of individual tools. While our focus on foundational terminology involves a deliberate trade-off, potentially missing studies that use only technology-specific language, it offers a consolidated evidence base, for both researchers and practitioners, of IaC quality assurance landscape by examining the following research questions:

- **RQ1:** What quality issues have been investigated in IaC research, and what are their characteristics?
- **RQ2:** What support approaches have been proposed in the literature, and what are their properties?

By answering these questions, this review makes four key contributions: (i) A categorization of IaC quality issues and their characteristics, (ii) A categorization synthesizing existing approaches to improve IaC quality, (iii) A taxonomy of IaC quality assurance approaches, and (iv) An identification of open challenges and future research directions for IaC quality assurance. Through these contributions, this review serves both audiences. For researchers, it provides a structured map for navigating the research space (Section 6.1). For practitioners, it can offer a lens for auditing existing QA toolchains against the quality dimensions derived from the scientific literature (Section 6.2). In doing so, the review brings together the scientific community and practitioners, and opens the door to improve underaddressed areas of IaC quality assurance.

2. Background

This section provides the background needed for the rest of the review. First the definition and scope we adopt for Infrastructure as Code, then the quality issues that affect IaC scripts.

2.1. Infrastructure as Code (IaC)

IaC has emerged as a central practice within DevOps for automating infrastructure management. By expressing infrastructure resources in machine-readable source code, IaC enables the repeatable, consistent, and scalable provisioning and configuration of deployment environments Wang (2022).

Despite its widespread adoption, the scientific community has not yet converged on a unified definition of IaC. Some authors adopt a narrow perspective, for example, for Drosos et al. (2024) IaC refers to configuration management systems, thereby excluding provisioning tools. Whereas, Eng et al. (2024) classify Docker and Docker Compose as part of the IaC ecosystem, thereby extending the concept to containerization technologies.

In an effort to structure this landscape, Wang (2022) further proposed a three-part categorization of IaC tools:

- **Infrastructure provisioning:** automates the allocation and management of infrastructure resources, typically through tools like Terraform and AWS CloudFormation.[4]
- **Configuration management:** automates the specification of infrastructure configurations; tools such as Ansible and Puppet encode desired system states to minimize errors and improve reproducibility.
- **Image building:** automates the creation of standardized machine and container images; tools like Packer [5] create machine images, while Docker [6] enables containerization for application portability.

While this categorization captures much of the IaC ecosystem, it does not explicitly account for **orchestration** technologies, which are increasingly central to modern cloud-native systems (e.g., Kubernetes, Docker Swarm). Container orchestration platforms are systems designed to manage the deployment of containerized applications in large-scale clusters, they facilitate the execution and monitoring of these applications by transparently managing tasks and data deployed on a set of distributed resources Rodriguez and Buyya (2019). In the context of modern DevOps and cloud-native practices, the boundary between infrastructure provisioning and application runtime orchestration has become increasingly blurred. Orchestrators do not merely schedule workloads; they abstract and manage underlying resources, such as networking (services, ingress), storage (persistent volumes), and compute constraints, essentially defining the infrastructure from the application's perspective. Through this nature, orchestration acts as an extension of IaC, utilizing the same core principles of declarative configuration and automated reconciliation to manage the platform's state. Consequently, we adopt a broad definition of IaC that encompasses orchestration specification languages, as they effectively treat the entire runtime environment as code. Then, to ensure comprehensive coverage of current practice, we extend the categorization of Wang (2022) by explicitly including **orchestration** as a fourth category. Accordingly, in the remainder of this article, we classify IaC tools into four categories: infrastructure provisioning, configuration management, image building, and

[4] `https://docs.aws.amazon.com/cloudformation/`
[5] `https://developer.hashicorp.com/packer/docs`
[6] `https://docs.docker.com/`

orchestration. Throughout this article, we use the term practitioner to refer to end-users of IaC tools, including DevOps engineers and infrastructure developers.

2.2. Quality issues

Like traditional source code, IaC scripts are susceptible to various **quality issues**, including defects, smells, and anti-patterns Reis et al. (2023); Rahman et al. (2020a); Drosos et al. (2024); Laplante (2007); Brown et al. (1998), which can impact critical quality attributes such as security, reliability, and maintainability Rahman and Sharma (2022). These issues can be differentiated based on their nature, **defects** are concrete imperfections or unintended behaviors requiring immediate correction def (2010), while **code smells** are structural or design weaknesses that may hinder maintainability or extensibility, flagging potential areas for concern rather than immediate errors Reddy Konala et al. (2023). In contrast, **anti-patterns** represent recurring poor practices that lead to negative outcomes and should be actively avoided Brown et al. (1998); Rahman et al. (2020b). Empirical studies confirm the prevalence of these problems in the IaC domain, revealing common defects like misconfigured data or idempotency violations Rahman et al. (2020a), and the existence of code smells, like Terraform sustainability-related ones Kosbar and Hamdaqa (2025). Additionally, even experts have low expert consensus on IaC smells, suggesting subjectivity and context dependence Masoumzadeh et al. (2025). Collectively, these findings support giving a broader definition of IaC **quality issues** as a combination of these elements, encompassing both erroneous behaviors and structural weaknesses that degrade overall system's quality.

3. Related work

Research on IaC quality has emerged along several directions, ranging from practitioner-oriented studies to technology-oriented investigations and systematic reviews. Table 1 summarizes key related works by their IaC focus, methodology, IaC layer focused on, quality issue(s) discussed, and nature of support approaches discussed. This section contextualizes our work within these strands and highlights how our study differs in scope and objectives.

3.1. Practitioner's perception of IaC challenges

A number of studies have examined the challenges practitioners face in adopting and using IaC. Kumara et al. (2021) extracted community-recommended best practices and pitfalls from blogs and online sources, offering insight into practitioner-driven quality guidelines. Guerriero et al. (2019) surveyed practitioners, showing that while IaC is widely adopted, debugging, testing, and learning curves remain major challenges. Begoug et al. (2023) mined Stack Overflow, identifying seven recurring themes of practitioner difficulties, including deployment pipelines, server configuration, and file management. Hasan et al. (2020) studied online artifacts about testing IaC, and identified six testing practices, namely, the use of automation, sandbox testing, remote testing, testing every IaC change, behavior-focused test coverage, and avoiding coding anti-patterns. However, they concluded that testing remains underdeveloped, because of reasons like, lack of known IaC testing practices, differences in environment testing compared to General Programming Languages (GPL), and potential unnecessary costs since testing instances must be destroyed after use. Reis et al. (2022) highlighted developer struggles with correctness, maintainability, and reuse of images, noting a reliance on trial-and-error approaches. Together, these studies reveal that practitioners face systematic quality issues across IaC layers, justifying the need for more comprehensive assurance approaches.

3.2. Technology oriented studies of IaC quality

Having explored the existing research on practitioners' perceptions of IaC challenges, in this section we examine related work driven by specific technological choices. For instance, Rahman and Sharma (2022) summarized research on key quality issues for **Puppet** scripts, primarily focusing on security and maintainability, to offer actionable recommendations. Schwarz et al. (2018) investigated the applicability of existing Puppet and software engineering smells to **Chef**. As a result they suggested a catalogue of 17 Chef smells, but it lacks generalizability evaluation against other IaC platforms. Similarly, Drosos et al. (2024), developed a bug taxonomy by analyzing 360 configuration-based IaC system (Ansible, Puppet, and Chef) bugs and investigated the symptoms, causes, reproduction requirements, and fixes. Pahl et al. (2025) took a broader approach by reviewing IaC technologies through a DevOps-specific framework and discussing research challenges, including defect categories and quality issues in IaC scripts. Finally, Diarra et al. (2024) addressed **Kubernetes**, evaluating 16 manifest verification tools and introducing a set of 13 criteria to help practitioners select appropriate tooling.

3.3. Systematization of IaC research

Early mapping studies, such as Rahman et al. (2019a), provided a high-level categorization of IaC research topics, like frameworks, adoption, and testing, but did not deeply investigate the quality aspect. Similarly, systematic reviews by Wurster et al. (2020) and Özdoğan et al. (2023) focused primarily on the structural comparison, deployment features, and architectural classification of the IaC technologies themselves (e.g., declarative vs. procedural), rather than the quality of the scripts they produce.

To position our work against the closely related QA surveys, we compare them along six dimensions, as presented in Table 2, and which we introduce at a high level; for the more in depth understanding you can refer to the sections indicated.

- **IaC Layer Coverage:** which layers of the infrastructure stack the study addresses: provisioning, configuration management, orchestration, or image building (Section 5.1.3).

- **Quality Issue Categorization:** whether the study classifies the quality issues it studies into a structured set of categories (Section 5.1.1).
- **Technique Categorization:** whether the study classifies the techniques it studies into a structured set of categories (Section 5.2.2).
- **Support Context:** whether the study characterizes when and at what scope support is applied, i.e., the life-cycle phase (Section 5.1.5) and the analysis granularity (Section 5.2.5).
- **Technology Dependency:** whether the study reports how far its issues and solutions are tied to a specific IaC technology versus being technology-agnostic (Section 5.1.2).
- **Empirical Validation:** whether the study reports tool support, evaluation context, and replication artifacts for the approaches it covers (Section 5.2.1).

The studies most closely related to our work are the survey by Chiari et al. (2022) and the Systematization of Knowledge (SoK) by Reddy Konala et al. (2023). Both provide valuable catalogs of detection techniques and tools; however, they explicitly restrict their scope to static analysis, thereby overlooking dynamic and hybrid approaches that are essential for capturing run-time and environment-dependent behaviors of infrastructure. In addition, their coverage of the IaC ecosystem remains partial. While Chiari et al. (2022) primarily focuses on configuration management and provisioning, and Reddy Konala et al. (2023) extends this to image building, both omit orchestration, which is increasingly central in modern cloud-native systems. Furthermore, existing QA surveys lack analysis of support context by missing life-cycle phase at which issues arise and the granularity at which detection is performed. Although Chiari et al. (2022) partially address this by distinguishing between design and implementation smells, neither study systematically classifies QA methodologies across the full support context. Finally, Reddy Konala et al. (2023) evaluated the empirical validation of the QA solutions based on their maturity, and DevOps integration, whereas Chiari et al. (2022) partially addressed this by reporting the evaluation scores of the studied tools.

To address these limitations, our SLR introduces a holistic and multi-dimensional synthesis of the IaC QA landscape. As shown in Table 2, we systematically cover all major IaC layers, provisioning, configuration management, orchestration, and image building, and extend the methodological scope beyond static analysis to include dynamic and hybrid techniques, enabling the study of both static and run-time quality issues. Moreover, we explicitly analyze the support context of QA approaches through multiple dimensions, including life-cycle phase, technology dependency, and analysis granularity. We also provide an assessment of empirical validation by examining tool support, evaluation context, and replication artifacts.

Finally, we consolidate these dimensions into a feature-model-based taxonomy, offering a structured framework to reason about QA coverage (Section 6), identify blind spots, and highlight future research opportunities (Section 7).

4. Methodology

We conduct this study as a Systematic Literature Review (SLR), following established guidelines by Kitchenham et al. (2007) and Petersen et al. (2015). The goal is to ensure transparency, and replicability in synthesizing the body of knowledge on quality issues and support approaches for IaC development.

4.1. Search strategy

The search strategy for this review was designed in three steps. We first construct a quasi-gold set of known-relevant studies, then describe the design of the query itself, and finally validate the query against the quasi-gold set to assess its recall and compare it against alternative formulations

4.1.1. Quasi-gold set construction

Following the guidance of Zhang et al. (2011) on validating literature-review searches, we constructed a quasi-gold set (QGS) of known-relevant IaC QA studies through a structured, three-step procedure:

- **Venue and time-span identification:** We defined the frame of the manual search as 5 journals and 17 international conferences in which IaC quality is susceptible to appear, over the period from 2015 to 2025, identified from the publication venues of well-known contributors, complemented by venues from the authors' domain expertise. Among the 22 venues, only 4 of the journals (Q1-ranked journals (SCImago[7])), and 8 of the conferences (A/A*-ranked conferences CORE2026[8]) had papers about IaC quality. The complete venue list, together with the rationale for each venue, is documented in the replication package (El Hayani et al., 2026).
- **Manual search:** For each selected venue, we screened all publications appearing in the defined period, applying the same inclusion and exclusion criteria as the main screening process (Section 4.2.3). The manual search retained 50 primary studies in total. Per-venue screening records are provided in the replication package.
- **Supplementary studies:** We complemented the manual search with studies cited as foundational by the closest prior reviews (Chiari et al. (2022), Reddy Konala et al. (2023)) that were published at venues outside the selected set. Three such studies were added, each explicitly documented as a supplement.

[7] https://www.scimagojr.com/
[8] https://portal.core.edu.au/conf-ranks/

Table 1: Summary of Related Work on Infrastructure as Code (IaC). IaC layer refers to image building (IB), provisioning (Pr), configuration management (CM), and orchestration (Or). The support refers to the overall kind of analysis: static (S), dynamic (D) or hybrid (H).

Study	IaC Focus	Methodology	IaC layer	Quality issue	Support
Practitioners' Perception					
Kumara et al. (2021)	Ansible, Puppet, Chef	Blog and online artifact mining	CM	Good/Bad practices	–
Guerriero et al. (2019)	General IaC	Semi-structured interviews	All	Good/Bad practices	S, D
Begoug et al. (2023)	General IaC	Stack Overflow mining	CM, Pr	Real-world practitioner challenges	–
Hasan et al. (2020)	Ansible, Puppet, Chef, Terraform	Online artifact analysis	CM, Pr	Testing practices	S, D
Reis et al. (2022)	Docker, Docker-compose	Online survey	IB, Or	Developer's challenges	–
Technology-Oriented Studies					
Rahman and Sharma (2022)	Puppet	Literature review	CM	Code smells/anti-patterns, configuration drift, and idempotence/dependency faults	S, D
Schwarz et al. (2018)	Chef	Empirical evaluation of code smells	CM	A catalog of 17 Chef-specific code smells	S
Drosos et al. (2024)	Ansible, Chef, Puppet	Qualitative bug analysis (360 bugs)	CM	Common bug symptoms, and root causes	S, D
Pahl et al. (2025)	General IaC	Technology review	CM, Pr	Defect categories, code smells, anti-patterns, configuration drift, and architecture conformance	S, D
Diarra et al. (2024)	Kubernetes	Evaluation of 16 manifest verification tools	Or	Characterized errors in Kubernetes manifests	S
Systematization of IaC Research					
Rahman et al. (2019a)	General IaC	Systematic mapping study	CM, Pr, Or	Defects, Code smells, and anti-patterns	–
Wurster et al. (2020)	General IaC	Systematic review	CM, Pr, Or	Manual deployment errors, migration complexity, technology lock-in, and lack of interoperability	–
Özdoğan et al. (2023)	General IaC	Systematic gray literature review	CM, Pr	Configuration errors and code smells	–
Chiari et al. (2022)	General IaC	Systematic literature review	CM, Pr	Code smells	S
Reddy Konala et al. (2023)	General IaC	Systematization of Knowledge (SoK)	CM, Pr, IB	Code smells and functional defects	S

Table 2: Comparison of our review with the closely related IaC QA surveys (✓: addressed; ✗: not addressed; ∼: partially addressed).

Study	IaC layer	Quality issue categorization	Technique categorization	Support context	Technology dependency	Empirical validation
Chiari et al. (2022)	CM, Pr	∼	✓(S)	∼	✓	∼
Reddy Konala et al. (2023)	CM, Pr, IB	✗	✓(S)	✗	✗	✓
This work	CM, Pr, Or, IB	✓	✓(S, D, H)	✓	✓	✓

The resulting QGS comprises 53 primary studies, it serves as a recall benchmark: each candidate query is scored by the *quasi-sensitivity* metric (QSM) defined by Zhang et al. (2011) in Equation 1, which measures the proportion of gold-set studies retrieved by the query.

$$\text{QSM} = \frac{\#\,\text{quasi-gold set studies retrieved by the query}}{\#\,\text{studies in the quasi-gold set}} \quad (1)$$

Against this QGS, we compared the candidate queries discussed during query construction (Section 4.1.2); the results of this comparison are reported in Section 4.1.3 .

4.1.2. Query construction

The search strategy was designed to maximize coverage while maintaining relevance. To this end, we aligned the search process with our research questions (Section 1) and the broad definitions of IaC and quality issues introduced in Section 2.

We conducted an exploratory search phase on Google Scholar to identify commonly used terminology and alternative formulations in the literature. This phase aimed to capture the diversity of terms used to describe IaC quality concerns, and evaluate candidate query structures under the syntactic and operator constraints imposed by major digital libraries. During this phase, we experimented with multiple query formulations targeting different aspects of the domain, including:

```
Q1: Quality improvement centric: ("Infrastructure as Code" OR "IaC
     ") AND ("Quality assurance" or "Testing" OR "Verification" OR
     "linting" OR "Static Analysis" OR "Dynamic analysis")

Q2: Issue-centric: ("Infrastructure as Code" OR "IaC") AND ("
     defects" OR "errors" OR "anti-patterns" OR "vulnerabilities"
     OR "code smells" OR "Quality issues" OR "bad practices" OR "
     Misconfiguration")

Q3: Survey-oriented: ("Infrastructure as Code" OR "IaC") AND ("
     taxonomy" OR "classification" OR "categorization" OR "survey")
```

The exploratory phase revealed that relevant studies consistently rely on a core set of recurring quality-related terms (e.g., defect, smell, analysis, verification), while more specific keywords introduce technological or topical bias. In particular, including explicit IaC technologies (e.g., Docker, Ansible, Terraform) term would disproportionately favor studies centered on those technologies. Additionally, adding the different IaC technologies to the search query is arguably not effective as we cannot be exhaustive regarding all the IaC technologies, or the emerging ones. Therefore, we deliberately designed a generic, and technology-agnostic query to ensure broad coverage and avoid over-representation of specific tools, platforms, or subdomains. By relying solely on the general concept of "Infrastructure as Code" and quality assurance terminology, the query reflects the scientific community's collective understanding of IaC as a unified discipline rather than a collection of tool-specific studies.

As a result of this exploratory phase we suggest the following candidate search query for our SLR:

```
Q4: Candidate query for the SLR: ("Infrastructure as Code" OR "IaC
    ") AND ("defect" OR "bug" OR "smell" OR "anti-pattern" OR "
    testing" OR "analysis" OR "verification")
```

Due to variations in syntax and operator constraints across digital libraries, the search queries' syntax was slightly adapted for each platform while preserving its semantics. We executed the queries across four major software engineering databases, ACM Digital Library, IEEE Xplore, SpringerLink, and ScienceDirect, covering publications up to July 29, 2025.

4.1.3. Query validation

Using the quasi-gold set defined in Section 4.1.1, we evaluated the four queries presented in Section 4.1.2 (Q1–Q4), plus a fully-merged variant (Q5) that combines all topical terms of Q1 and Q2. Q5 is included specifically to test whether broader topical coverage retrieves gold-set studies missed by the more selective Q4.

```
Q5: ("Infrastructure as Code" OR "IaC") AND ("defect" OR "bug" OR "
    smell" OR "anti-pattern" OR "vulnerability" OR "
    misconfiguration" OR "bad practice" OR "error" OR "fault" OR "
    testing" OR "analysis" OR "verification" OR "linting" OR "
    quality assurance" OR "static analysis" OR "validation")
```

Three observations follow from Table 3. First, Q4 achieves a QSM of 85%, above the 70–80% adequacy threshold suggested by Zhang et al. (2011) and well above the narrower formulations Q1–Q3. Second, expanding Q4 to Q5 produces no additional gold-set hits, the same 8 studies remain missed. Q4 is therefore preferred over Q5 because it matches Q5 on validated recall while retrieving substantially fewer candidates, and because it can be executed as a single query on all four databases without the splitting and result merging that Q5's length forces on some libraries.

