An Empirical Study of Policy as Code: Adoption, Purpose, and Maintenance

Ruben Opdebeeck
Vrije Universiteit Brussel
Belgium
Ruben.Denzel.Opdebeeck@vub.be

Mahmoud Alfadel
University of Calgary
Canada
mahmoud.alfadel@ucalgary.ca

Akond Rahman
Auburn University
Alabama, USA
akond@auburn.edu

Yutaro Kashiwa
Nara Institute of Science and Technology
Japan
yutaro.kashiwa@is.naist.jp

João F. Ferreira
INESC-ID and Faculty of Engineering, University of Porto
Portugal
joao@joaoff.com

Raula Gaikovina Kula
The University of Osaka
Japan
raula-k@ist.osaka-u.ac.jp

Coen De Roover
Vrije Universiteit Brussel
Belgium
Coen.De.Roover@vub.be

ABSTRACT

Policy as Code (PaC) is an emerging DevOps practice that enables teams to specify organisational and technical policies, such as regulatory compliance, security requirements, and resource limits, through machine-enforceable declarative code. As PaC gains prominence, practitioners face difficulties in adopting PaC while there remains a limited empirical understanding of how these policies are introduced, what types can be expressed, and how they are maintained in practice.

This paper aims to address this gap through an empirical study of PaC based on 10,560 PaC files from 499 open-source repositories spanning nine PaC tools. We find that PaC is introduced throughout all phases of repository lifecycles, often co-occurring with IaC tools such as Kubernetes or Terraform, with most repositories adopting one of five policy enforcement strategies. Our taxonomy of 12 policy categories reveals that while most policies govern infrastructures and security requirements, they can also express broader constraints related to software development, intellectual property, and expenses. We observe that PaC files are maintained through infrequent yet often substantial changes. Most changes concern refactoring, yet when policy behaviour does change, policies tend to become stricter rather than more lenient.

These findings motivate and support wider and earlier adoption of PaC tools. To this end, the taxonomy of policy categories can serve as a reference to practitioners to identify use cases for PaC in their projects. Meanwhile, our catalogue of enforcement strategies, co-occurring IaC tools, and PaC tool coverage of the taxonomy can inform practitioners when deciding which PaC tool to adopt and how to integrate it in their projects. Finally, our findings motivate future research to automate PaC file generation and maintenance.

KEYWORDS

configuration management, devops, policy, policy as code

ACM Reference Format:
Ruben Opdebeeck, Mahmoud Alfadel, Akond Rahman, Yutaro Kashiwa, João F. Ferreira, Raula Gaikovina Kula, and Coen De Roover. 2026. An Empirical Study of Policy as Code: Adoption, Purpose, and Maintenance. In *Proceedings of 23rd International Conference On Mining Software Repositories (MSR '26).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Today's software systems need to comply with numerous rules and regulations imposed by government organisations, such as the General Data Protection Regulation (GDPR), the Cyber Resilience Act (CRA), and the NIS2 directive of the European Union, or executive orders 14028 and 14144 of the United States. Non-compliance can lead to substantial penalties. For instance, NIS2 promises to fine organisations up to 2% of global annual revenue[1].

To meet these demands, organisations implement *policies*, which are sets of rules ensuring systems operate within certain boundaries [62]. Policies may require that data is stored within the EU to comply with GDPR, mandate the use of multifactor authentication to comply with NIS2, or disallow the use of untrusted third-party software to protect software supply chain integrity. Organisations may also adopt policies that achieve goals beyond regulatory compliance, such as limiting the financial costs of cloud instances[2].