Third, the eight gold-set papers missed by both Q4 and Q5 share a common failure pattern, each describes a specific technology or terminology (*Puppet*, *TOSCA*, *cloud infrastructure code*) without invoking "Infrastructure as Code" or "IaC" in its title or abstract. Four concern container technologies (Durieux (2024), Rosa et al. (2022), Bui et al. (2023), Xu et al. (2024)), two mainly focus on Puppet (van der Bent et al. (2018), Weiss et al. (2017)), and two are phrased in general configuration or cloud terms (Cannavacciuolo and Mariani (2022b), Brogi et al. (2017)).

The failure therefore lies in the technology-agnostic foundational term group, not in the topic group. We considered enlarging the foundational group with terms such as *configuration* or *cloud orchestration*, but these are broadly used outside the IaC context and would have inflated the candidate set with out-of-scope studies. Therefore we kept the main part of the query restricted to the foundational terms, and the resulting limitation is acknowledged in Section 8.3.

This limitation is mitigated by the snowballing stage (Section 4.2.5), which extends the candidate set through the citation neighborhoods of retained studies. Applying snowballing, all eight query-missed gold-set papers were recovered into the final corpus; the protocol-level recall on the gold set is therefore 53/53 = 100%. Together, these results justify the selection of Q4 as the final search query as the fully-merged variant offers no recall gain, and the snowballing stage recovers the studies that no candidate query alone can retrieve.

Executed between 25/02/2025 and 29/07/2025, Q4 returned a total of 2,821 publications across the four databases (Table 4).

Table 3: Candidate query evaluation against the quasi-gold set.

Query	QSM	# retrieved articles
Q1	57%	3,860
Q2	73%	2,304
Q3	57%	1,990
Q4	**85%**	**2,821**
Q5	85%	4,361

Table 4: Search results returned from digital libraries (Q4).

Digital Library	URL	# Articles
ACM Digital Lib.	`dl.acm.org`	816
IEEE Xplore	`ieeexplore.ieee.org`	745
SpringerLink	`link.springer.com`	857
ScienceDirect	`sciencedirect.com`	403
Total		**2,821**

4.2. Study Selection and Filtering

To ensure transparency and reproducibility, we followed a multi-stage filtering process combining automated and manual screening as recommended in Kitchenham et al. (2007); Brereton et al. (2007).

4.2.1. Step 1 - Automated Filtering

Duplicate records and irrelevant publication types were removed. Specifically, we excluded:

- Duplicates across databases
- Entire books
- Entire conference proceedings
- Entries with no title

This stage reduced the number of publications from 2,821 to 2,678 publications (143 removed).

4.2.2. Step 2 - Keyword-based screening

Although the search query enforces semantic relevance at the database level, a complementary keyword-based screening step was required to filter false positives introduced by its necessary generality, particularly acronym ambiguity (e.g., "IaC" referring to *Inertial Axis Constraint* or *International Alcohol Control*) and quality-related papers in unrelated domains. As a result, this step helped improving the overall relevance of the candidate set.

Each paper's title, abstract, and keywords were analyzed for the presence of terms from two concept groups that we present an excerpt of:

- **IaC-related terms:** Infrastructure as Code, IaC, Configuration as Code, Ansible, Terraform, Chef, Puppet, Pulumi, Docker, Kubernetes, CloudFormation
- **Quality-related terms:** quality, bug, defect, fault, smell, anti-pattern, security, vulnerability, misconfiguration, analysis, testing, validation, practice, lint, verification

A publication was retained if at least one term from each group appeared. Due to space reasons, the **complete keyword list and filtering script** are available in the replication package (El Hayani et al., 2026).

The asymmetry between the technology-agnostic search query and the technology-aware keyword list of IaC related terms is intentional and reflects the distinct objectives of the two stages. The database query prioritizes recall and relies only on the foundational terms ("Infrastructure as Code" and "IaC") to capture how the research community collectively frames IaC as a unified discipline. In contrast, the keyword-based screening acts as a precision-enhancing filter over the retrieved corpus. Rather than restricting the corpus, it retains studies using either the foundational IaC terminology or recognized IaC technologies, together with quality-related terms. The inclusion of technology names therefore expands coverage rather than narrowing it.

The IaC technology list was derived from three complementary sources: (i) the most frequently occurring technologies in the retrieved corpus, whose match counts over the 2,678 deduplicated candidates are reported in the replication package; (ii) widely adopted IaC tools reported in independent industry surveys, namely the Stack Overflow Developer Survey 2025[9], the CNCF Annual Survey 2024[10], and Firefly's report about state of IaC 2024[11]; and (iii) the authors' domain expertise regarding emerging or specialized IaC platforms. The list remains non-exclusive by construction, since any study explicitly using foundational IaC terminology was retained regardless of the technology investigated.

To evaluate the empirical impact of this asymmetry, we compared two screening configurations over the 2,678 deduplicated candidates: (1) a technology-agnostic configuration using only foundational IaC terms, and (2) the full configuration combining foundational terms with recognized IaC technologies. The technology-agnostic configuration retained 329 candidates, whereas the full configuration retained 471. The additional 142 studies correspond mainly to papers discussing specific IaC technologies (e.g., Docker or Helm) without explicitly using foundational IaC terminology in their title, abstract, or keywords. Examples include Cito et al. (2017) on the Docker ecosystem on GitHub and a recent study on Helm chart security misconfigurations Minna et al. (2025). The full configuration was therefore retained because it improved recall without introducing observable precision loss.

To validate the filtering reliability, we examined a random sample of 45 excluded studies (~2%), confirming that exclusions fell into three categories: (i) *domain ambiguity*, papers using overlapping terminology in unrelated fields such as civil infrastructure or biological sciences; (ii) *general IT/DevOps context*, studies on CI/CD or cloud platforms that do not explicitly address IaC; and (iii) *functional oriented*, papers mentioning IaC aspects such as containerization purely descriptively without addressing quality concerns. The selected sample and the selection script have been added to the replication package.

This validation confirms that studies were excluded when failing to satisfy both the domain (IaC) and topic (quality assurance) criteria.

4.2.3. Step 3 - Manual Screening (Title & Abstract)

For the remaining 471 publications, we have evaluated them through the title and the abstract. The evaluation was done by applying the inclusion and exclusion criteria based on the research questions' objectives, and the search query presented before. From this we got the following:

- **Inclusion criteria**
  - The study explicitly addresses quality issues in IaC scripts
  - The study presents or evaluates approaches, tools, techniques, or practices for enhancing IaC quality
  - The study is peer-reviewed and written in English
- **Exclusion criteria**
  - Studies that do not focus specifically on IaC (e.g., general DevOps or Cloud)
  - Studies improving cloud deployment or operations not expressed through IaC
  - Non–peer-reviewed material (editorials, short papers (less than 3 pages), posters, theses, technical reports), with the exception of relevant papers or preprints published during 2025.

At this stage, 143 studies were retained for full-text reading

4.2.4. Step 4 - Manual Screening (Full text reading)

Title and abstract are not always enough to identify whether a paper is relevant or not. So we have carefully read the content of the remaining papers, while considering both our research objectives and inclusion/exclusion criteria. As a result of this in-depth review, 79 primary studies were retained (64 were excluded).

[9] https://survey.stackoverflow.co/2025/technology#most-popular-technologies-platform-platform

[10] https://www.cncf.io/wp-content/uploads/2025/04/cncf_annual_survey24_031225a.pdf

[11] https://www.firefly.ai/state-of-iac-report-2024

4.2.5. Step 5 - Snowballing

To complement the automatic database search, we performed one round of backward and forward snowballing following the guidelines of Wohlin (2014). This process was applied to the set of 79 studies retained after full-text screening to identify relevant literature that may have been missed due to terminological variations or database limitations.

Snowballing was deliberately applied at this final stage for two methodological reasons. First, applying snowballing to the full set of retained studies, rather than to intermediate candidate sets, maximizes the signal-to-noise ratio, preventing the inclusion of citations from studies that would eventually be excluded. Second, applying it earlier, for example to the 471 candidates surviving keyword screening, would have required a substantially larger manual effort and amount of time, since each newly identified study must be assessed against the full inclusion and exclusion criteria.

All identified studies were subjected to the same inclusion/exclusion criteria and extraction protocols as the primary corpus. The process ran for three iterations of backward and forward snowballing, yielding 26, 11, and 7 new studies (44 in total). Per-iteration results are provided in the replication package (El Hayani et al. (2026)). Snowballing thus increased the corpus from 79 to 123 primary studies (Figure 1). Two works (S2, S60) were included, despite being non-peer-reviewed yet, since they were recently published (after February 2025), and they directly address our research scope.

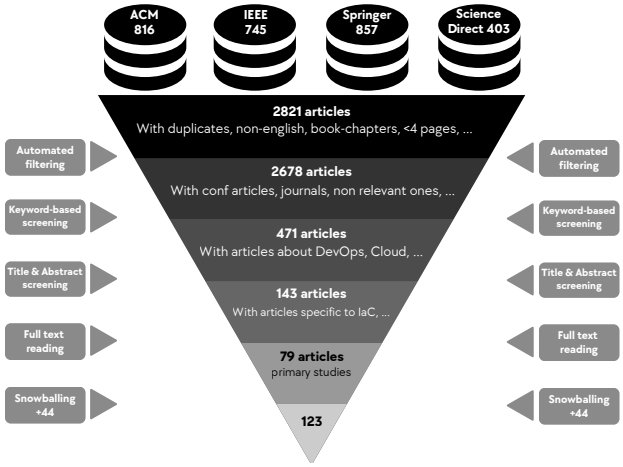


Figure 1: Screening process

4.2.6. Step 6 - Quality Assessment of Primary Studies

Following the selection process, we evaluated the quality of the 123 primary studies. Rather than employing venue prestige as an exclusion criterion, which risks marginalizing emerging research or practitioner contributions, we utilized venue rankings as a descriptive indicator of the corpus's scientific rigor.

Studies were categorized using the CORE ranking for conferences (A*,A,B,C) and SCImago quartiles for journals (Q1–Q4). Venues not indexed by either system (e.g., IEEE SecDev, or national/regional conferences) were classified as unranked. The majority of retained studies originate from A*, A, and Q1 venues (68%), with the full distribution available in the replication package. This characterization is reported to allow readers to situate the corpus within the broader publication landscape and reflect the quality of the considered papers.

4.3. Data Extraction and Synthesis

Following the guidelines of Kitchenham et al. (2007) and Petersen et al. (2015), we developed, collaboratively and iteratively, a data extraction form to systematically collect and organize relevant information. Our data extraction process followed a structured and iterative protocol designed to maximize consistency and minimize subjective interpretation. Prior to any classification, all authors collaboratively defined the dimensions, admissible values, and expected outcomes for each field of the extraction form (Table 5). These definitions were explicitly documented and agreed upon before extraction began, ensuring that all authors shared a common understanding of each concept and reducing the risk of divergent interpretation across raters. This definitional alignment was treated as a living artifact, as it was revisited and refined during weekly meetings throughout the extraction process, particularly when new patterns emerged from the literature.

To validate that this shared understanding was consistently applied, 25 out of 123 studies (20%) were independently coded by at least two authors. Each author filled in the extraction form separately, without consulting the other's answers, and the resulting extractions were then compared dimension by dimension. Disagreements identified during this pair-coding phase were collectively discussed until consensus was reached, and where needed, the agreed definitions were further refined. We computed Cohen's kappa on this validation set, obtaining $\kappa = 0.9$, indicating almost perfect agreement Landis and Koch (1977). This step served both as a reliability check and as a further calibration mechanism, allowing all authors to converge toward a more precise and shared interpretation of the classification scheme, as well as to better refine the initial set of definitions set at the beginning of the process.

The remaining 98 studies were extracted by the first author. Given that the definitional framework had already been established, validated, and refined through both the initial agreement phase and the pair-coding exercise, single-author extraction at this stage represented a controlled and informed process rather than an unchecked one. Uncertain or ambiguous cases encountered during this phase, specifically, entries where a dimension value was absent or underspecified in the paper, or where a study could potentially be mapped to multiple categories, were systematically flagged and brought to the weekly meetings. These meetings served three purposes throughout the process: (i) establishing and documenting the initial shared definitions, (ii) discussing disagreements from the pair-coding phase and refining definitions accordingly, and (iii) collectively validating the first author's extractions and resolving flagged ambiguities through discussion until a consensus was reached.

Table 5: **Articles extraction sheet**

Assoc. RQ	Data dimension	Description	Examples
–	Title	The title of the publication	Title
	Venue name	Name of the publication venue	Conference, journal, ...
	Venue rating	Quality index of the venue	Q1, A*, A ...
	Publication year	Year of publication of the article	2015–2025
RQ1	Quality issue type	Nature of the quality issue	Syntax, security, performance, ...
	Technology dependency	Dependency of quality issues over the different IaC technology	Technology Specific, Technology Agnostic, ...
	IaC Technology	IaC technology(ies) studied in the article	Ansible, Kubernetes, Terraform, ...
	IaC layer	Targeted IaC layer category	Provisioning, Orchestration, ...
	Life-cycle phase	Stage in which the issue arises or is detected	Implementation, Design, Run-time
RQ2	Empirical validation	Details about approaches' evaluation	Tool support, Evaluation context, Reproducibility
	Detection technique	Approach used to detect quality issues	Regex-based, Machine Learning, SMT Solvers, ...
	Code analysis	Nature of the analysis technique	Static, dynamic
	Tool name	Name of the proposed tool	Glitch, Terra-Lint ...
	Code access	Degree of visibility into code during analysis	Black-box, White-box
	Granularity of support	Level at which quality assurance is applied	Script, Module, ...

5. Findings

In this section, we present the findings of our SLR, structured according to the research questions presented before. Before delving into the details of each RQ, Figure 2 represents the distribution of publication years of papers considered in our SLR. The results show an overall increasing research activity over time, reflecting the growing interest of the scientific community in improving the quality of IaC scripts. The notable anomaly observed in 2021 may have been influenced by several factors. Among these, we can mention the COVID-19 pandemic, which disrupted conference schedules globally in 2020 and 2021, leading to cancelled, merged, or scaled-down editions of several software engineering venues. Additionally, this could have impacted the publication cycles and indexing period, introducing significant delays.

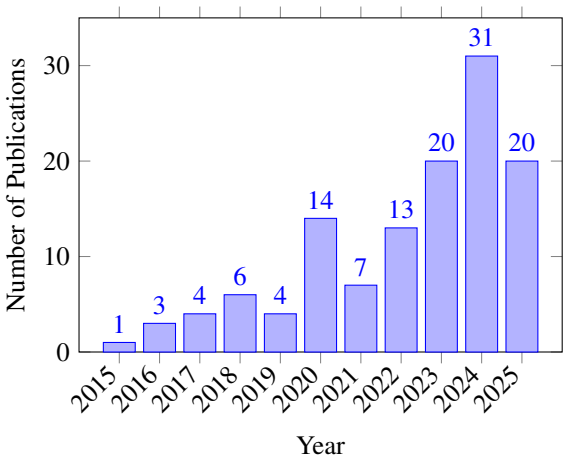


Figure 2: Distribution of studies by publication year.

5.1. *RQ1: Nature of studied quality issues*

To address RQ1, we analyzed the nature and scope of quality issues investigated in prior IaC research. Our objective is to identify which issues have been studied, and characterize how they manifest across different IaC ecosystems, layers, and life-cycle phases. This enables a more comprehensive understanding of the diversity, frequency, and technological dependency of quality issues in IaC development. Specifically, our analysis examined five dimensions, enabling both a vertical (per issue type) and horizontal (per technology and life-cycle phase) understanding:

- **Quality issue categories** to classify and assess which types of issues (e.g., syntax, security, operational) have received the most and least attention in the literature.
- **Technology dependency** to determine to what extent the identified issues are tied to specific IaC technologies or not.
- **Addressed IaC technology** to explore the distribution of quality issues across IaC technologies such as Terraform, Ansible, and Kubernetes, and identify dominant and underrepresented technologies.
- **IaC layer** to map the investigated issues to broader layers of IaC practice, including provisioning, configuration management, orchestration, and image building.
- **Life-cycle phase** to examine when the addressed issues arise within the development life-cycle (e.g., design, implementation, or run-time).

For the remaining of the section, unless otherwise stated, all classifications reported were performed following the data extraction protocol described in Section 4.3.

5.1.1. *Categories of quality issues*

The categorization of IaC quality issues was derived through a collaborative, iterative process involving all authors. Starting from an initial sample of 20 studies, all authors collectively identified the quality issues reported in those studies and grouped them conceptually into an initial set of high-level categories. Each high-level category was internally structured into sub-categories, documented in the replication package, which ensured that mappings were performed at a fine-grained level,

even though results are reported at the high-level category level to preserve readability and focus on the most significant trends.

The categories were revised, merged, or introduced whenever a new quality issue emerging from subsequent studies did not fit the existing structure, or when recurring patterns suggested that existing categories could be better delineated. Mapping studies to categories followed three main steps, first the quality issues addressed by each study were identified and recorded as instances, then mapped to their corresponding sub-category as intermediate step, before being aggregated to the corresponding high-level category.

- **Structural issues** concern the overall architectural design, organization, and modularity of IaC code. It includes abstraction mechanisms, code structuring patterns, and architectural anti-patterns that affect readability, evolvability, and maintainability of infrastructure definitions.
- **Syntactic & Linguistic issues** relate to the correctness and consistent use of language constructs, syntax, and conventions in IaC scripts. They encompass formatting inconsistencies, naming violations, or misuse of technology-specific keywords that can hinder readability and static verification.
- **Security issues** encompass vulnerabilities and weaknesses in IaC configurations that may lead to unauthorized access, privilege escalation, or data leakage. Typical examples include secret exposure, insecure defaults, network misconfigurations, and inadequate authentication or encryption mechanisms.
- **Operational issues** refer to problems that compromise the reliability and reproducibility of infrastructure deployments. They include idempotency violations, dependency ordering errors, state-management flaws, or insufficient error handling that affect the stability of deployed environments.
- **Performance & Resource Management issues** concern inefficient allocation or utilization of infrastructure resources. This category covers over-provisioning, scalability bottlenecks, suboptimal instance sizing, and lifecycle misconfigurations that increase cost or reduce performance.
- **Maintainability & Documentation issues** capture factors that hinder code evolution, collaboration, and long-term sustainability. They involve poor documentation, lack of modularization, weak versioning practices, or hard-coded configuration values that reduce reusability and comprehension.
- **Data & Variable Management issues** address the definition, organization, and handling of variables and parameters within IaC scripts. They include data duplication, inconsistent naming, improper parameter passing, and weak separation between configuration data and control logic.
- **Control Flow & Logic issues** relate to the algorithmic or functional correctness of IaC scripts. They manifest through flawed conditional statements, misused loops, or incorrect sequencing of provisioning and configuration steps, potentially leading to faulty infrastructure states.
- **Domain-Specific issues** capture problems that are specific to certain technologies, cloud providers, or orchestration environments. Examples include container orchestration misconfigurations, provider-specific anti-patterns, and tool-dependent deployment constraints that require contextual expertise to detect and resolve.

Figure 3 presents the distribution of the identified categories across the reviewed studies. Security-related issues dominate the field (57 studies), followed by operational (44), structural (37) and syntactic (27). These results indicate that the research community has largely prioritized securing IaC systems against vulnerabilities, increasing their reliability, and ensuring syntactic and structural correctness. In contrast, control-flow (24), performance (24), and domain-specific (12) issues have received comparatively limited attention. This imbalance, as detailed in Table 6, suggests that aspects such as execution efficiency, domain-awareness, and logical correctness remain under-explored.

Overall, this reflects a defensive research orientation, where the community focuses on preventing infrastructure failures (e.g., reliability or security violations), rather than advancing broader IaC quality aspects such as domain specific awareness, or performance optimization.

Finding 1: Uneven quality research focus

Research on IaC quality is concentrated on security, operational, and structural issues, whereas performance, control-flow, and domain-specific aspects remain comparatively under-investigated.

5.1.2. Technology dependency

To evaluate the extent to which identified quality issues depend on specific IaC technologies, we adopted the classification proposed by Schwarz et al. (2018), which distinguishes quality issues based on their degree of technology dependence. The quality issues are considered:

- **Technology-Specific (TS):** When tightly coupled to one IaC technology.
- **Technology-Dependent (TD):** When partially portable to different IaC technologies after modifying the detection method.
- **Technology-Agnostic (TA):** When reusable across multiple technologies.

As shown in Figure 4, the majority of studies 65% (80 studies) address technology-specific issues, while 21% (26 studies) are technology-dependent, and only 14% technology-agnostic.

Table 6: Distribution of studied quality categories across studies

Quality issue	Studies
Structural	S2, S6, S7, S8, S16, S20, S23, S27, S29, S37, S38, S39, S46, S48, S49, S52, S53, S55, S57, S64, S68, S70, S34, S72, S76, S77, S85, S86, S90, S96, S99, S112, S114, S116, S117, S121, S122
Syntactic & Linguistic	S2, S6, S7, S8, S9, S11, S12, S16, S18, S22, S23, S29, S34, S39, S45, S49, S50, S52, S57, S61, S64, S72, S89, S103, S111, S113, S117, S120, S122
Security	S1, S2, S3, S4, S8, S9, S10, S13, S14, S17, S19, S21, S26, S28, S31, S32, S33, S34, S35, S36, S38, S39, S41, S42, S43, S44, S45, S50, S53, S54, S57, S58, S63, S66, S69, S70, S73, S74, S76, S79, S81, S82, S83, S87, S90, S93, S97, S100, S102, S104, S105, S107, S109, S115, S119, S120, S123
Operational	S2, S3, S5, S8, S9, S11, S12, S14, S21, S22, S25, S26, S27, S28, S29, S34, S38, S40, S46, S50, S52, S56, S57, S58, S59, S65, S67, S68, S69, S75, S78, S79, S81, S85, S94, S95, S98, S101, S106, S108, S110, S113, S117, S118
Performance & Resource Management	S1, S8, S9, S22, S24, S38, S46, S48, S50, S51, S60, S65, S69, S71, S85, S94, S96, S110, S111, S114, S116, S117, S121, S122
Maintainability & Documentation	S2, S8, S9, S12, S14, S15, S21, S28, S29, S38, S39, S49, S52, S56, S57, S58, S64, S65, S86, S90, S92, S93, S103, S117, S121, S122
Data & Variable Management	S2, S3, S11, S12, S21, S22, S28, S31, S39, S47, S49, S50, S52, S57, S58, S64, S69, S70, S72, S80, S84, S88, S106, S111, S112
Control Flow & Logic	S2, S3, S12, S21, S28, S29, S31, S34, S50, S52, S57, S64, S67, S68, S72, S80, S83, S108, S117, S118
Domain-specific	S4, S8, S21, S28, S38, S44, S50, S53, S60, S62, S67, S109

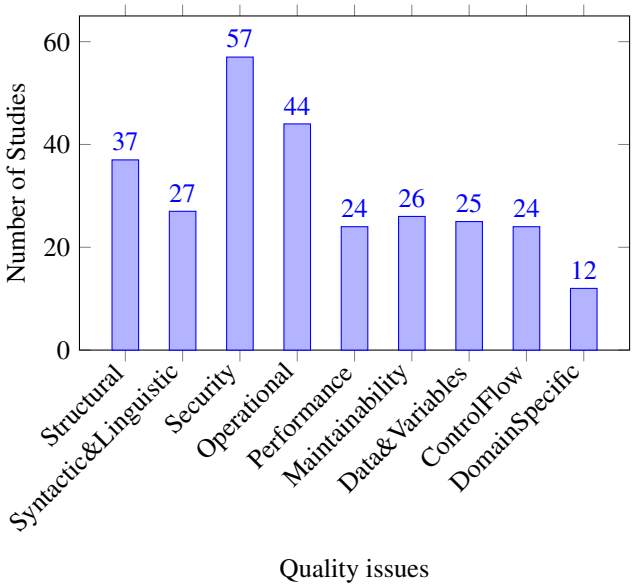


Figure 3: Distribution of studied quality categories

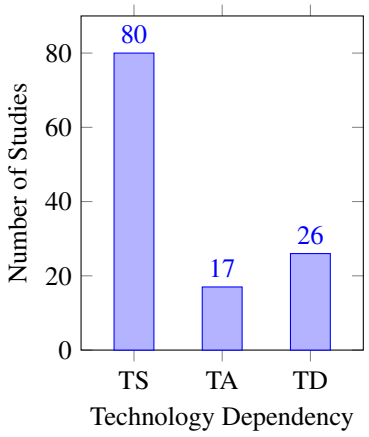


Figure 4: Distribution of technology dependency of quality issues

This imbalance highlights that most investigated quality issues are tied to the implementation details of specific IaC platforms.

Consequently, generalizable quality assessments of IaC independently from the implementation language or platform remain limited. This technology dependency limits interoperability, knowledge transfer, and reproducibility across IaC technologies.

Finding 2: Technology Dependency

Most studies focus on individual IaC tools, resulting in solutions that are technology-specific rather than generalizable or technology-agnostic.

5.1.3. IaC layer coverage

Extending the technology dependency analysis, we examined how identified quality issues are distributed across IaC layers, namely provisioning, configuration management, orchestration, and image building. This analysis gives an abstracted manner of visualizing the distribution of quality issues across the IaC ecosystem. As shown in Figure 5, configuration management is the most frequently studied layer (51 studies). Provisioning follows with 27 studies, image building (27), and orchestration (20). These differences in layer coverage closely align with researchers' technological choices, revealing a strong emphasis on configuration management tools compared to other IaC technologies.

Figure 6 reveals that security is the most frequently addressed quality concern across all IaC layers. Configuration management exhibits the broadest and complete coverage of quality aspects, with studies addressing a wide range of issues beyond security, including syntactic, operational, and variable management. Provisioning follows, characterized by a strong focus on operational and security quality, but limited attention to domain specific issues which were mostly related to cloud providers' specifications. Research on image building is relatively evenly distributed across quality issue categories, with performance and structural issues receiving the most attention. Orchestration receives the narrowest coverage, with studies largely con-

centrated on security.

All things considered, these findings reveal that research attention remains asymmetric across IaC layers, reflecting the technological priorities when studying IaC quality. Additionally, the comprehensiveness of quality issue coverage varies substantially between layers, underscoring the need for more balanced and comprehensive investigations of IaC quality across the entire ecosystem.

Finding 3: Limited IaC layers coverage

Research activity is primarily concentrated on configuration management layer, while orchestration receive comparatively limited attention.

Note: A study was excluded from layer-level counts when it addressed quality issues in a technology-agnostic manner, without explicitly targeting a specific IaC tool or layer. In such cases, assigning the study to any particular layer would have required unwarranted assumptions about its scope, risking artificial inflation of layer-specific counts. In total, 7 studies were excluded.

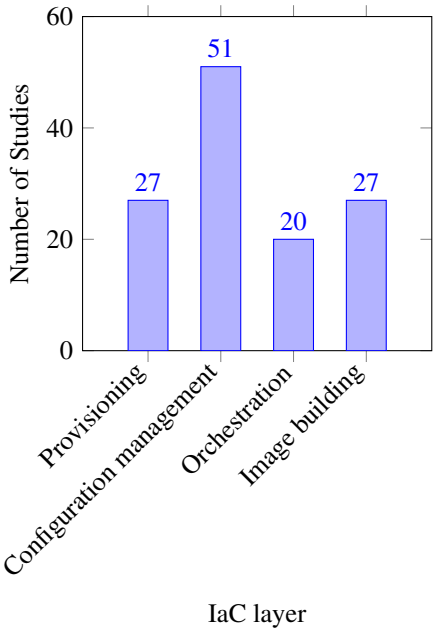


Figure 5: Distribution of studies across IaC layers

5.1.4. *Addressed IaC technologies*

Following the categorization of IaC quality issues, and their distribution over IaC layers, this section examines their distribution in a more detailed level, through evaluating how they occur across different IaC technologies. This analysis will allow us to understand how these quality issues manifest within the ecosystem. As shown in Figure 7, most studies target Ansible (29) and Docker (26), followed by Puppet (21), Terraform (15), and Kubernetes (14), while Docker Compose (2), Helm(3), and

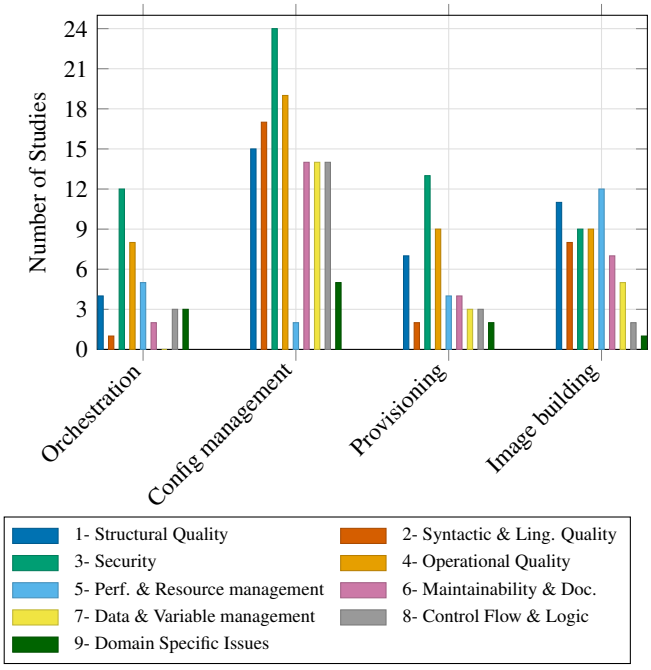


Figure 6: Grouped distribution of quality issues discussed in literature over IaC layers

TOSCA (4) remain barely examined. This uneven distribution reflects the community's technological preferences when studying IaC quality concepts.

Furthermore, Figure 7 highlights how researchers' technology priorities vary across IaC layers. Studies focusing on configuration management predominantly target Ansible, while provisioning-related research most frequently centers on Terraform. For orchestration, Kubernetes is by far the most commonly addressed technology, and Docker emerges as the primary focus for studies investigating image-building and containerization related quality issues.

Moreover, the focus of quality issues differs across technologies. As illustrated in Figure 8, security dominates research on Puppet, Ansible, Kubernetes, Terraform, and Cloud Formation. Whereas Pulumi studies emphasize operational reliability of the code. In contrast, Docker research, gives importance to performance, and structural issues, typically improving container build times, and better structuring Dockerfiles for an optimized image definition. These variations show how the implementation details of IaC concepts influence research priorities and selection of studied quality aspects.

The observed distribution of studies may be influenced by multiple factors, including practitioner adoption, tooling maturity, repository availability, and access to research datasets. However, the present review did not systematically quantify these factors, preventing any causal conclusions regarding their relative influence. Nevertheless, some observations are worth noting. For example, the Stack Overflow Developer Survey 2025[12] reports high adoption rates for Docker (71.1%), Kuber-

[12] https://survey.stackoverflow.co/2025/technology#most-popular-technologies-platform-platform

netes (28.5%), Terraform (17.8%), and Ansible (11.7%), which broadly correspond to the technologies most frequently investigated in the reviewed literature. Similarly, several widely used research datasets, such as PIPr Sokolowski et al. (2024b), TerraDS Buhler et al. (2025), and Andromeda Opdebeeck et al. (2021a), are associated with specific IaC technologies, potentially facilitating research on those ecosystems. At the same time, the existence of datasets for less-represented technologies, such as the Chef security-smell corpus of Rahman (2020), indicates that dataset availability alone cannot fully explain the observed distribution. Researchers have also explored alternative data acquisition strategies, including repository mining and synthetic data generation (e.g., using Large Language Models to generate Ansible playbooks Hassan et al. (2024)).

Overall, the reviewed literature reveals an uneven distribution of research attention across IaC technologies.

Finding 4: Uneven Ecosystem Representation

Research on IaC quality issues is unevenly distributed across tools, leaving large parts of the IaC ecosystem insufficiently examined, resulting in a fragmented and tool-biased understanding of IaC quality.

Note: Technology-level counts should not be interpreted as directly proportional to layer-level distributions. Several studies address multiple IaC technologies simultaneously; such studies are counted once at the layer level but contribute to multiple entries in the technology-level analysis.

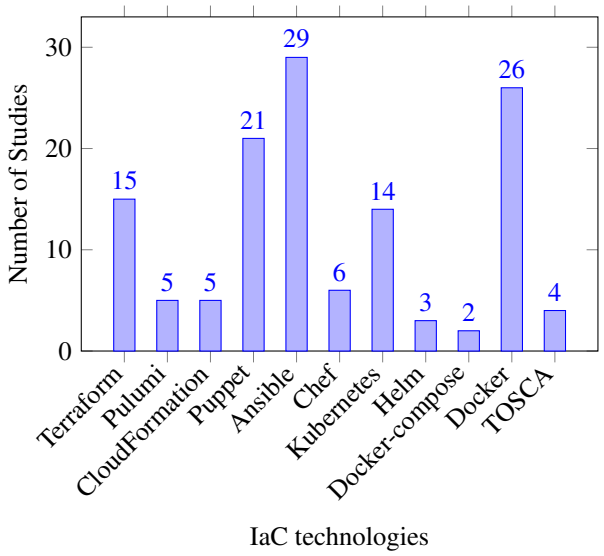


Figure 7: Distribution of studies across IaC technologies.

5.1.5. *Life-cycle phase*

To determine the phases in which quality issues emerge, we adapted and extended the categorization proposed by Sharma et al. (2016). Specifically, we broadened the scope of implementation and design issues and introduced a new category for run-time issues. Accordingly, quality issues are classified as follows:

- **Implementation-time issues:** Occur during the development of IaC scripts, typically involving coding practices, syntax, or logical errors.
- **Design-time issues:** Arise during the planning and structuring of IaC scripts, affecting the architecture of project or module design.
- **Run-time issues:** Occur when IaC scripts are executed , often involving configuration drift, dependency conflicts, or operational failures.

Figure 9 shows that most investigated quality issues (75%) arise during the implementation phase, while design-time and run-time issues remain comparatively under-explored, representing only 10% and 15% of studies, respectively. This imbalance reflects the prevailing research focus on improving IaC scripts during development stage, where issues are easier to detect. In contrast, design- and run-time issues require more complex reasoning about deeper contextual understanding and resource-intensive log analysis, often handled by alternative support solutions, such as monitoring tools. Consequently, the literature shows a research preference of providing proactive solutions, to capture issues in early development phases, and avoid deploying insecure or not reliable infrastructure.

Finding 5: Life-cycle phase imbalance

Existing work largely addresses implementation-time issues, with minimal exploration of design-time and run-time quality aspects.

The five dimensions analyzed in RQ1, quality issue type, technology dependency, IaC layer, IaC technology, and lifecycle phase, collectively define the Quality Issues branch of the feature model presented in Section 6. Each finding in this section corresponds directly to a dimension of that branch, providing the empirical basis for the model's structure.

5.2. *RQ2: Support approaches developed in the literature*

While RQ1 highlights the types of quality issues and their distribution across platforms, RQ2 examines how these issues have been addressed. To characterize the nature of approaches developed for enhancing IaC script quality, we analyzed the following dimensions:

- **Empirical validation:** the extent to which proposed approaches are accompanied by concrete implementations, evaluated under realistic conditions, and supported by replicable research artifacts.
- **Detection technique:** the approach employed to identify quality issues.
- **Code analysis:** whether the approaches operate through static, dynamic, or hybrid analysis.

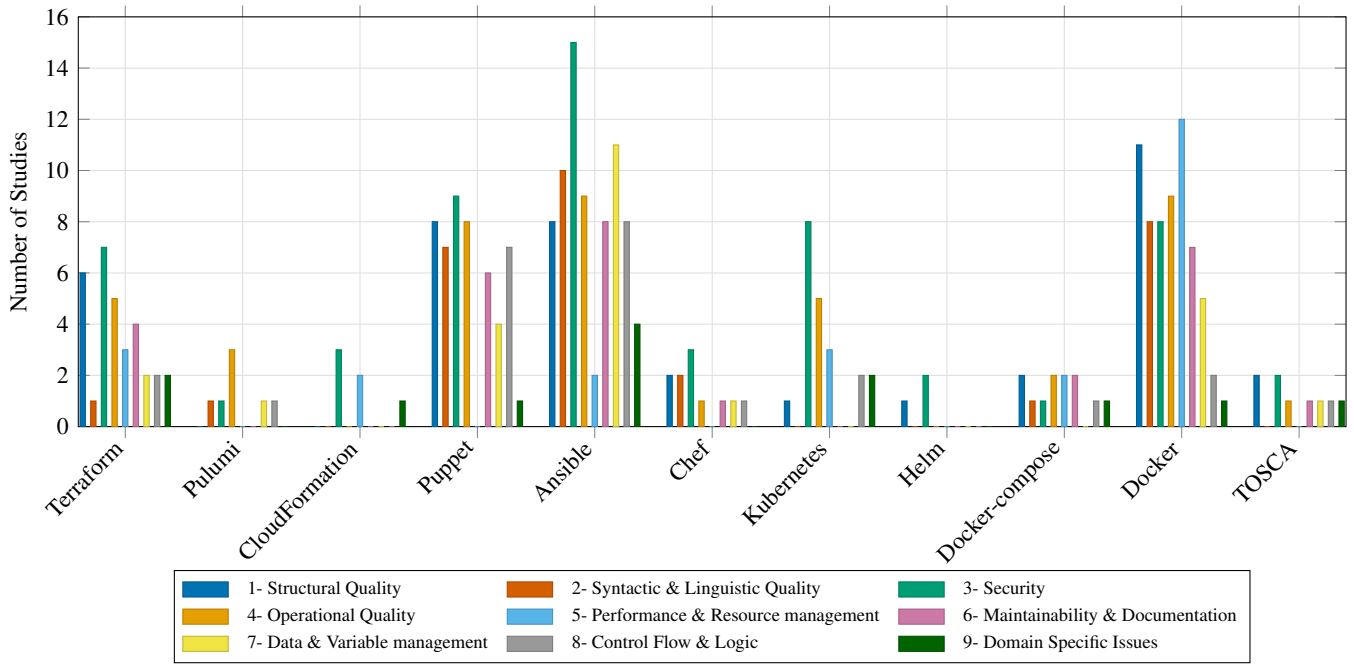


Figure 8: Grouped distribution of quality issues discussed in literature over IaC platforms

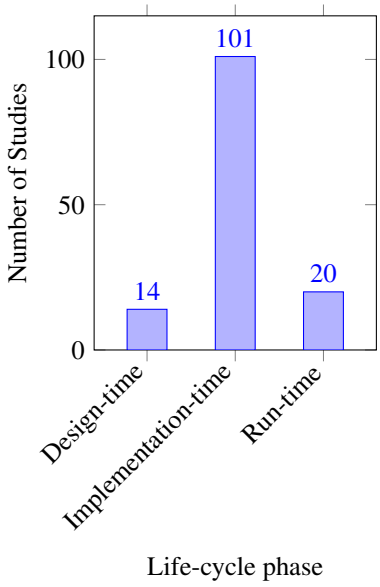


Figure 9: Distribution of quality issues across life-cycle phase

- **Code access:** degree of visibility into code during analysis.
- **Granularity of support:** the scope of focus of developed approach.