Naturally, adherence to policies must be enforced for them to be effective. However, accelerated software release cycles realised through DevOps and Continuous Delivery practices necessitate a shift towards automating operational tasks through dedicated tooling. Infrastructure as Code (IaC) tools, for instance, have transformed deploying a cloud infrastructure from a laborious manual process into an automated routine [41]. Following in the footsteps

MSR '26, April 13–14, 2026, Rio de Janeiro, Brazil
2026. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]https://nis2directive.eu/nis2-fines/
[2]https://www.finops.org/framework/capabilities/policy-governance/

of IaC, *Policy as Code* (PaC) [62] has emerged as a practice to automatically enforce policies through executable code artefacts called *PaC files*. PaC rose to prominence in the early 2020s through "Open Policy Agent" (OPA), a general-purpose PaC tool introduced in 2016 and accepted as a graduated Cloud Native Computing Foundation (CNCF) project[3] in 2021. Domain-specific tools have emerged since then, such as Kyverno and Sentinel, which enforce policies in Kubernetes and Terraform deployments, respectively.

PaC supports shift-left security by implementing *Security as Code* and *Compliance as Code* through codified security and compliance checks, respectively [62]. A recent survey reports that out of 285 respondents, 94% agree that PaC is "*vital for preventative security and compliance at scale*", and 91% found PaC to increase productivity [76]. Nonetheless, practitioners face challenges in adopting PaC, caused by a lack of awareness and by the complexity of transforming existing policy enforcement mechanisms into PaC [76]. Furthermore, there are few academic insights into how PaC is implemented and maintained in software projects. Although a recent study investigated the policies that security scanners within Terraform repositories are configured with [80], the broader landscape of user-defined policies remains unexplored.

Addressing this knowledge gap may aid practitioners in adopting PaC, as they often prefer to learn new technologies through the experiences of others [59]. Moreover, as pointed out by Mazinanian et al. [39], such knowledge gaps negatively impact researchers, who remain unaware of the research gaps and opportunities, and tool builders, who may benefit from empirical insights to improve their tools. Combined, our limited understanding of how PaC is applied in practice inhibits future adoption of PaC, as its success depends on its ability to assist developers while reducing human effort [14]. This motivates us to conduct an empirical study into how PaC is applied in practice, in which we aim to uncover insights into when and how PaC is adopted in repositories, the types of policies that are implemented, and how PaC files are maintained.

We begin our study with a preliminary question (PQ) to explore the PaC landscape.

- **PQ: Which tools are used to implement PaC?** We identify nine popular tools by analysing Internet artefacts, including OPA, Kyverno, and Sentinel, covering a wide range of domains.

Having identified popular tools, we mine open-source software (OSS) repositories to construct a dataset of over 10,000 PaC files across 499 repositories. Using this dataset, we conduct a deeper investigation into how PaC is adopted in these repositories (**RQ1**), the purpose served by the policies (**RQ2**), and how PaC files are maintained over time (**RQ3**).

- **RQ1 (Adoption): When is PaC introduced and enforced?** We find that PaC is adopted throughout all stages of development, often co-occurring with IaC tools such as Kubernetes or Terraform. We also identify five policy enforcement strategies, such as admission-time validation of Kubernetes resources and run-time enforcement of access control policies.
- **RQ2 (Purpose): What types of policies are implemented using PaC?** We apply a qualitative analysis technique called open coding [67] on a sample of 1,944 PaC files to construct a taxonomy of 12 policy categories. Most policies relate to cloud utilities, virtualisation, and security, whereas policies related to intellectual property and expenses are rare. We find that tool coverage is scattered, with most PaC tools only supporting subsets of the policy categories.
- **RQ3 (Maintenance): How are PaC files maintained?** We observe that PaC file maintenance is infrequent but may require substantial changes. We also investigate the reasons why PaC files are changed, and find that most changes are refactorings, yet when behavioural changes occur, the policies tend to become stricter more often than more lenient.

```
package kubernetes.validating.images

deny[msg] if {
  input.request.kind.kind == "Pod"

  some container in input.request.object.spec.containers
  not startswith(container.image, "org.internal/")
  msg := sprintf("Image '%v' comes from untrusted
     registry", [container.image])
}
```

Listing 1: PaC file containing an OPA policy to validate Kubernetes container origins.