5.2.1. *Empirical validation*

Empirical validation is a key indicator of the maturity and practical applicability of IaC quality assurance research. In this section, we analyze three complementary dimensions of empirical validation, whether proposed approaches are accompanied by concrete tool implementations, the context of their evaluation, and the reproducibility of the resulting research artifacts.

1. **Prevalence of tool-based approaches** A first dimension concerns whether the proposed approaches in the literature are supported by an implementation. Our analysis shows that 87% (107) of the studies provide at least a prototype or Proof-of-Concept (PoC) solutions, often released as open-source, whereas only 13% (16) are purely conceptual approaches. The prevalence of implemented solutions suggests that the field is practice-oriented, and the community's orientation to offer practical guidance for improving IaC quality. However, the existence of a tool implementation does not guarantee practical impact. As detailed in the following subsections, most tools remain at the prototype stage and have been evaluated under controlled or open-source conditions, with limited exposure to real-world industrial settings.

Finding 6-a: Tool-centric orientation

Most studies provide prototype tools, reflecting a strong practice-oriented culture in the IaC QA research community.

2. **Context of evaluation** Beyond the existence of tool implementations, we characterize the empirical validation of the 107 tool-based studies along two orthogonal dimensions. The **evaluation environment** captures the setting in which the approach was assessed:
   - **Academic:** The evaluation is designed and executed

by the researchers themselves under controlled or artificially constructed conditions.

- **Industrial:** The evaluation involves practitioners or organizations, structured feedback from developers, or deployment in a real production environment.

The **evaluation corpus** captures the nature of the artifacts used during evaluation:

- **Open-source:** Publicly available IaC scripts, typically mined from platforms such as GitHub or public data-sets.
- **Closed/Internal:** Proprietary code, internal organizational workloads, or researcher-constructed synthetic environments.

As shown in Figure 10, the large majority of tool-based studies, 92 out of 107 (86%), were evaluated in academic or controlled settings. Among these, 80 studies (87%) relied on open-source repositories, while 12 studies (13%) constructed their own experimental environments, typically in early research stage or proof-of-concept work. Industrial evaluations were reported in 15 studies (14%), of which 7 incorporated open-source repositories as part of their evaluation and 8 relied exclusively on closed, internal, or real production workloads.

This distribution reveals a structural validation gap. While open-source evaluation enables reproducibility and large-scale mining, it does not capture the complexities of industrial environments. Therefore, our understanding of how proposed approaches perform under real production conditions remains limited, making in-the-wild validation a priority for the field.

Finding 6-b: Predominance of open-source evaluation with limited industrial validation

Most studies ground their evaluations using open-source code, whereas industrial and practitioner-involved evaluations remain limited.

3. **Reproducibility of experiments** While the majority of studies provide prototype tools, their reproducibility varies substantially. Our analysis shows (Figure 11a) that 71% (87) of the reviewed studies offer some form of replication package. Among these 87 papers, 15 studies (17%) provide experimental data-sets only, 11 studies (13%) release source code only, and 61 studies (70%) supply comprehensive replication packages that include both code and data-sets. In contrast, 36 studies (29%) do not provide any replication artifacts, or reference links that are no longer accessible. In several cases, this lack of availability is attributable to the nature of the experiments, particularly those conducted in industrial settings, or to ethical and security-related constraints.

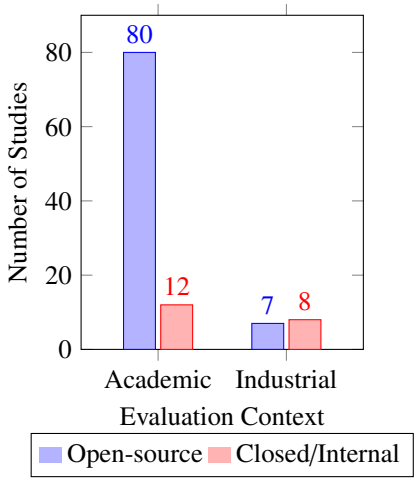


Figure 10: Distribution of evaluation contexts across studies

Figure 11b summarizes the platforms used by the selected studies to host their replication packages. Among studies providing replication artifacts, 49% rely on GitHub and 47% use archival repositories such as Zenodo and Figshare, while Docker Hub and Kaggle are used only marginally. These platforms serve distinct and complementary purposes, archival repositories provide DOIs and immutable snapshots suited for scientific replication, while platforms like GitHub support iterative development and community-driven tool evolution. The choice between them reflects different research objectives rather than different levels of commitment to reproducibility. Ideally, a dual-hosting practice can be adopted, archiving a citable snapshot for replication while maintaining an actively developed version for community reuse, a dual-hosting practice that most studies currently do not follow.

Finding 6-c: High commitment to reproducibility vs. heterogeneous practices

The community demonstrates a strong commitment to reproducibility with high artifact availability, yet there's variability in completeness and archival hosting priorities.

5.2.2. *Techniques and methods*

Given that most studies proposed tool-based solutions (as shown in the previous section), we analyzed the techniques employed to detect and mitigate IaC quality issues. Following a similar approach to our issue categorization, we first extracted and consolidated techniques from an initial sample of 20 papers, grouping those sharing conceptual or operational similarities. Through iterative refinement during the review process, we identified five main families of techniques, plus a residual "other" category for approaches that did not fit into these groups:

- **Rule-Based and Pattern Matching:** Deterministic approaches that rely on predefined rules or syntactic patterns

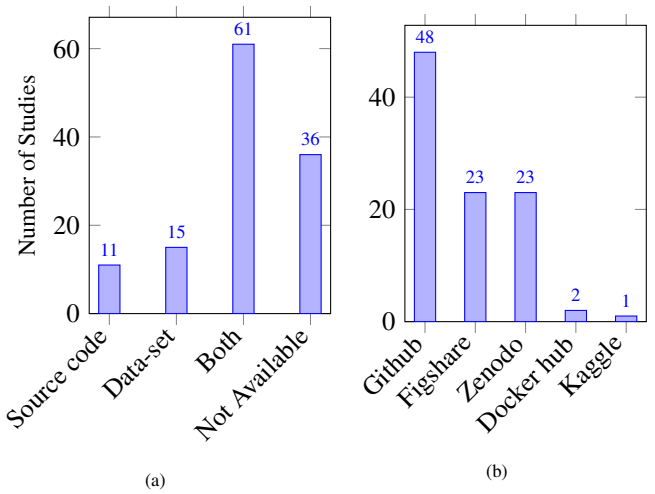


Figure 11: Reproducibility (left) and hosting platforms (right).

to identify known issues. They are transparent and easily interpretable but limited to previously defined patterns (e.g., regular expressions, rule-based checks).

- **Graph-Based analyses:** Approaches that construct intermediate representations such as Abstract Syntax Trees (ASTs), Program Dependency Graphs (PDGs), or Resource Graphs (RGs) to reason about dependencies and execution flow. These techniques provide deeper semantic insights than purely text-based matching.
- **Formal Methods & Verification:** Logic-driven approaches that encode IaC configurations as constraint or satisfiability problems to verify properties (e.g. SMT solvers)
- **Statistical & Learning-Based:** Data-driven approaches that rely on statistical inference, machine learning, or large language models (LLMs) to detect anomalies or predict defects from historical data.
- **Testing and Execution-Based** Techniques that validate IaC scripts by executing them (in real or simulated environments) or generating test cases to verify expected behaviors. This includes defining tests, using fuzzing techniques, or log analysis.

Figure 12 shows that rule-based and pattern-matching (49 studies) and graph-based analysis (49 studies) dominate current research, presenting more than 50% of the methods used, revealing a clear research preference for deterministic and interpretable techniques, often combining syntactic rules with structural reasoning for higher accuracy and analysis of quality issues. These methods excel at syntactic and structural detection, allowing for good syntactic and semantic checks. Additionally, learning-based approaches (36 studies) represent a growing interest toward adaptive, probabilistic techniques that leverage historical data and large-scale repositories. This reflects the community's increasing interest in AI-assisted IaC analysis using techniques such as machine learning (ML), deep learning (DL), and more recently, Large Language Models (LLMs) and Retrieval Augmented Generation (RAG). Formal verification (10 studies) and testing and execution-based techniques (20 studies) remain marginal. Their limited adoption likely reflects the difficulty of formally modeling heterogeneous, dynamic IaC environments and the high computational cost of executing or simulating IaC deployments. Table 7 summarizes the distribution of techniques employed across the selected tool-based studies.

Consequently, the current research remains biased toward more deterministic approaches, with an increasing interest in probabilistic approaches.

Finding 7: Methodological bias

IaC support approaches predominantly rely on deterministic techniques (e.g., rule-based, pattern-matching, and graph-based analysis), while formal verification, and testing and execution-based methods remain marginal.

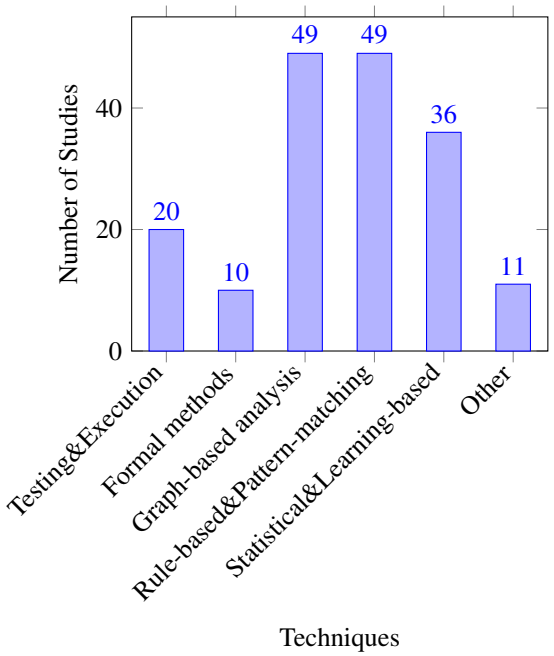


Figure 12: Distribution of techniques

5.2.3. *Code analysis*

After presenting the different categories of techniques used in the literature, we further analyzed the code analysis nature of the proposed approaches, classifying them as follows:

- **Static analysis:** which examines the syntactic or semantic properties of IaC scripts without executing them Chiari et al. (2022).
- **Dynamic analysis:** which evaluates IaC through execution, testing, or run-time monitoring Chiari et al. (2022).

Table 7: Distribution of studied techniques across studies

Technique	Studies
Rule-based & Pattern-matching	S2, S4, S8, S14, S15, S16, S19, S20, S23, S24, S28, S29, S33, S35, S37, S39, S43, S44, S45, S46, S54, S55, S60, S66, S67, S69, S72, S73, S77, S80, S81, S83, S85, S86, S88, S90, S98, S100, S101, S104, S105, S106, S112, S114, S115, S116, S119, S120, S122
Graph-based analysis	S4, S14, S16, S17, S19, S20, S24, S26, S29, S30, S33, S35, S37, S40, S47, S51, S52, S53, S62, S65, S66, S67, S69, S72, S74, S75, S76, S77, S78, S81, S84, S85, S86, S89, S92, S95, S99, S100, S101, S104, S106, S107, S109, S112, S114, S115, S116, S119, S122
Formal methods & verification	S31, S40, S51, S56, S62, S75, S91, S99, S108, S118
Statistical & Learning-based	S2, S6, S12, S15, S18, S21, S25, S27, S29, S36, S41, S43, S49, S53, S55, S61, S62, S64, S67, S68, S73, S78, S79, S80, S84, S85, S88, S89, S90, S92, S94, S96, S97, S98, S103, S109
Testing & Execution-Based	S22, S26, S33, S48, S50, S59, S75, S78, S81, S91, S95, S96, S97, S98, S101, S102, S110, S115, S119, S123
Other	S1, S7, S10, S11, S13, S32, S58, S63, S76, S82, S87

- **Hybrid analysis:** which combines static inspection with dynamic validation.

As illustrated in Figure 13, static analysis dominates the literature (73% of the studies), while dynamic and hybrid approaches are marginally used. Although static analysis facilitates early detection of issues, since it doesn't need executing the code, it cannot capture dynamic or environment-dependent behaviors, revealing a persistent limitation in current research. This limitation is perfectly aligned with the findings of the previous section, where Testing & Execution techniques were one of the least used techniques, representing less than 12%. Overall, these findings highlight the need for broader exploration of dynamic and hybrid analyses to achieve more comprehensive IaC quality assurance.

Finding 8: Static analysis dominance

Static analysis of IaC is predominant, with dynamic and hybrid approaches remaining scarce, limiting the field's ability to detect dynamic behavior of IaC.

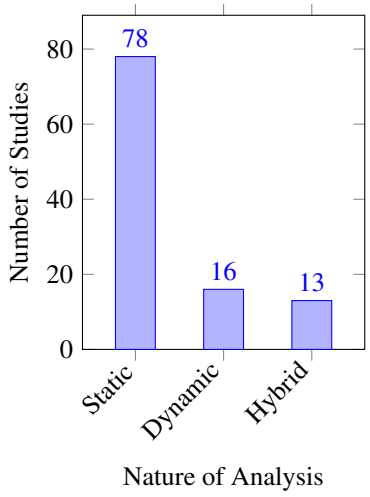


Figure 13: Distribution of IaC quality analysis approaches

5.2.4. *Code access level*

IaC scripts can be analyzed at two access levels, **white-box**, where full code access is required, and **black-box**, where the code is treated as opaque. Our results show that white-box analysis dominates current research, reflecting a strong preference for direct code analysis.

The works of Choi et al. (2024), Hanappi et al. (2016), and Mishra et al. (2022) represent the only three studies, out of 123, in our corpus that adopt a black-box approach. Choi et al. (2024) identify Misconfigured Container Components (MCCs) through external inspection techniques including port scanning, server-signature analysis, and request-response observation, without accessing the underlying Kubernetes manifests. Similarly, Hanappi et al. (2016) evaluate the reliable convergence and idempotence of Puppet configuration scripts by treating the individual resource actions as completely opaque. Likewise, Mishra et al. (2022) observe running containers entirely from the outside, capturing malicious container behaviour. Rather than analyzing the semantic logic within the code, these approaches rely exclusively on external deployment behavior and runtime artifacts; they dynamically track state changes by intercepting system calls and observing transient network and process states.

Despite not operating directly on the internal logic of the IaC code, the contributions of all studies result in actionable manifest-level improvements: Choi et al. (2024) provide recommendations on insecure defaults, software versioning, and network access controls, while Hanappi et al. (2016) identify specific resource conflicts and missing satisfaction checks within the deployment, and Mishra et al. (2022) allow detecting unauthorized container operations, requiring an immediate permission changes.

We emphasize that the *code-access*, and *analysis-type* dimensions are orthogonal and capture different aspects of QA approaches. The former refers to the level of visibility and access to the IaC artifacts (e.g., direct inspection of source code or configurations), whereas the latter characterizes whether the analysis is performed without execution (static) or relies on runtime information (dynamic). These dimensions should not be conflated, as approaches that leverage execution traces or run-

time artifacts (e.g., system calls, logs, or deployment behavior) have a *dynamic* aspect, but they can be classified either as white-box if they require access to IaC scripts, besides the runtime metrics, or black-box if not.

Overall, the evidence shows that current IaC support approaches heavily rely on white-box reasoning, with limited exploration of environment or deployment oriented analyses that could capture quality issues not detectable through code analysis alone.

Finding 9: Code access

IaC code analysis relies heavily on direct, full code access analysis, rather than environment inspection.

5.2.5. *Granularity of Detection*

We examined the granularity at which IaC quality analyses are conducted, distinguishing three levels of focus:

- **Script level :** analysis confined to individual IaC scripts.
- **Module level :** considers interactions among multiple scripts within a module.
- **Project level :** captures dependencies and consistency across modules and the entire project.

As shown in Figure 14, script-level analyses dominate the literature, with 91 studies (72%), while module-level (10 studies) and project-level (25 studies) investigations remain comparatively limited. This reveals a strong bias toward fine-grained, localized quality issue detection, likely driven by the relative simplicity and automation potential of script-level checks. The predominance of fine-grained, script-level analyses indicates that current solutions often treat IaC files as isolated blocks, overlooking the complexity of real infrastructure systems where multiple configuration files reside in one project. Consequently, this restricts the ability to identify cross-cutting quality issues, such as inter-script dependency conflicts, module interaction failures, or project-wide configuration inconsistencies. Overall, current research provides a narrow view of IaC quality, centered on individual files rather than the broader configuration ecosystem in which they operate.

Finding 10: Granularity focus

Analyses are often confined to the script level, missing system interactions between the different infrastructure scripts that influence IaC quality in complex deployments.

The five dimensions analyzed in RQ2, tool support, detection technique, code analysis type, code access level, and granularity, collectively define the Support Approaches branch of the feature model presented in Section 6. Together with the RQ1 dimensions, they form a complete characterization of the IaC QA landscape that the taxonomy synthesizes.

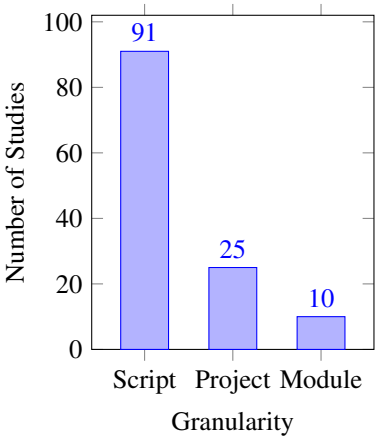


Figure 14: Granularity levels of IaC analyses

6. Taxonomy: A Feature Model for IaC Quality Assurance

The feature model presented in Figure 15a is a direct synthesis of the findings reported in RQ1 and RQ2. It serves as the integrating artifact that organizes the ten dimensions analyzed across both research questions into a unified, navigable structure. Its purpose is twofold: to make the findings actionable for researchers positioning new work, and to provide practitioners with a structured lens for auditing their QA toolchains against the full landscape of known quality dimensions.

Quality issues are described through the following dimensions:

- **Type:** Defined by the categorization proposed in Section 5.1.1. A support approach may address one or multiple quality issue types.
- **Technology dependency:** The degree to which the issue is specific to, dependent on, or agnostic of a given IaC technology.
- **IaC platform:** The particular technology or tool targeted (e.g., Ansible, Terraform, Puppet).
- **IaC layer:** The addressed IaC layer (Provisioning, Configuration management, Orchestration, or Image building).
- **Life-cycle phase:** The stage of the IaC development where the issue manifests (design, implementation, or runtime).

Support approaches are characterized according to the following dimensions:

- **Tool Based:** Whether the contribution provides an implemented tool or conceptual guidance
- **Techniques:** The technique employed, as defined in our categorization in Section 5.2.2.
- **Code analysis:** The type of analysis, static, dynamic, or hybrid, through which quality issues are identified.

- **Granularity:** The level of abstraction targeted, ranging from individual scripts to full project-level analysis.
- **Code access:** The analysis paradigm adopted, white-box approaches requiring full code access versus black-box methods emphasizing contextual or behavioral insights.