Contributions. This paper makes the following contributions:

- *Taxonomy of PaC policies.* Our taxonomy can help raise practitioner awareness by providing a set PaC use cases, and can serve as motivation for tool builders to improve PaC tools by addressing gaps in their supported coverage.
- *Reusable dataset of PaC files.* Our curated dataset of over 10,000 PaC files [5] provides a foundation for future studies of PaC.
- *Empirical insights into PaC adoption.* Our characterisation of PaC adoption can help alleviate the complexity of adopting PaC by informing practitioners on when to adopt PaC and how to enforce policies.
- *Empirical insights into PaC maintenance.* Our insights motivate future research on PaC maintenance and inform organisations of the maintenance efforts required when adopting PaC.

The dataset and analysis code for the study are available online [5].

2 POLICY AS CODE: AN EXAMPLE

PaC tools use *policy engines* to evaluate input data, typically specified as structured data (e.g., YAML or JSON), against one or more *policies* contained in a *PaC file* [62]. Each policy comprises two core aspects: its rules, often specified in a domain-specific logic programming language, and the action to take when a rule is violated. To illustrate, Listing 1 depicts a PaC file adapted from Open Policy Agent's playground[4], containing one policy that uses the Rego policy language to secure a software supply chain by validating the origins of containers deployed in a Kubernetes cluster. The policy's *rule* checks whether the input data is a "Pod" and whether some container in this pod originates from an untrusted repository. If the rule matches, the policy's *action* is to deny the request with an explanatory message.

Policy enforcement can occur at various points in time depending on the policy and target domain. The policy exemplified in Listing 1

[3]https://www.cncf.io/projects/open-policy-agent-opa/

[4]https://play.openpolicyagent.org/

is checked during deployment, i.e., when a pod is submitted to a Kubernetes cluster, and aborts the deployment upon a policy violation. Policies can also be checked prior to deployment, e.g., policies that reason about Terraform infrastructure definitions, or after deployment, such as policies that audit an already-deployed infrastructure. Finally, applications can apply policy checks in production, e.g., to implement access control.

3 DATASET CONSTRUCTION

We aim to study PaC files that are representative of the wider PaC landscape. Therefore, we first identify the most popular PaC tools (Section 3.1) and subsequently collect and curate a dataset of their PaC files found in OSS repositories (Section 3.2).

3.1 PaC Tool Identification

Motivation. As literature on PaC is scarce, a comprehensive overview of PaC tools is currently missing. Therefore, we first review Internet artefacts to identify which PaC tools are used in practice to answer the following preliminary question:

PQ: Which tools are used to implement PaC?

Approach. To identify PaC tools that are popular in practice, we first review a popular industry-oriented book on PaC [62] and identify which PaC tools are described within. However, the book may not describe all tools, nor does it provide an indication of their popularity. Therefore, we also analyse Internet artefacts (e.g., blog posts, tutorials) to identify tools that are often discussed online.

To collect Internet artefacts, we use Google's search engine in a private browsing window to avoid bias caused by browsing histories [22]. We build search queries using a search term (e.g., "policy as code") combined with keywords to focus the search to concrete tools (i.e., "tool OR technology OR language OR framework OR engine"). We start with a generic search using "policy as code" as the search term, and note the common domains in the search results (e.g., cloud vendors or infrastructure tools). Afterwards, we conduct scoped searches in which we construct search terms by combining each identified domain with "policy" or "policy as code".

For each query, we open all results from the first page and manually review their content, excluding results that are deemed of dubious quality, irrelevant, or duplicated. We also ignore Google's AI summaries to avoid misinformation caused by large language model hallucinations. From each relevant result, we extract tools whose primary purpose is PaC. To this end, we dismiss IaC tools (e.g., Terraform, Ansible) or infrastructure testing tools (e.g., TestInfra)[5]. Moreover, we omit tools that are merely utilities for another PaC tool, tools that do not support user-defined policies, or tools that focus solely on access control policies and thus fall outside the scope of our study, which focuses on PaC tools with broader application domains. We iterate through the search result pages until saturation is reached [22], i.e., when an entire page of search results does not yield new tools. We review at least two pages of search results for each query.