6.1. Implications for researchers

To illustrate how the feature model can be used to position existing studies within the IaC QA landscape, we randomly instantiate it with the work of Rahman et al., who developed ACID, a tool for detecting defects in Puppet scripts Rahman et al. (2020a). Using our model, their contribution can be characterized along the following quality issue dimensions:

- **Type of issues studied:** Security, Operational, Control Flow & Logic, Data and Variable Management, Maintainability and Documentation, and Domain-Specific issues.
- **Technology dependency:** The study is technology-specific, as it exclusively targets Puppet.
- **IaC technology:** Puppet.
- **IaC layer:** Configuration Management.
- **Life-cycle phase:** The proposed support is provided primarily at implementation time.

From the perspective of support approaches, the study can be described as follows:

- **Tool-based:** A concrete tool implementation (ACID) is provided.
- **Techniques:** The approach relies mainly on rule-based and pattern-matching techniques.
- **Type of analysis:** Static analysis.
- **Granularity:** Script-level analysis.
- **Code access:** White-box access to the Puppet codebase.

By mapping ACID onto these dimensions, the study becomes clearly positioned within the broader literature (Figure 15b), making its scope, and contributions explicit. This structured positioning also facilitates direct comparison with other IaC QA tools and approaches. More importantly, the feature model highlights where opportunities for extension or differentiation lie. For instance, a future study inspired by ACID could:

- Retain similar support techniques but target a different or under-studied IaC technology (e.g., Ansible or Terraform),
- Broaden ACID's detection mechanisms to be multi-technology or technology-agnostic,
- Extend ACID's coverage to run-time Puppet issues, thereby addressing a different life-cycle phase,
- Combine rule-based detection with complementary techniques such as statistical or learning-based analysis,
- Or expand the granularity from script-level analysis to module-level or project-level reasoning.

In each case, the feature model provides a clear vocabulary for articulating how a new contribution diverges from, builds upon, or complements prior work. This makes it a practical reference for researchers seeking to identify unaddressed research space, and to position their contributions compared to prior works.

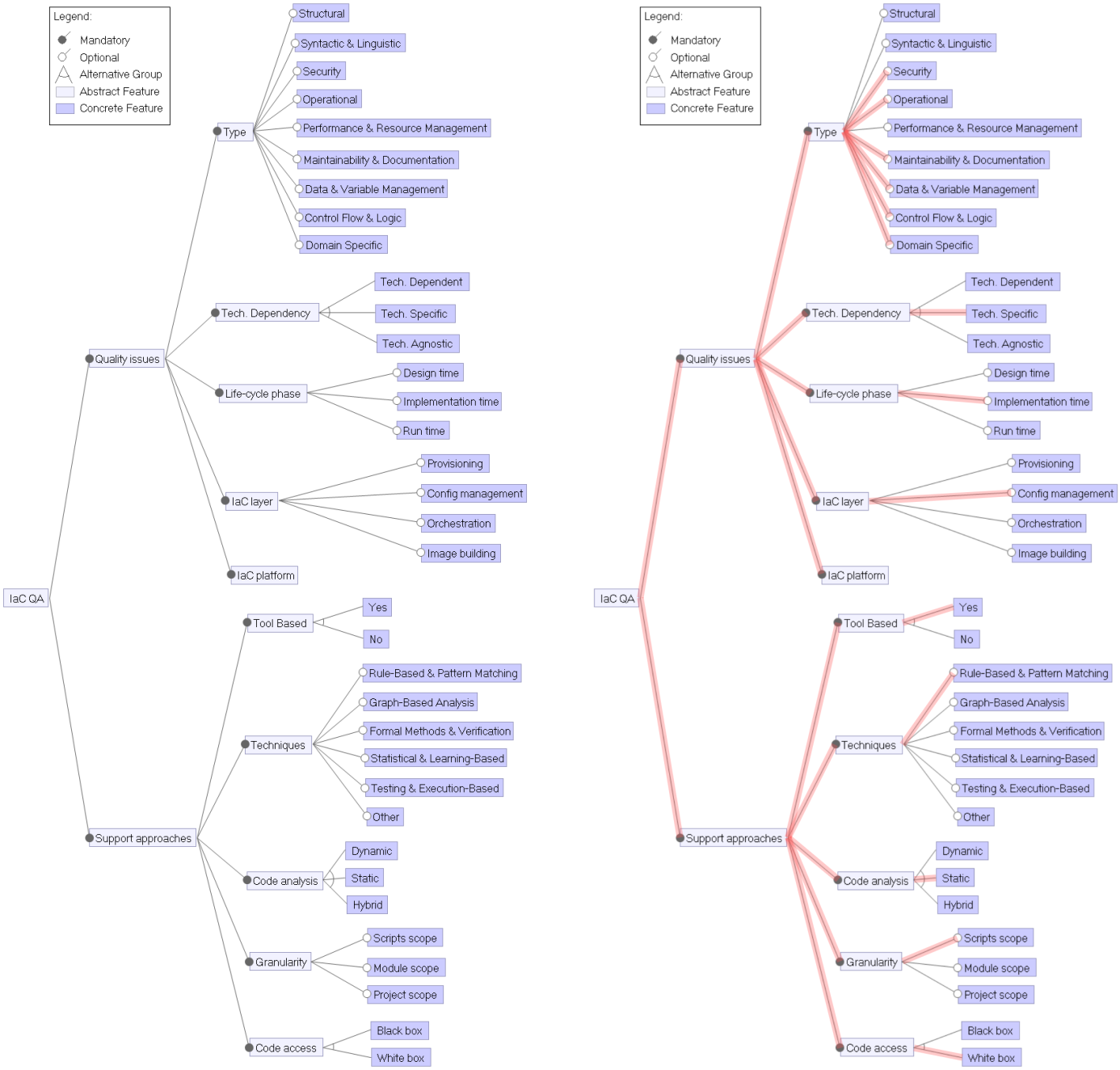
6.2. Implications for practitioners

The proposed feature model supports practitioners in systematically addressing real-world challenges in IaC quality assurance. By instantiating their organizational context within the model, teams can derive a tailored, lifecycle-aware QA strategy aligned with the complexity of modern cloud-native infrastructures. In particular, the model enables the following practitioner-facing use cases:

- **Detection of QA blind spots:** Our analysis reveals a significant research and tool bias toward static analysis (Finding 8) and implementation-time issues (Finding 5). Practitioners can use the Life-cycle phase and Code analysis dimensions of the feature model to audit their existing toolchains. If a team's current QA suite is limited to static, script-level checks, the model serves as a strategic roadmap to identify missing dynamic or run-time checks. This facilitates a transition toward a more comprehensive QA strategy capable of detecting issues like configuration drift, idempotency violations, and environment-dependent failures.
- **Informed tool selection and adoption:** The feature model helps practitioners systematically assess whether existing IaC quality assurance tools adequately cover the quality dimensions most relevant to their context (e.g., security, reliability, or performance). For instance, by defining the target IaC Layer (e.g., Orchestration), IaC technology (e.g., Kubernetes), and the desired Quality Issue Type (e.g., Security), teams can benchmark candidate tools against the Techniques, and the Code analysis dimensions. This allows practitioners to better identify gaps in their current toolchains before these gaps manifest as operational failures or outages.

To illustrate concretely how these use cases could manifest in real world, consider a DevOps team operating a typical cloud-native stack: Terraform for cloud provisioning, Kubernetes for container orchestration, and Docker for image building. The team currently integrates static linters in their CI pipeline, like tflint [13] for Terraform, kube-linter [14] for Kubernetes manifests, and hadolint [15] for Dockerfiles. . They wish to audit whether

[13] `https://github.com/terraform-linters/tflint`
[14] `https://docs.kubelinter.io/#/`
[15] `https://github.com/hadolint/hadolint`



(a) Taxonomy's Feature Model

(b) Rahman et al. (2020a) work's projection on the feature mode

Figure 15: Feature Model Representation: (a) Proposed Taxonomy and (b) Projection of Rahman et al.'s Work Rahman et al. (2020a)

this toolchain provides adequate quality assurance. Instantiating the feature model with the team's context may produce the coverage Table 8.

From this audit, three concrete actions are possible. First, the team can complement the current static linters with graph-based analysis at the project level, for example, by integrating an open-source tool such as GLITCH (S39) for cross-layer analysis. Second, the static linters do not detect state reconciliation defects that manifest only at run-time (Hassan et al. (2024), S50); the team can address this with smoke tests (Cannavacciuolo and Mariani (2022b), S48) or runtime checkes after deployment. Third, none of the current tools target performance or cost anti-patterns (Bolhuis et al. (2025), S60; Durieux (2024), S24); the team can incorporate performance or cost-aware policies into review gates. This showcases the principal use of the feature model for practitioners, by making blind spots explicit. The same instantiation procedure applies to any other technology stack, each dimension is checked against the team's existing tooling.

This feature model offers both researchers and practitioners a structured lens through which to explore the current landscape, and to identify under-studied aspects to prioritize for future re-

Table 8: Coverage table for IaC toolchain QA audit

Dimension	Current coverage	Gaps revealed by the model
Quality issue type	Security, syntactic	Operational, performance, control flow not covered
IaC layer	Pr, Or, IB	CM (if Ansible/Puppet is later added, gap reappears)
Life-cycle phase	Implementation-time only	Design-time and run-time gaps (Finding 5)
Code analysis	Static only	No dynamic or hybrid analysis (Finding 8)
Granularity	Script-level	No module- or project-level checks (Finding 10)
Techniques	Rule-based & pattern-matching	Maybe considering graph-based or learning-based for advanced detection

search directions. Moreover, it serves as a practical reference for those developing new tools to assist DevOps engineers and IaC developers in enhancing script quality, helping them identify which features remain insufficiently addressed and how their solutions can go beyond existing approaches.

7. Research agenda

The findings reveal that current research on IaC offers a dual fragmentation in what is studied (quality issues) and how it is addressed (support approaches). Existing studies tend to be tool-specific and lack a holistic, technology-independent understanding of IaC quality. Similarly, support approaches are predominantly static, rule- and graph-based, and white-box oriented, leaving significant gaps in dynamic, environment-aware, and comprehensive quality assessments. These observations motivate future research directions aimed at advancing both the limitations in quality assessment and the evolution of support techniques.

7.1. Advancing the Study of IaC Quality Issues

Three directions emerge from RQ1 for advancing the study of IaC quality issues: the breadth of quality dimensions currently in scope (Section 7.1.1), the technology-independence of the issues studied (Section 7.1.2), and the life-cycle phase at which they are addressed (Section 7.1.3). Each is grounded in a specific finding from Section 5.1 and points to concrete extensions of existing primary studies.

7.1.1. ***Broadening the Landscape of IaC Quality Dimensions***

Our review highlights several research gaps that open promising research directions for advancing the understanding of IaC quality. A substantial opportunity lies in addressing under-explored quality dimensions, particularly *performance*, and *domain-specific* issues. Current research predominantly ensures that infrastructure "works" and is "secure," but future investigations should extend this scope toward efficiency, scalability, maintainability, and contextual correctness (Finding 1).

A particularly promising direction concerns sustainability and cost-awareness, where early work has begun to emerge. Kosbar and Hamdaqa (2025) (S46) introduced a catalog of sustainability smells in IaC, while Bolhuis et al. (2025) (S60) identified cost anti-patterns specific to Terraform and CloudFormation. Durieux (2024) (S24) empirically studied the impact of Docker smells on image size, providing initial evidence that structural quality choices have measurable resource consequences. Qiao et al. (2025) (S51) took a step further by statically inferring resource usage bounds for IaC programs, demonstrating that performance-related properties can, under certain conditions, be approximated without full execution.

These studies collectively represent early but isolated efforts toward performance-aware IaC quality assurance. Future research work should build on these foundations to develop approaches to performance and resource quality that reflect the constraints and workload patterns of real cloud environments.

For practitioners, the sustainability smells of Kosbar and Hamdaqa (2025), and the cost anti-patterns of Bolhuis et al. (2025) can be encoded as policy-as-code rules in tools such as Open Policy Agent [16] or Checkov [17] , and the Docker image-size impact study of Durieux (2024) provides a concrete checklist for image hygiene.

7.1.2. ***Towards Tool-Agnostic, Cross-Platform IaC Quality Foundations***

Finding 2 reveals that 65% of studies address technology-specific quality issues, resulting in solutions that are tightly coupled to individual IaC technologies and difficult to generalize. This fragmentation limits knowledge transfer across tools and prevents the emergence of a unified IaC quality vocabulary.

Several recent studies have begun to address this challenge. Saavedra et al. (2023) (S39) developed GLITCH, a polyglot smell detection framework capable of identifying quality issues across multiple IaC technologies through a unified intermediate representation. Building on this, Saavedra et al. (2025) (S31) introduced InfraFix, a technology-agnostic repair framework that applies fixes across different IaC platforms without requiring tool-specific adaptations. At the metrics level, Dalla Palma et al. (2020b) (S7) proposed a catalog of software quality metrics applicable to infrastructure code broadly, while Konala et al. (2025a) (S2) developed a framework for measuring IaC quality in a tool-independent manner. These contributions represent meaningful first steps toward platform-independent IaC quality foundations, but they remain limited in scope and coverage.

Future research should build on these efforts to formalize tool-agnostic IaC quality constructs, develop cross-platform quality metrics, and establish reusable assessment frameworks that operate across the full IaC ecosystem. Supporting this

[16] `https://www.openpolicyagent.org/`
[17] `https://www.checkov.io/`

generalization requires investment in comprehensive, cross-platform datasets that go beyond the tool-specific corpora currently available, such as PIPr Sokolowski et al. (2024b), TerraDS Buhler et al. (2025), and Andromeda Opdebeeck et al. (2021a), to better reflect the heterogeneity of real-world IaC projects combining multiple tools and layers.

Practitioners managing multi-technology stacks can mitigate operational failures due to the fragmented, and technology-specific visibility of quality issues, by adopting technology-agnostic frameworks such as GLITCH Saavedra et al. (2023) for a unified smell reporting, and by normalizing tool outputs into a shared internal vocabulary (a small mapping layer between technology-specific linters like tflint, kube-linter, hadolint, and ansible-lint [18] pays back quickly in cross-team comprehension).

7.1.3. *Extending IaC Quality Assurance Across the Entire Lifecycle*

Finding 5 shows that 75% of investigated quality issues arise at implementation time, with design-time and run-time issues accounting for only 10% and 15% of studies respectively. This imbalance reflects the prevailing research preference for proactive, early-stage detection, but leaves critical lifecycle phases largely unaddressed. Some studies have begun to explore quality issues beyond the implementation phase. At the design level, Shambaugh et al. (2016) (S40) proposed a configuration verification tool for Puppet that reasons about the intended behavior of scripts before deployment, while Weiss et al. (2017) (S56) introduced Tortoise an interactive repair tool that combines static analysis with runtime system call observation to fix configuration errors. At the run-time level, Hassan et al. (2024) (S50) investigated state reconciliation defects that manifest only during execution, and Xu et al. (2024) (S5) conducted an empirical study of Kubernetes operator bugs that surface in live cluster environments. Sokolowski (2022) (S30) explored IaC for dynamic deployments, where infrastructure configurations must adapt to runtime conditions. Together, these studies demonstrate that lifecycle-aware IaC quality assurance is feasible, but remains limited.

Future research should develop reactive, context-aware quality mechanisms capable of spanning the full IaC lifecycle, from architectural design decisions through operational deployment and runtime monitoring. This includes exploring how quality issues evolve across lifecycle phases, and how early design choices propagate into operational failures.

Following this direction, practitioners can close the design-time gap with lightweight architecture-review checks before code is written, and the run-time gap with deployment-time smoke tests or operator monitoring that compares declared state against observed state. Both interventions can be added incrementally to an existing pipeline making it more aware of the IaC context.

[18] `https://docs.ansible.com/projects/lint/`

7.2. *Advancing IaC Support Approaches*

While Section 7.1 focused on the quality issues aspect, this section is about how they are detected, validated, and scoped. Three directions follow from RQ2: stronger collaboration with industrial partners to validate proposed approaches under realistic conditions (Section 7.2.1), broader coverage of dynamic and hybrid analyses beyond the prevailing static focus (Section 7.2.2), and a shift from script-level to system-level scoping that reflects the interdependent nature of real infrastructure projects (Section 7.2.3). Each direction builds on a small set of pioneering studies already present in our corpus.

7.2.1. *Strengthening Real-World Validation of IaC Support Approaches*

Our analysis of empirical validation practices (Section 5.2.1) reveals only 14% of the tool-based studies involve industrial or practitioner-engaged evaluation (Finding 6-b). This gap is particularly motivating further collaboration between academic researchers and industrial experts.

A small number of studies have demonstrated the value and feasibility of practitioner-involved evaluation. Reis et al. (2023) (S35) directly engaged practitioners to collect structured feedback on a security linter, using their responses to iteratively refine the tool's detection rules and reduce false positives. Hu et al. (2023) (S14) reported an experience from practitioners at AWS characterizing static analysis alerts for Terraform manifests, providing rare insight into how academic tools perform under industrial conditions. Sahoo et al. (2024) (S6) presented Ansible Lightspeed, a code generation service developed and deployed in an industrial context at IBM and Red Hat, representing one of the few examples of a fully industrially-validated IaC quality support tool. Verdet et al. (2025) (S10) assessed the actual adoption of security policies by developers across different cloud providers, grounding their findings in observed practitioner behavior rather than repository mining alone.

These studies illustrate that practitioner involvement is both achievable and methodologically valuable. Future research should strengthen collaboration with DevOps teams, validate tools in production-like environments, and treat practitioner feedback as an evaluation criterion. A particularly impactful contribution would be datasets pairing IaC scripts with operational artifacts, like runtime logs, deployment histories, and configuration drift records, enabling a new class of IaC QA research capable of reasoning about quality issues not only in scripts, but in the operational context in which they are executed.

7.2.2. *Extending Beyond Static Analysis*

Finding 7 & 8 show that static, rule- and graph-based analyse predominate the support approaches reviewed, while dynamic and hybrid approaches remain rare. Although static analysis enables early detection without code execution, it is fundamentally limited in its ability to capture runtime behaviors, environment-dependent failures, and performance-related issues that only manifest during deployment.

A growing body of work has begun to explore dynamic and testing-based approaches for IaC quality assurance.

Sokolowski et al. (2024a) (S11) developed an automated testing framework for IaC programs that generates and executes test cases to verify infrastructure behavior. Spielmann et al. (2023) (S26) proposed an extensible testing infrastructure that supports the definition and execution of IaC-specific test suites. Sokolowski and Salvaneschi (2023) (S59) explored foundational principles for reliable IaC through a testing-oriented lens. Cannavacciuolo and Mariani (2022b) (S48) introduced smoke testing for cloud systems as a lightweight dynamic validation strategy. Reis and Correia (2022) (S22) developed Dockerlive, a live development environment for Dockerfiles that provides immediate feedback during script editing by partially executing the build process.

Despite these contributions, dynamic and testing-based approaches remain marginal (less than 10% of the considered corpus) and are rarely combined with static techniques into hybrid pipelines. Future research should invest in IaC-specific testing methodologies, infrastructure mocking and simulation frameworks, fuzzing techniques adapted to declarative IaC semantics, and runtime telemetry integration. Hybrid approaches that combine the efficiency of static analysis with the behavioral coverage of dynamic techniques represent a particularly promising direction for comprehensive IaC quality assurance.

DevOps teams already perform dynamic validation, through integration tests, canary deployments, and chaos engineering, but typically outside the IaC quality discussion. Discovering methods for connecting their outputs back to the originating IaC scripts would strengthen the dynamic validation of IaC scripts.

7.2.3. ***From Script-Level to System-Level Analysis***

Finding 10 shows that most of studies operate at the script level, treating individual IaC files as isolated analysis units. This granularity is insufficient for real-world infrastructure systems, where quality issues often arise from interactions across scripts, modules, roles, and services rather than from defects within a single file.

Several studies have begun to move beyond script-level analysis. Ntentos et al. (2023) (S20) investigated coupling-related architecture smells in microservice deployments described through IaC, reasoning about inter-service dependencies across multiple configuration files. Qiu et al. (2024) (S62) developed an approach for unearthing semantic checks in cloud IaC programs that reasons about cross-resource dependencies and consistency constraints at the project level. Opdebeeck et al. (2023a) (S53) analyzed the Docker Hub image inheritance network, demonstrating how quality issues propagate across image layers and organizational boundaries.

These contributions illustrate that system-level IaC analysis is both necessary and tractable, but remains at an early stage. Future approaches should broaden analysis granularity to capture inter-script dependencies, module interaction failures, and project-wide configuration inconsistencies. By treating IaC as an interconnected configuration ecosystem rather than a collection of isolated files, such approaches would enable more holistic quality assessments and provide engineers with insights that script-level tools are structurally unable to deliver.

Circular dependencies between Terraform modules, conflicting Kubernetes resource selectors across namespaces, Docker base-image drift across services are typical limitations of single-file linters. Practitioners can begin to address this by enforcing project-wide checks through dependency graphs across modules, naming conventions, and resource quotas at the project level. The cross-resource semantic checks of Qiu et al. (2024) and the image-inheritance analyses of Opdebeeck et al. (2023a) are first demonstrations of what is achievable today.

8. Threats to validity

To ensure the credibility of our findings, we discuss threats to validity according to the four dimensions proposed by Wohlin et al. (2012).

8.1. Construct validity

Construct validity refers to how accurately the study's methods and measures capture the theoretical concepts they are intended to represent Wohlin et al. (2012). In our review, construct validity concerns whether the definitions of concepts, such as Infrastructure as Code, quality issues, support approaches, and their characteristics, actually represent their meaning in the IaC domain. To mitigate this risk, we used an iterative process, supported by multiple validation rounds, where definitions were refined through comparison with existing taxonomies and IaC-specific terminology. Moreover, before starting the classification of studies, we have explicitly mentioned the definitions of the different concepts we were using for classifying the papers (e.g., the definitions of quality issues and support approaches). These efforts were done to ensure a common starting point between the authors, and to reduce the bias that may be introduced due to subjective interpretation of concepts.

8.2. Internal validity

Internal validity addresses whether observed patterns are credible and not influenced by confounding factors Wohlin et al. (2012), ensuring rigor in data collection. For this work, internal validity is primarily influenced by the inclusion/exclusion of studies and consistency in data extraction. Selection bias was mitigated through explicitly defined inclusion/exclusion criteria applied consistently across all four digital libraries. For data extraction, all authors collectively established and documented explicit definitions for every dimension prior to extraction, ensuring a shared interpretive baseline. A sample of 25 studies (20%) was independently double-coded, yielding Cohen's kappa $\kappa = 0.9$ (almost perfect agreement). The remaining studies were extracted by the first author under the same definitional framework, with ambiguous cases flagged and resolved collectively at weekly meetings. While single-author extraction represents a residual threat, the combination of prior definitional alignment, demonstrated inter-rater agreement, and systematic collective resolution of uncertain cases provides reasonable confidence in extraction consistency.

8.3. External validity

External validity concerns the extent to which the study's findings can be generalized beyond the analyzed sample Wohlin et al. (2012). This may be threatened by the limited coverage of digital libraries and the uneven distribution of studies across IaC technologies. Our technology-agnostic search query, while designed to avoid tool-specific bias, may have missed studies framing IaC quality under technology-specific terminology, such as Dockerfile smells or Kubernetes manifest analysis. This reflects a deliberate trade-off, a technology-specific query would have risked overrepresenting certain ecosystems and undermining the unified view of IaC quality we sought to synthesize. However, through the three iterations of snowballing, we tried to overcome this limitation by looking for studies that were connected through deeper citation chains. Additionally, the exclusion of grey literature may overlook relevant industrial practices, and the underrepresentation of certain technologies (e.g., Chef, TOSCA, Helm) limits generalizability to the broader IaC ecosystem. All together, while our synthesis offers a representative view of current research, it should be interpreted as a reflection of the accessible academic literature rather than the entire IaC ecosystem.

8.4. Conclusion validity

Conclusion validity examines the reliability and validity of conclusions Wohlin et al. (2012). To mitigate this, and to give transparency to our results, we have created a replication package tracing the different steps of our methodology. Nevertheless, as the SLR is based on the analysis of different studies with varying quality, some conclusions, particularly those about under-represented IaC platforms, or used techniques, should be viewed as indicative rather than definitive.

9. Conclusion

This study provides a synthesis of the current state of IaC quality assurance, through an SLR analyzing both quality issues and support approaches. Our taxonomy presented as a feature model highlights the fragmented nature of existing efforts, dominated by technology-specific, implementation-time analyses, and emphasizes the need for more technology-independent, and context-aware investigations. From a methodological perspective, current research remains largely static and tool-based, with limited validation in industrial contexts. This gap underscores the necessity for stronger collaboration between academia and practice to develop solutions that are not only technically sound but also operationally relevant. Future work should consider addressing under-studied quality aspects (e.g., performance issues), and a paradigm shift from tool-specific, static analysis toward more adaptive, context-aware, and cross-layer IaC quality frameworks.

Acknowledgments

This work was supported by a French government grant managed by the Agence Nationale de la Recherche under the France 2030 program, reference "ANR-23-PECL-0008".

Data availability

To support transparency, all materials produced during this systematic literature review are archived in a permanent replication package (El Hayani et al., 2026). Hosted on Zenodo with a citable DOI and mirrored on GitHub, the package comprises: (i) the process overview for reproducing the results; (ii) the quasi-gold set, its construction protocol, the venue list and per-venue manual-search records; (iii) the analysis scripts with documentation; (iv) the data produced ranging from initial search results through intermediate screening stages to the final paper classification. Together, these artifacts allow verification of the search process, replication of the screening pipeline, regeneration of figures, and extension of this review as new primary studies emerge.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT, Gemini, and Grammarly in order to improve readability, rephrase sentences, and correct grammatical errors. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Table .9: List of studies included in the Systematic Literature Review

ID	Title	Year	Reference
S1	A Comparative Study on the Security of Kubernetes Deployments	2024	Kamieniarz and Mazurczyk (2024)
S2	A Framework for Measuring the Quality of Infrastructure-as-Code Scripts	2025	Konala et al. (2025a)
S3	A Pilot Study of Testing Infrastructure as Code for Cloud Systems	2023	Suwanachote et al. (2023)
S4	An empirical study of task infections in Ansible scripts	2023	Rahman et al. (2023a)
S5	An Empirical Study on Kubernetes Operator Bugs	2024	Xu et al. (2024)
S6	Ansible Lightspeed: A Code Generation Service for IT Automation	2024	Sahoo et al. (2024)
S7	AnsibleMetrics: A Python library for measuring Infrastructure-as-Code blueprints in Ansible	2020	Dalla Palma et al. (2020b)
S8	As Code Testing: Characterizing Test Quality in Open Source Ansible Development	2022	Hassan and Rahman (2022)
S9	Assessing and Improving the Quality of Docker Artifacts	2022	Rosa et al. (2022)
S10	Assessing the adoption of security policies by developers in terraform across different cloud providers	2025	Verdet et al. (2025)
S11	Automated Infrastructure as Code Program Testing	2024	Sokolowski et al. (2024a)
S12	Automatically Generating Dockerfiles via Deep Learning: Challenges and Promises	2023	Rosa et al. (2023)
S13	Characterizing Co-located Insecure Coding Patterns in Infrastructure as Code Scripts	2020	Bhuiyan and Rahman (2020)
S14	Characterizing Static Analysis Alerts for Terraform Manifests: An Experience Report	2023	Hu et al. (2023)
S15	Co-evolution of Infrastructure and Source Code - An Empirical Study	2015	Jiang and Adams (2015)
S16	Code Smells in Infrastructure as Code	2018	Schwarz et al. (2018)
S17	Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort?	2023	Opdebeeck et al. (2023b)
S18	DeepIaC: deep learning-based linguistic anti-pattern detection in IaC	2020	Borovits et al. (2020)
S19	Detecting and Characterizing Propagation of Security Weaknesses in Puppet-Based Infrastructure Management	2023	Rahman and Parnin (2023)
S20	Detecting and Resolving Coupling-Related Infrastructure as Code Based Architecture Smells in Microservice Deployments	2023	Ntentos et al. (2023)
S21	Defuse: A Data Annotator and Model Builder for Software Defect Prediction	2022	Dalla Palma et al. (2022b)
S22	Dockerlive: A live development environment for Dockerfiles	2022	Reis and Correia (2022)
S23	Does Your Configuration Code Smell?	2016	Sharma et al. (2016)
S24	Empirical Study of the Docker Smells Impact on the Image Size	2024	Durieux (2024)
S25	Exploring LLM-Based Automated Repairing of Ansible Script in Edge-Cloud Infrastructures	2023	Kwon et al. (2023)
S26	Extensible Testing for Infrastructure as Code	2023	Spielmann et al. (2023)
S27	Fine-Grained Just-In-Time Defect Prediction at the Block Level in Infrastructure-as-Code (IaC)	2024	Begoug et al. (2024b)
S28	Gang of Eight: A Defect Taxonomy for Infrastructure as Code Scripts	2020	Rahman et al. (2020a)
S29	How good is your puppet? An empirically defined and validated quality model for puppet	2018	van der Bent et al. (2018)
S30	Infrastructure as code for dynamic deployments	2022	Sokolowski (2022)

ID	Title	Year	Ref
S31	InfraFix: Technology-Agnostic Repair of Infrastructure as Code	2025	Saavedra et al. (2025)
S32	K8s Pro Sentinel: Extend Secret Security in Kubernetes Cluster	2024	Gunathilake and Ekanayake (2024)
S33	KubeHound: Detecting Microservices' Security Smells in Kubernetes Deployments	2023	Dell'Immagine et al. (2023)
S34	Lessons from Research to Practice on Writing Better Quality Puppet Scripts	2022	Rahman and Sharma (2022)
S35	Leveraging Practitioners' Feedback to Improve a Security Linter	2023	Reis et al. (2023)
S36	LLM Agentic Workflow for Automated Vulnerability Detection and Remediation in Infrastructure-as-Code	2025	Toprani and Madisetti (2025)
S37	On the Prevalence, Co-occurrence, and Impact of Infrastructure-as-Code Smells	2024	Bessghaier et al. (2024)
S38	Patterns of multi-container composition for service orchestration with Docker Compose	2024	Eng et al. (2024)
S39	Polyglot Code Smell Detection for Infrastructure as Code with GLITCH	2023	Saavedra et al. (2023)
S40	Rehearsal: a configuration verification tool for puppet	2016	Shambaugh et al. (2016)
S41	Repairing Infrastructure-as-Code using Large Language Models	2024	Low et al. (2024)
S42	Securing Infrastructure as Code (IaC) through DevSecOps: A Comprehensive Risk Management Framework	2023	Zeini et al. (2023)
S43	Security in DevSecOps: Applying Tools and Machine Learning to Verification and Monitoring Steps	2023	Cankar et al. (2023)
S44	Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study	2023	Rahman et al. (2023b)
S45	Security Smells in Ansible and Chef Scripts: A Replication Study	2021	Rahman et al. (2021b)
S46	Smells-sus: Sustainability Smells in IaC	2025	Kosbar and Hamdaqa (2025)
S47	Smelly variables in ansible infrastructure code: detection, prevalence, and lifetime	2022	Opdebeeck et al. (2022)
S48	Smoke Testing of Cloud Systems	2022	Cannavacciuolo and Mariani (2022b)
S49	Source code properties of defective infrastructure as code scripts	2019	Rahman and Williams (2019)
S50	State Reconciliation Defects in Infrastructure as Code	2024	Hassan et al. (2024)
S51	Statically Inferring Usage Bounds for Infrastructure as Code	2025	Qiao et al. (2025)
S52	TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform	2024	Begoug et al. (2024a)
S53	The Docker Hub Image Inheritance Network: Construction and Empirical Insights	2023	Opdebeeck et al. (2023a)
S54	The Seven Sins: Security Smells in Infrastructure as Code Scripts	2019	Rahman et al. (2019b)
S55	"Through the looking-glass ..." An empirical study on blob infrastructure blueprints in the Topology and Orchestration Specification for Cloud Applications	2024	Dalla Palma et al. (2024)
S56	Tortoise: Interactive system configuration repair	2017	Weiss et al. (2017)
S57	Toward a catalog of software quality metrics for infrastructure code	2020	Dalla Palma et al. (2020a)
S58	Towards a Taxonomy of Infrastructure as Code Misconfigurations: An Ansible Study	2024	Nasiri et al. (2024)
S59	Towards Reliable Infrastructure as Code	2023	Sokolowski and Salvaneschi (2023)

ID	Title	Year	Ref
S60	Towards Sustainable Cloud Deployments: A Cost (Anti)Patterns Catalog for Terraform and Cloudformation	2025	Bolhuis et al. (2025)
S61	Towards the self-healing of Infrastructure as Code projects using constrained LLM technologies	2024	Diaz-De-Arcaya et al. (2024)
S62	Unearthing Semantic Checks for Cloud Infrastructure-as-Code Programs	2024	Qiu et al. (2024)
S63	Uncovering Threats in Container Systems: A Study on Misconfigured Container Components in the Wild	2024	Choi et al. (2024)
S64	Within-Project Defect Prediction of Infrastructure-as-Code Using Product and Process Metrics	2022	Dalla Palma et al. (2022a)
S65	An empirical analysis of the docker container ecosystem on github	2017	Cito et al. (2017)
S66	Dockercleaner: Automatic repair of security smells in dockerfiles	2023	Bui et al. (2023)
S67	It Works (only) on My Machine: A Study on Reproducibility Smells in Ansible Scripts	2025	Sobhani et al. (2025)
S68	Singling the odd ones out: a novelty detection approach to find defects in infrastructure-as-code	2020	Dalla Palma et al. (2020)
S69	Automatically detecting risky scripts in infrastructure code	2020	Dai et al. (2020)
S70	Shhh!: 12 practices for secret management in infrastructure as code	2021	Rahman et al. (2021a)
S71	A Catalog of Cost Patterns and Antipatterns for Infrastructure as Code	2024	Bolhuis et al. (2024)
S72	An Approach to Identifying Error Patterns for Infrastructure as Code	2018	Chen et al. (2018)
S73	Analyzing and mitigating (with LLMs) the security misconfigurations of Helm charts from Artifact Hub	2025	Minna et al. (2025)
S74	Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities	2021	Lepiller et al. (2021)
S75	Asserting reliable convergence for configuration management scripts	2016	Hanappi et al. (2016)
S76	Assessing Architecture Conformance to Security-Related Practices in Infrastructure as Code Based Deployments	2022	Ntentos et al. (2022a)
S77	Assessing architecture conformance to coupling-related infrastructure-as-code best practices: Metrics and case studies	2022	Ntentos et al. (2022b)
S78	Automated Bug Discovery in Cloud Infrastructure-as-Code Updates with LLM Agents	2025	Xiang et al. (2025)
S79	Characterizing Defective Configuration Scripts Used for Continuous Deployment	2018	Rahman and Williams (2018)
S80	Code Smell-Guided Prompting for LLM-Based Defect Prediction in Ansible Scripts	2024	Hong et al. (2024a)
S81	Compliance Management of IaC-Based Cloud Deployments During Runtime	2024	Falazi et al. (2024)
S82	Container security: precaution levels, mitigation strategies, and research perspectives	2023	VS et al. (2023)
S83	Different Kind of Smells: Security Smells in Infrastructure as Code Scripts	2021	Rahman and Williams (2021)
S84	Does Infrastructure as Code Adhere to Semantic Versioning? An Analysis of Ansible Role Evolution	2020	Opdebeeck et al. (2020)
S85	DRIVE: Dockerfile Rule Mining and Violation Detection	2023	Zhou et al. (2023)
S86	DRMiner: A Tool For Identifying And Analyzing Refactorings In Dockerfile	2024	Ksontini et al. (2024)

ID	Title	Year	Ref
S87	Enhancing Secure Deployment with Ansible: A Focus on Least Privilege and Automation for Linux	2024	Billoir et al. (2024)
S88	Enhancing software defect prediction in ansible scripts using code-smell-guided prompting with large language models in edge-cloud infrastructures	2024	Hong et al. (2024b)
S89	FindICI: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code	2022	Borovits et al. (2022)
S90	Harnessing the Power of LLMs for Code Smell Detection in Terraform Infrastructure as Code	2025	Vo et al. (2025)
S91	Hybrid Fuzzing of Infrastructure as Code Programs (Short Paper)	2025	Coppa et al. (2025)
S92	On the practice of semantic versioning for Ansible galaxy roles: An empirical study and a change classification model	2021	Opdebeeck et al. (2021b)
S93	On the Understandability of Design-Level Security Practices in Infrastructure-as-Code Scripts and Deployment Architectures	2024	Ntentos et al. (2024)
S94	Optimizing resource allocation using proactive scaling with predictive models and custom resources	2024	Kumar et al. (2024)
S95	Practical fault detection in puppet programs	2020	Sotiropoulos et al. (2020)
S96	Refactoring for Dockerfile Quality: A Dive into Developer Practices and Automation Potential	2025	Ksontini et al. (2025)
S97	Secure container Orchestration: A framework for detecting and mitigating Orchestrator-level vulnerabilities	2025	Mahavaishnavi et al. (2025)
S98	Shipwright: A Human-in-the-Loop System for Dockerfile Repair	2021	Henkel et al. (2021)
S99	Sommelier: a tool for validating TOSCA application topologies	2017	Brogi et al. (2017)
S100	TerrARA: Automated Security Threat Modeling for Infrastructure as Code	2025	Tran et al. (2025)
S101	Test suite reduction in idempotence testing of infrastructure as code	2017	Ikeshita et al. (2017)
S102	Test-Based Least Privilege Discovery on Cloud Infrastructure as Code	2020	Shimizu and Kanuka (2020)
S103	The 'as code' activities: development anti-patterns for infrastructure as code	2020	Rahman et al. (2020b)
S104	Towards Semantic Detection of Smells in Cloud Infrastructure Code	2020	Kumara et al. (2020)
S105	Vulnerabilities in infrastructure as code: what, how many, and who?	2025	War et al. (2025)
S106	RUDSEA: Recommending Updates of Dockerfiles via Software Environment Analysis	2018	Hassan et al. (2018)
S107	Metadata Assisted Supply-Chain Attack Detection for Ansible	2025	Konala et al. (2025b)
S108	Kivi: Verification for Cluster Management	2024	Liu et al. (2024)
S109	EPScan: Automated Detection of Excessive RBAC Permissions in Kubernetes Applications	2025	Gu et al. (2025)
S110	Automatic Generation of Smoke Test Suites for Kubernetes	2022	Cannavacciuolo and Mariani (2022a)
S111	An Insight Into the Impact of Dockerfile Evolutionary Trajectories on Quality and Latency	2018	Zhang et al. (2018)
S112	DockInsight: A Knowledge-Augmented Dependency Extraction Approach for Dockerfile	2025	Zhu et al. (2025)

ID	Title	Year	Ref
S113	Characterizing the Occurrence of Dockerfile Smells in Open-Source Software: An Empirical Study	2020	Wu et al. (2020)
S114	An Empirical Case Study on the Temporary File Smell in Dockerfiles	2019	Lu et al. (2019)
S115	Dr. Docker: A Large-Scale Security Measurement of Docker Image Ecosystem	2025	Shi et al. (2025)
S116	Dockerfile TF Smell Detection Based on Dynamic and Static Analysis Methods	2019	Xu et al. (2019)
S117	Refactorings and Technical Debt in Docker Projects: An Empirical Study	2021	Ksontini et al. (2021)
S118	Anvil: Verifying Liveness of Cluster Management Controllers	2024	Sun et al. (2024)
S119	KubeFence: Security Hardening of the Kubernetes Attack Surface	2025	Cesarano and Natella (2025)
S120	DFScan: Security Scanner of the Dockerfile Based on Instruction Coverage and Attack Perspective	2024	Hao et al. (2024)
S121	Not all Dockerfile Smells are the Same: An Empirical Evaluation of Hadolint Writing Practices by Experts	2024	Rosa et al. (2024a)
S122	Fixing Dockerfile Smells: An Empirical Study	2024	Rosa et al. (2024b)
S123	CONTAIN4n6: A Systematic Evaluation of Container Artifacts	2022	Mishra et al. (2022)

References

, 2010. Ieee standard classification for software anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) , 1–23doi:doi: 10.1109/IEEESTD.2010.5399061.

Begoug, M., Bessghaier, N., Ouni, A., AlOmar, E.A., Mkaouer, M.W., 2023. What do infrastructure-as-code practitioners discuss: An empirical study on stack overflow, in: 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–12. doi:doi: 10.1109/ESEM56168.2023.10304847.

Begoug, M., Chouchen, M., Ouni, A., 2024a. Terrametrics: An open source tool for infrastructure-as-code (iac) quality metrics in terraform, in: Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, Association for Computing Machinery, New York, NY, USA. p. 450–454. doi:doi: 10.1145/3643916.3644439.

Begoug, M., Chouchen, M., Ouni, A., Abdullah Alomar, E., Mkaouer, M.W., 2024b. Fine-grained just-in-time defect prediction at the block level in infrastructure-as-code (iac), in: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA. p. 100–112. doi:doi: 10.1145/3643991.3644934.

van der Bent, E., Hage, J., Visser, J., Gousios, G., 2018. How good is your puppet? an empirically defined and validated quality model for puppet, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 164–174. doi:doi: 10.1109/SANER.2018.8330206.

Bessghaier, N., Begoug, M., Mebarki, C., Ouni, A., Sayagh, M., Mkaouer, M.W., 2024. On the prevalence, co-occurrence, and impact of infrastructure-as-code smells, in: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 23–34. doi:doi: 10.1109/SANER60148.2024.00009.

Bhuiyan, F.A., Rahman, A., 2020. Characterizing co-located insecure coding patterns in infrastructure as code scripts, in: 2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), pp. 27–32. doi:doi: 10.1145/3417113.3422154.

Billoir, E., Laborde, R., Wazan, A.S., Rutschle, Y., Benzekri, A., 2024. Enhancing secure deployment with ansible: A focus on least privilege and automation for linux, in: Proceedings of the 19th International Conference on Availability, Reliability and Security, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/3664476.3670929.

Bolhuis, K., Feitosa, D., Andrikopoulos, V., 2024. A catalog of cost patterns and antipatterns for infrastructure as code, in: 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 399–406. doi:doi: 10.1109/SEAA64295.2024.00067.

Bolhuis, K., Neamt, A., Feitosa, D., Andrikopoulos, V., 2025. Towards Sustainable Cloud Deployments: A Cost (Anti)Patterns Catalog for Terraform and Cloudformation. Working Paper 5137165. SSRN. doi:doi: 10.2139/ssrn.5137165.

Borovits, N., Kumara, I., Di Nucci, D., Krishnan, P., Palma, S.D., Palomba, F., Tamburri, D.A., Heuvel, W.J.v.d., 2022. Findici: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code. Empirical Software Engineering 27, 178.

Borovits, N., Kumara, I., Krishnan, P., Palma, S.D., Di Nucci, D., Palomba, F., Tamburri, D.A., van den Heuvel, W.J., 2020. Deepiac: deep learning-based linguistic anti-pattern detection in iac, in: Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation, Association for Computing Machinery, New York, NY, USA. p. 7–12. doi:doi: 10.1145/3416505.3423564.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software 80, 571–583. doi:doi: https://doi.org/10.1016/j.jss.2006.07.009. software Performance.

Brogi, A., Di Tommaso, A., Soldani, J., 2017. Sommelier: a tool for validating tosca application topologies, in: International Conference on Model-Driven Engineering and Software Development, Springer. pp. 1–22.

Brown, W.H., Malveau, R.C., McCormick, H.W.S., Mowbray, T.J., 1998. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. 1st ed., John Wiley & Sons, Inc., USA.

Buhler, C., Spielmann, D., Meier, R., Salvaneschi, G., 2025. TerraDS: A Dataset for Terraform HCL Programs , in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), IEEE Computer Society, Los Alamitos, CA, USA. pp. 654–658. doi:doi: 10.1109/MSR66628.2025.00101.

Bui, Q.C., Laukötter, M., Scandariato, R., 2023. Dockercleaner: Automatic repair of security smells in dockerfiles, in: 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 160–170. doi:doi: 10.1109/ICSME58846.2023.00026.

Cankar, M., Petrovic, N., Pita Costa, J., Cernivec, A., Antic, J., Martincic, T., Stepec, D., 2023. Security in devsecops: Applying tools and machine learning to verification and monitoring steps, in: Companion of the 2023 ACM/SPEC International Conference on Performance Engineering, Association for Computing Machinery, New York, NY, USA. p. 201–205. doi:doi: 10.1145/3578245.3584943.

Cannavacciuolo, C., Mariani, L., 2022a. Automatic generation of smoke test suites for kubernetes, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA. p. 769–772. URL: https://doi.org/10.1145/3533767.3543298, doi:doi: 10.1145/3533767.3543298.

Cannavacciuolo, C., Mariani, L., 2022b. Smoke testing of cloud systems, in: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 47–57. doi:doi: 10.1109/ICST53961.2022.00016.

Cesarano, C., Natella, R., 2025. Kubefence: Security hardening of the kubernetes attack surface, in: 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 497–510. doi:doi: 10.1109/DSN64029.2025.00054.

Chen, W., Wu, G., Wei, J., 2018. An approach to identifying error patterns for infrastructure as code, in: 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 124–129. doi:doi: 10.1109/ISSREW.2018.00-19.

Chiari, M., De Pascalis, M., Pradella, M., 2022. Static analysis of infrastructure as code: a survey, in: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), pp. 218–225. doi:doi: 10.1109/ICSA-C54293.2022.00049.

Choi, D., Seo, H., Kim, K., You, M., Shin, S., Kim, J., 2024. Uncovering threats in container systems: A study on misconfigured container components in the wild. IEEE Access 12, 192931–192945. doi:doi: 10.1109/ACCESS.2024.3514751.

Cito, J., Schermann, G., Wittern, J.E., Leitner, P., Zumberi, S., Gall, H.C., 2017. An empirical analysis of the docker container ecosystem on github, in: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 323–333. doi:doi: 10.1109/MSR.2017.67.

Coppa, E., Sokolowski, D., Salvaneschi, G., 2025. Hybrid fuzzing of infrastructure as code programs (short paper), in: Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA. p. 92–97. doi:doi: 10.1145/3713081.3731721.

Dai, T., Karve, A., Koper, G., Zeng, S., 2020. Automatically detecting risky scripts in infrastructure code, in: Proceedings of the 11th ACM Symposium on Cloud Computing, Association for Computing Machinery, New York, NY, USA. p. 358–371. doi:doi: 10.1145/3419111.3421303.

Dalla Palma, S., van Asseldonk, C., Catolino, G., Di Nucci, D., Palomba, F., Tamburri, D.A., 2024. "Through the looking-glass . . . " An empirical study on blob infrastructure blueprints in the Topology and Orchestration Specification for Cloud Applications. Journal of Software: Evolution and Process 36, e2533. doi:doi: https://doi.org/10.1002/smr.2533.

Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A., 2020a. Toward a catalog of software quality metrics for infrastructure code. Journal of Systems and Software 170, 110726. doi:doi: https://doi.org/10.1016/j.jss.2020.110726.

Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A., 2022a. Within-project defect prediction of infrastructure-as-code using product and process metrics. IEEE Transactions on Software Engineering 48, 2086–2104. doi:doi: 10.1109/TSE.2021.3051492.

Dalla Palma, S., Di Nucci, D., Tamburri, D., 2022b. Defuse: A data annotator and model builder for software defect prediction, in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 479–483. doi:doi: 10.1109/ICSME55016.2022.00063.

Dalla Palma, S., Di Nucci, D., Tamburri, D.A., 2020b. Ansiblemetrics: A python library for measuring infrastructure-as-code blueprints in ansible. SoftwareX 12, 100633. doi:doi: https://doi.org/10.1016/

j.softx.2020.100633.

Dalla Palma, S., Mohammadi, M., Di Nucci, D., Tamburri, D.A., 2020. Singling the odd ones out: a novelty detection approach to find defects in infrastructure-as-code, in: Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation, Association for Computing Machinery, New York, NY, USA. p. 31–36. doi:doi: 10.1145/3416505.3423563.

Dell'Immagine, G., Soldani, J., Brogi, A., 2023. Kubehound: Detecting microservices' security smells in kubernetes deployments. Future Internet 15. doi:doi: 10.3390/fi15070228.

Diarra, B., Guillouard, K., Ouzzif, M., Merle, P., Stefani, J.B., 2024. In-depth analysis of kubernetes manifest verification tools for robust cnf deployment, in: 2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN), pp. 17–24. doi:doi: 10.1109/ICIN60470.2024.10494445.

Diaz-De-Arcaya, J., López-De-Armentia, J., Zárate, G., Torre-Bastida, A.I., 2024. Towards the self-healing of infrastructure as code projects using constrained llm technologies, in: Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair, Association for Computing Machinery, New York, NY, USA. p. 22–25. doi:doi: 10.1145/3643788.3648014.

Drosos, G.P., Sotiropoulos, T., Alexopoulos, G., Mitropoulos, D., Su, Z., 2024. When your infrastructure is a buggy program: Understanding faults in infrastructure as code ecosystems. Proc. ACM Program. Lang. 8. doi:doi: 10.1145/3689799.

Durieux, T., 2024. Empirical study of the docker smells impact on the image size, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/3597503.3639143.

El Hayani, H., Philippe, J., Challita, S., Barais, O., Benoit, C., 2026. Replication package: Quality assurance in infrastructure as code — issues, approaches, and open challenges. doi:doi: doi.org/10.5281/zenodo.20379472.

Eng, K., Hindle, A., Stroulia, E., 2024. Patterns of multi-container composition for service orchestration with Docker Compose. Empirical Software Engineering 29. doi:doi: 10.1007/s10664-024-10462-8. publisher: Springer Science and Business Media LLC.

Falazi, G., Harzenetter, L., Képes, K., Leymann, F., Breitenbücher, U., Ntentos, E., Zdun, U., Becker, M., Heldwein, E., 2024. Compliance management of iac-based cloud deployments during runtime, in: Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/3603166.3632135.

Forsgren, N., Smith, D., Humble, J., Frazelle, J., 2019. 2019 accelerate state of devops report. https://services.google.com/fh/files/misc/state-of-devops-2019.pdf. A DORA and Google Cloud collaboration.

Gu, Y., Tan, X., Zhang, Y., Gao, S., Yang, M., 2025. Epscan: Automated detection of excessive rbac permissions in kubernetes applications, in: 2025 IEEE Symposium on Security and Privacy (SP), pp. 3199–3217. doi:doi: 10.1109/SP61157.2025.00011.

Guerriero, M., Garriga, M., Tamburri, D.A., Palomba, F., 2019. Adoption, support, and challenges of infrastructure-as-code: Insights from industry, in: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 580–589. doi:doi: 10.1109/ICSME.2019.00092.

Gunathilake, K., Ekanayake, I., 2024. K8s pro sentinel: Extend secret security in kubernetes cluster, in: 2024 9th International Conference on Information Technology Research (ICITR), pp. 1–5. doi:doi: 10.1109/ICITR64794.2024.10857769.

Hanappi, O., Hummer, W., Dustdar, S., 2016. Asserting reliable convergence for configuration management scripts. SIGPLAN Not. 51, 328–343. doi:doi: 10.1145/3022671.2984000.

Hao, J., Lu, H., Jiang, Y., Gupta, B.B., Almomani, A., Zhang, M., Tian, Z., 2024. Dfscan: Security scanner of the dockerfile based on instruction coverage and attack perspective. Human-centric Computing and Information Sciences 14, 10.

Hasan, M.M., Bhuiyan, F.A., Rahman, A., 2020. Testing practices for infrastructure as code, Association for Computing Machinery, New York, NY, USA. p. 7–12. doi:doi: 10.1145/3416504.3424334.

Hassan, F., Rodriguez, R., Wang, X., 2018. Rudsea: recommending updates of dockerfiles via software environment analysis, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 796–801. URL: https://doi.org/10.1145/3238147.3240470, doi:doi: 10.1145/3238147.3240470.

Hassan, M.M., Rahman, A., 2022. As code testing: Characterizing test quality in open source ansible development, in: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST), pp. 208–219. doi:doi: 10.1109/ICST53961.2022.00031.

Hassan, M.M., Salvador, J., Santu, S.K.K., Rahman, A., 2024. State reconciliation defects in infrastructure as code. Proc. ACM Softw. Eng. 1. doi:doi: 10.1145/3660790.

Henkel, J., Silva, D., Teixeira, L., d'Amorim, M., Reps, T., 2021. Shipwright: A human-in-the-loop system for dockerfile repair, in: Proceedings of the 43rd International Conference on Software Engineering, IEEE Press. p. 1148–1160. doi:doi: 10.1109/ICSE43902.2021.00106.

Hong, H., Lee, S., Ryu, D., Baik, J., 2024a. Code smell-guided prompting for llm-based defect prediction in ansible scripts. Journal of Web Engineering 23, 1107–1126. doi:doi: 10.13052/jwe1540-9589.2383.

Hong, H., Lee, S., Ryu, D., Baik, J., 2024b. Enhancing software defect prediction in ansible scripts using code-smell-guided prompting with large language models in edge-cloud infrastructures, in: International Conference on Web Engineering, Springer. pp. 30–42.

Hu, H., Bu, Y., Wong, K., Sood, G., Smiley, K., Rahman, A., 2023. Characterizing static analysis alerts for terraform manifests: An experience report, in: 2023 IEEE Secure Development Conference (SecDev), pp. 7–13. doi:doi: 10.1109/SecDev56634.2023.00014.

Humble, J., Farley, D., 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.

Ikeshita, K., Ishikawa, F., Honiden, S., 2017. Test suite reduction in idempotence testing of infrastructure as code, in: International Conference on Tests and Proofs, Springer. pp. 98–115.

Jiang, Y., Adams, B., 2015. Co-evolution of infrastructure and source code - an empirical study, in: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 45–55. doi:doi: 10.1109/MSR.2015.12.

Kamieniarz, K., Mazurczyk, W., 2024. A comparative study on the security of kubernetes deployments, in: 2024 International Wireless Communications and Mobile Computing (IWCMC), pp. 0718–0723. doi:doi: 10.1109/IWCMC61514.2024.10592468.

Kitchenham, B., Charters, S., et al., 2007. Guidelines for performing systematic literature reviews in software engineering. Software Engineering Group, EBSE Technical Report, Keele University and Department of Computer Science University of Durham .

Konala, P.R.R., Kumar, V., Bainbridge, D., Haseeb, J., 2025a. A framework for measuring the quality of infrastructure-as-code scripts. URL: https://arxiv.org/abs/2502.03127, arXiv:2502.03127.

Konala, P.R.R., Kumar, V., Bainbridge, D., Haseeb, J., 2025b. Metadata assisted supply-chain attack detection fornbsp;ansible, in: Data and Applications Security and Privacy XXXIX: 39th IFIP WG 11.3 Annual Conference on Data and Applications Security and Privacy, DBSec 2025, Gjøvik, Norway, June 23-24, 2025, Proceedings, Springer-Verlag, Berlin, Heidelberg. p. 333–350. URL: https://doi.org/10.1007/978-3-031-96590-6_18, doi:doi: 10.1007/978-3-031-96590-6_18.

Kosbar, S., Hamdaqa, M., 2025. Smells-sus: Sustainability smells in iac, in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), pp. 801–812. doi:doi: 10.1109/MSR66628.2025.00117.

Ksontini, E., Abid, A., Khalsi, R., Kessentini, M., 2024. Drminer: A tool for identifying and analyzing refactorings in dockerfile, in: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA. p. 584–594. doi:doi: 10.1145/3643991.3644921.

Ksontini, E., Kessentini, M., Ferreira, T.d.N., Hassan, F., 2021. Refactorings and technical debt in docker projects: An empirical study, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 781–791. doi:doi: 10.1109/ASE51524.2021.9678585.

Ksontini, E., Mastouri, M., Khalsi, R., Kessentini, W., 2025. Refactoring for dockerfile quality: A dive into developer practices and automation potential, in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), pp. 788–800. doi:doi: 10.1109/MSR66628.2025.00116.

Kumar, B., Verma, A., Verma, P., 2024. Optimizing resource allocation using proactive scaling with predictive models and custom resources. Computers and Electrical Engineering 118,

109419. URL: https://www.sciencedirect.com/science/article/pii/S0045790624003471, doi:doi: https://doi.org/10.1016/j.compeleceng.2024.109419.

Kumara, I., Garriga, M., Romeu, A.U., Di Nucci, D., Palomba, F., Tamburri, D.A., van den Heuvel, W.J., 2021. The do's and don'ts of infrastructure code: A systematic gray literature review. Information and Software Technology 137, 106593. doi:doi: https://doi.org/10.1016/j.infsof.2021.106593.

Kumara, I., Vasileiou, Z., Meditskos, G., Tamburri, D.A., Van Den Heuvel, W.J., Karakostas, A., Vrochidis, S., Kompatsiaris, I., 2020. Towards semantic detection of smells in cloud infrastructure code, in: Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics, Association for Computing Machinery, New York, NY, USA. p. 63–67. doi:doi: 10.1145/3405962.3405979.

Kwon, S., Lee, S., Kim, T., Ryu, D., Baik, J., 2023. Exploring llm-based automated repairing of ansible script in edge-cloud infrastructures. Journal of Web Engineering 22, 889–912. doi:doi: 10.13052/jwe1540-9589.2263.

Landis, J.R., Koch, G.G., 1977. The measurement of observer agreement for categorical data. Biometrics 33, 159–174. URL: http://www.jstor.org/stable/2529310.

Laplante, P.A., 2007. What Every Engineer Should Know about Software Engineering (What Every Engineer Should Know). CRC Press, Inc., USA.

Lepiller, J., Piskac, R., Schäf, M., Santolucito, M., 2021. Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer. pp. 105–123.

Liu, B., Lim, G., Beckett, R., Godfrey, P.B., 2024. Kivi: verification for cluster management, in: Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference, USENIX Association, USA.

Low, E., Cheh, C., Chen, B., 2024. Repairing infrastructure-as-code using large language models, in: 2024 IEEE Secure Development Conference (SecDev), pp. 20–27. doi:doi: 10.1109/SecDev61143.2024.00008.

Lu, Z., Xu, J., Wu, Y., Wang, T., Huang, T., 2019. An empirical case study on the temporary file smell in dockerfiles. IEEE Access 7, 63650–63659. doi:doi: 10.1109/ACCESS.2019.2905424.

López-Fernández, D., Díaz, J., García, J., Pérez, J., González-Prieto, , 2022. Devops team structures: Characterization and implications. IEEE Transactions on Software Engineering 48, 3716–3736. doi:doi: 10.1109/TSE.2021.3102982.

Mahavaishnavi, V., Saminathan, R., Prithviraj, R., 2025. Secure container orchestration: A framework for detecting and mitigating orchestrator-level vulnerabilities. Multimedia Tools and Applications 84, 18351–18371.

Masoumzadeh, S., Saavedra, N., Maipradit, R., Wei, L., Ferreira, J.F., Varró, D., McIntosh, S., 2025. Do experts agree about smelly infrastructure? IEEE Transactions on Software Engineering 51, 1472–1486. doi:doi: 10.1109/TSE.2025.3553383.

Minna, F., Massacci, F., Tuma, K., 2025. Analyzing and mitigating (with llms) the security misconfigurations of helm charts from artifact hub. Empirical Software Engineering 30, 132.

Mishra, A.K., Pilli, E.S., Govil, M.C., 2022. Contain4n6: a systematic evaluation of container artifacts. Journal of Cloud Computing 11, 28.

Morris, K., 2020. Infrastructure as Code: Dynamic Systems for the Cloud Age. 2nd ed., O'Reilly Media, Inc.

Nasiri, R., Kumara, I., Tamburri, D.A., van den Heuvel, W.J., 2024. Towards a Taxonomy of Infrastructure as Code Misconfigurations: An Ansible Study, in: Service-Oriented Computing. Springer Nature Switzerland, pp. 83–103. doi:doi: 10.1007/978-3-031-72578-4_5. iSSN: 1865-0937.

Ntentos, E., Lueger, N.E., Simhandl, G., Zdun, U., Schneider, S., Scandariato, R., Díaz Ferreyra, N.E., 2024. On the understandability of design-level security practices in infrastructure-as-code scripts and deployment architectures. ACM Trans. Softw. Eng. Methodol. 34. doi:doi: 10.1145/3691630.

Ntentos, E., Zdun, U., Falazi, G., Breitenbücher, U., Leymann, F., 2022a. Assessing architecture conformance to security-related practices in infrastructure as code based deployments, in: 2022 IEEE International Conference on Services Computing (SCC), pp. 123–133. doi:doi: 10.1109/SCC55611.2022.00029.

Ntentos, E., Zdun, U., Falazi, G., Breitenbücher, U., Leymann, F., 2023. Detecting and resolving coupling-related infrastructure as code based architecture smells in microservice deployments, in: 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), pp. 201–211. doi:doi: 10.1109/CLOUD60044.2023.00031.

Ntentos, E., Zdun, U., Soldani, J., Brogi, A., 2022b. Assessing architecture conformance to coupling-related infrastructure-as-code best practices: Metrics and case studies, in: European Conference on Software Architecture, Springer. pp. 101–116.

Opdebeeck, R., Lesy, J., Zerouali, A., De Roover, C., 2023a. The docker hub image inheritance network: Construction and empirical insights, in: 2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 198–208. doi:doi: 10.1109/SCAM59687.2023.00029.

Opdebeeck, R., Zerouali, A., De Roover, C., 2021a. Andromeda: A dataset of ansible galaxy roles and their evolution, in: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp. 580–584. doi:doi: 10.1109/MSR52588.2021.00078.

Opdebeeck, R., Zerouali, A., De Roover, C., 2022. Smelly variables in ansible infrastructure code: detection, prevalence, and lifetime, in: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA. p. 61–72. doi:doi: 10.1145/3524842.3527964.

Opdebeeck, R., Zerouali, A., De Roover, C., 2023b. Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?, in: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 534–545. doi:doi: 10.1109/MSR59073.2023.00079.

Opdebeeck, R., Zerouali, A., Velázquez-Rodríguez, C., De Roover, C., 2021b. On the practice of semantic versioning for ansible galaxy roles: An empirical study and a change classification model. Journal of Systems and Software 182, 111059. doi:doi: doi.org/10.1016/j.jss.2021.111059.

Opdebeeck, R., Zerouali, A., Velázquez-Rodríguez, C., Roover, C.D., 2020. Does infrastructure as code adhere to semantic versioning? an analysis of ansible role evolution, in: 2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 238–248. doi:doi: 10.1109/SCAM51674.2020.00032.

Pahl, C., Gunduz, N., Övgüm Sezen, Ghamgosar, A., Ioini, N.E., 2025. Infrastructure as code: Technology review and research challenges, in: Proceedings of the 15th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER, INSTICC. SciTePress. pp. 151–158. doi:doi: 10.5220/0013247700003950.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology 64, 1–18. doi:doi: https://doi.org/10.1016/j.infsof.2015.03.007.

Qiao, F., Mohammadi, A., Cito, J., Santolucito, M., 2025. Statically inferring usage bounds for infrastructure as code, in: Protzenko, J., Raad, A. (Eds.), Verified Software. Theories, Tools and Experiments, Springer Nature Switzerland, Cham. pp. 84–95.

Qiu, Y., Kon, P.T.J., Beckett, R., Chen, A., 2024. Unearthing semantic checks for cloud infrastructure-as-code programs, in: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, Association for Computing Machinery, New York, NY, USA. p. 574–589. doi:doi: 10.1145/3694715.3695974.

Rahman, A., 2020. Dataset for 'Security Smells for Ansible and Chef Scripts: A Replication Study' URL: https://doi.org/10.6084/m9.figshare.8085755, doi:doi: 10.6084/m9.figshare.8085755.v2.

Rahman, A., Barsha, F.L., Morrison, P., 2021a. Shhh!: 12 practices for secret management in infrastructure as code, in: 2021 IEEE Secure Development Conference (SecDev), pp. 56–62. doi:doi: 10.1109/SecDev51306.2021.00024.

Rahman, A., Bose, D.B., Zhang, Y., Pandita, R., 2023a. An empirical study of task infections in ansible scripts. Empirical Softw. Engg. 29. doi:doi: 10.1007/s10664-023-10432-6.

Rahman, A., Farhana, E., Parnin, C., Williams, L., 2020a. Gang of eight: A defect taxonomy for infrastructure as code scripts, in: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 752–764. doi:doi: 10.1145/3377811.3380409.

Rahman, A., Farhana, E., Williams, L., 2020b. The 'as code' activities: development anti-patterns for infrastructure as code. Empirical Softw. Engg. 25, 3430–3467. doi:doi: 10.1007/s10664-020-09841-8.

Rahman, A., Mahdavi-Hezaveh, R., Williams, L., 2019a. A systematic mapping study of infrastructure as code research. Information and Software Technology 108, 65–77. doi:doi: https://doi.org/10.1016/j.infsof.2018.12.004.

Rahman, A., Parnin, C., 2023. Detecting and characterizing propagation of security weaknesses in puppet-based infrastructure management. IEEE

Transactions on Software Engineering 49, 3536–3553. doi:doi: 10.1109/TSE.2023.3265962.

Rahman, A., Parnin, C., Williams, L., 2019b. The seven sins: Security smells in infrastructure as code scripts, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 164–175. doi:doi: 10.1109/ICSE.2019.00033.

Rahman, A., Rahman, M.R., Parnin, C., Williams, L., 2021b. Security smells in ansible and chef scripts: A replication study. ACM Trans. Softw. Eng. Methodol. 30. doi:doi: 10.1145/3408897.

Rahman, A., Shamim, S.I., Bose, D.B., Pandita, R., 2023b. Security misconfigurations in open source kubernetes manifests: An empirical study. ACM Trans. Softw. Eng. Methodol. 32. doi:doi: 10.1145/3579639.

Rahman, A., Sharma, T., 2022. Lessons from research to practice on writing better quality puppet scripts, in: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 63–67. doi:doi: 10.1109/SANER53432.2022.00019.

Rahman, A., Williams, L., 2018. Characterizing defective configuration scripts used for continuous deployment, in: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), pp. 34–45. doi:doi: 10.1109/ICST.2018.00014.

Rahman, A., Williams, L., 2019. Source code properties of defective infrastructure as code scripts. Information and Software Technology 112, 148–163. doi:doi: https://doi.org/10.1016/j.infsof.2019.04.013.

Rahman, A., Williams, L., 2021. Different kind of smells: Security smells in infrastructure as code scripts. IEEE Security & Privacy 19, 33–41. doi:doi: 10.1109/MSEC.2021.3065190.

Reddy Konala, P.R., Kumar, V., Bainbridge, D., 2023. Sok: Static configuration analysis in infrastructure as code scripts, in: 2023 IEEE International Conference on Cyber Security and Resilience (CSR), pp. 281–288. doi:doi: 10.1109/CSR57506.2023.10224925.

Reis, D., Correia, F.F., 2022. Dockerlive : A live development environment for dockerfiles, in: 2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 1–4. doi:doi: 10.1109/VL/HCC53370.2022.9833145.

Reis, D., Piedade, B., Correia, F.F., Dias, J.P., Aguiar, A., 2022. Developing docker and docker-compose specifications: A developers' survey. IEEE Access 10, 2318–2329. doi:doi: 10.1109/ACCESS.2021.3137671.

Reis, S., Abreu, R., d'Amorim, M., Fortunato, D., 2023. Leveraging practitioners' feedback to improve a security linter, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/3551349.3560419.

Rodriguez, M.A., Buyya, R., 2019. Container-based cluster orchestration systems: A taxonomy and future directions. Software: Practice and Experience 49, 698–719. doi:doi: https://doi.org/10.1002/spe.2660.

Rosa, G., Mastropaolo, A., Scalabrino, S., Bavota, G., Oliveto, R., 2023. Automatically generating dockerfiles via deep learning: Challenges and promises, in: 2023 IEEE/ACM International Conference on Software and System Processes (ICSSP), pp. 1–12. doi:doi: 10.1109/ICSSP59042.2023.00011.

Rosa, G., Scalabrino, S., Oliveto, R., 2022. Assessing and improving the quality of docker artifacts, in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 592–596. doi:doi: 10.1109/ICSME55016.2022.00081.

Rosa, G., Scalabrino, S., Robles, G., Oliveto, R., 2024a. Not all dockerfile smells are the same: An empirical evaluation of hadolint writing practices by experts, in: 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR), pp. 231–241.

Rosa, G., Zappone, F., Scalabrino, S., Oliveto, R., 2024b. Fixing dockerfile smells: an empirical study. Empirical Software Engineering 29, 108.

Saavedra, N., Ferreira, J.a.F., 2023. Glitch: Automated polyglot security smell detection in infrastructure as code, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/3551349.3556945.

Saavedra, N., Ferreira, J.a.F., Mendes, A., 2025. Infrafix: Technology-agnostic repair of infrastructure as code, in: Proceedings of the 34th ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA. p. 41–45. doi:doi: 10.1145/3713081.3731735.

Saavedra, N., Goncalves, J., Henriques, M., Ferreira, J.F., Mendes, A., 2023. Polyglot Code Smell Detection for Infrastructure as Code with GLITCH , in: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE Computer Society, Los Alamitos, CA, USA. pp. 2042–2045. doi:doi: 10.1109/ASE56229.2023.00162.

Sahoo, P., Pujar, S., Nalawade, G., Gebhardt, R., Mandel, L., Buratti, L., 2024. Ansible lightspeed: A code generation service for it automation, in: 2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 2148–2158. doi:doi: 10.1145/3691620.3695277.

Schwarz, J., Steffens, A., Lichter, H., 2018. Code smells in infrastructure as code, in: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 220–228. doi:doi: 10.1109/QUATIC.2018.00040.

Shambaugh, R., Weiss, A., Guha, A., 2016. Rehearsal: a configuration verification tool for puppet, in: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, Association for Computing Machinery, New York, NY, USA. p. 416–430. doi:doi: 10.1145/2908080.2908083.

Sharma, T., Fragkoulis, M., Spinellis, D., 2016. Does your configuration code smell?, in: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 189–200.

Shi, H., Ying, L., Chen, L., Duan, H., Liu, M., Xue, Z., 2025. Dr. docker: A large-scale security measurement of docker image ecosystem, in: Proceedings of the ACM on Web Conference 2025, Association for Computing Machinery, New York, NY, USA. p. 2813–2823. URL: https://doi.org/10.1145/3696410.3714653, doi:doi: 10.1145/3696410.3714653.

Shimizu, R., Kanuka, H., 2020. Test-based least privilege discovery on cloud infrastructure as code, in: 2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 1–8. doi:doi: 10.1109/CloudCom49646.2020.00007.

Sobhani, G., Haque, I., Sharma, T., 2025. It works (only) on my machine: A study on reproducibility smells in ansible scripts, in: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), pp. 384–395. doi:doi: 10.1109/MSR66628.2025.00069.

Sokolowski, D., 2022. Infrastructure as code for dynamic deployments, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 1775–1779. doi:doi: 10.1145/3540250.3558912.

Sokolowski, D., Salvaneschi, G., 2023. Towards reliable infrastructure as code, in: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), pp. 318–321. doi:doi: 10.1109/ICSA-C57050.2023.00072.

Sokolowski, D., Spielmann, D., Salvaneschi, G., 2024a. Automated infrastructure as code program testing. IEEE Transactions on Software Engineering 50, 1585–1599. doi:doi: 10.1109/TSE.2024.3393070.

Sokolowski, D., Spielmann, D., Salvaneschi, G., 2024b. The pipr dataset of public infrastructure as code programs, in: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA. p. 498–503. doi:doi: 10.1145/3643991.3644888.

Sotiropoulos, T., Mitropoulos, D., Spinellis, D., 2020. Practical fault detection in puppet programs, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 26–37. doi:doi: 10.1145/3377811.3380384.

Spielmann, D., Sokolowski, D., Salvaneschi, G., 2023. Extensible testing for infrastructure as code, in: Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity, Association for Computing Machinery, New York, NY, USA. p. 58–60. doi:doi: 10.1145/3618305.3623607.

Sun, X., Ma, W., Gu, J.T., Ma, Z., Chajed, T., Howell, J., Lattuada, A., Padon, O., Suresh, L., Szekeres, A., Xu, T., 2024. Anvil: Verifying liveness of cluster management controllers, in: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), USENIX Association, Santa Clara, CA. pp. 649–666. URL: https://www.usenix.org/conference/osdi24/presentation/sun-xudong.

Suwanachote, N., Pornmaneerattanatri, S., Kashiwa, Y., Ichikawa, K., Leelaprute, P., Rungsawang, A., Manaskasemsak, B., Iida, H., 2023. A pilot study of testing infrastructure as code for cloud systems, in: 2023 30th Asia-Pacific Software Engineering Conference (APSEC), pp. 584–588. doi:doi: 10.1109/APSEC60848.2023.00075.

Toprani, D., Madisetti, V.K., 2025. Llm agentic workflow for automated vul-

nerability detection and remediation in infrastructure-as-code. IEEE Access 13, 69175–69181. doi:doi: 10.1109/ACCESS.2025.3560911.

Tran, A.D., Sion, L., Yskout, K., Joosen, W., 2025. Terrara: Automated security threat modeling for infrastructure as code, in: Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy, Association for Computing Machinery, New York, NY, USA. p. 269–280. URL: https://doi.org/10.1145/3714393.3726494, doi:doi: 10.1145/3714393.3726494.

Verdet, A., Hamdaqa, M., Silva, L.D., Khomh, F., 2025. Assessing the adoption of security policies by developers in terraform across different cloud providers. Empirical Software Engineering 30. doi:doi: 10.1007/s10664-024-10610-0. publisher: Springer Science and Business Media LLC.

Vo, Q.H., Dao, H., Fukuda, K., 2025. Harnessing the power of llms for code smell detection in terraform infrastructure as code, in: 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 533–542. doi:doi: 10.1109/COMPSAC65507.2025.00075.

VS, D.P., Sethuraman, S.C., Khan, M.K., 2023. Container security: precaution levels, mitigation strategies, and research perspectives. Computers & Security 135, 103490.

Wang, R., 2022. Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform.

War, A., Diallo, A., Habib, A., Klein, J., Bissyandé, T.F., 2025. Vulnerabilities in infrastructure as code: what, how many, and who? Empirical Software Engineering 30, 120.

Weiss, A., Guha, A., Brun, Y., 2017. Tortoise: Interactive system configuration repair, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 625–636. doi:doi: 10.1109/ASE.2017.8115673.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, Association for Computing Machinery, New York, NY, USA. doi:doi: 10.1145/2601248.2601268.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A., 2012. Experimentation in Software Engineering. Springer Publishing Company, Incorporated. doi:doi: https://doi.org/10.1007/978-3-662-69306-3.

Wu, Y., Zhang, Y., Wang, T., Wang, H., 2020. Characterizing the occurrence of dockerfile smells in open-source software: An empirical study. IEEE Access 8, 34127–34139. doi:doi: 10.1109/ACCESS.2020.2973750.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., Soldani, J., 2020. The essential deployment metamodel: a systematic review of deployment automation technologies. SICS Software-Intensive Cyber-Physical Systems 35, 63–75. doi:doi: 10.1007/s00450-019-00412-x.

Xiang, Y., Yang, Z., Peng, J., Bauer, H., Kon, P.T.J., Qiu, Y., Chen, A., 2025. Automated bug discovery in cloud infrastructure-as-code updates with llm agents, in: 2025 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps), pp. 20–25. doi:doi: 10.1109/AIOps66738.2025.00011.

Xu, J., Wu, Y., Lu, Z., Wang, T., 2019. Dockerfile tf smell detection based on dynamic and static analysis methods, in: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), pp. 185–190. doi:doi: 10.1109/COMPSAC.2019.00033.

Xu, Q., Gao, Y., Wei, J., 2024. An empirical study on kubernetes operator bugs, in: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA. p. 1746–1758. doi:doi: 10.1145/3650212.3680396.

Zeini, A., Lennon, R.G., Lennon, P., 2023. Securing infrastructure as code (iac) through devsecops: A comprehensive risk management framework, in: 2023 Cyber Research Conference - Ireland (Cyber-RCI), pp. 1–11. doi:doi: 10.1109/Cyber-RCI59474.2023.10671452.

Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Information and Software Technology 53, 625–637. doi:doi: https://doi.org/10.1016/j.infsof.2010.12.010. special Section: Best papers from the APSEC.

Zhang, Y., Yin, G., Wang, T., Yu, Y., Wang, H., 2018. An insight into the impact of dockerfile evolutionary trajectories on quality and latency, in: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 138–143. doi:doi: 10.1109/COMPSAC.2018.00026.

Zhou, Y., Zhan, W., Li, Z., Han, T., Chen, T., Gall, H., 2023. Drive: Dockerfile rule mining and violation detection. ACM Trans. Softw. Eng. Methodol. 33. doi:doi: 10.1145/3617173.

Zhu, Z., Chen, T., Zhong, Y., Song, Q., 2025. Dockinsight: A knowledge-augmented dependency extraction approach for dockerfile, in: 2025 IEEE/ACM 22nd International Conference on Software and Systems Reuse (ICSR), pp. 67–77. doi:doi: 10.1109/ICSR66718.2025.00013.

Özdoğan, E., Ceran, O., ÜSTÜNDAĞ, M., 2023. Systematic analysis of infrastructure as code technologies. Gazi University Journal of Science Part A: Engineering and Innovation 10. doi:doi: 10.54287/gujsa.1373305.