Finally, we count the number of unique sources that mention each tool. We identify unique sources using the domain name of the URLs, and include author names if the domain is not unique to one source (e.g., Medium, GitHub). Afterwards, we manually combine sources that represent the same organisation[6]. We also consider the industry-oriented book as an additional source.

Results. We conducted the searches on April 28–29, 2025. The generic search uncovered 22 tools mentioned by 53 sources. From these, we constructed eight additional search terms, including two domains (*Cloud* and *Infrastructure as Code*), three technologies (*Kubernetes*, *Terraform*, and *Docker*)[7], and three cloud vendors (*AWS*, *Azure*, and *Google Cloud Platform*). These additional search terms led to the inclusion of 38 new sources and four new tools. In total, we reviewed 325 (254 unique) search results, identifying 26 PaC tools mentioned in 151 pages across 91 unique sources.

We decide to focus on PaC tools that are mentioned by at least five unique sources, corresponding to nine tools. We selected this threshold as it marks a point beyond which most tools are either rarely-mentioned or nascent, therefore likely limiting their practical adoption. Table 1 provides an overview of the selected tools and the domain they target. The "**Int.**" column contains the number of unique sources by which each tool is mentioned. We do not discuss the remaining tools due to space constraints, and refer to our replication package [5] for an overview.

Open Policy Agent (OPA) is by far the most mentioned tool, likely due to its versatility and multi-domain support. We also observe that technology-specific PaC tools are mentioned often, such as HashiCorp's Sentinel, which enforces policies for Terraform IaC, and PaC tools for Kubernetes, namely Kyverno, Gatekeeper, and jsPolicy. Numerous sources also mention Checkov, a multi-domain security scanner for cloud configuration files that supports custom policies written in either Python or YAML. Vendor-specific tools that check policies for cloud infrastructures, such as Azure's PaC or AWS' CloudFormation Guard, also received substantial attention online, while Cloud Custodian, a multi-cloud alternative to these vendor-specific PaC offerings, has received slightly less attention.

We also encountered several utilities for OPA, such as `conftest`, a tool that enables practitioners to enforce policies against various types of configuration files. We do not consider these utility tools separately as their policies are written in OPA's Rego domain-specific language and are eventually evaluated by OPA itself. However, we make an exception for Gatekeeper, an OPA extension targeting Kubernetes, as Gatekeeper's Rego policies are embedded within Kubernetes manifests rather than specified in standalone files and may thus differ substantially from plain OPA policies.

PQ. *Which tools are used to implement PaC?*

We identify 26 PaC tools across 91 unique Internet sources. We focus on the nine most popular tools, covering a wide range of cloud vendors and IaC technologies.

3.2 Repository Collection and Filtering

Having identified the most popular PaC tools, we now collect a dataset of open-source repositories that use these tools. Table 1 depicts the final size of this dataset for each considered PaC tool. A

[5]Although some of these tools claim to support PaC, it is not their primary purpose.

[6]For instance, *openpolicyagent.org*, *styra.com*, and *github.com/open-policy-agent* all belong to Styra, the creators of Open Policy Agent.

[7]We also experimented with other, less-mentioned technologies, such as Ansible and Pulumi, but this did not yield new results.

Table 1: PaC dataset overview. (Int. = Internet sources)

Tool	Domain	Int.	Repos	Files	Sample
Open Policy Agent	Generic	53	225	3,614	347
HashiCorp Sentinel	Terraform	20	15	39	35
Kyverno	Kubernetes	19	125	608	236
Checkov	Generic	18	8	988	388
CloudFormation Guard	AWS	14	8	244	149
Azure Policy as Code	Azure Cloud	14	57	3,906	350
Gatekeeper	Kubernetes	13	72	852	265
jsPolicy	Kubernetes	7	1	4	4
Cloud Custodian	Multi-cloud	5	18	305	170
Total (Unique)		**78**	**499**	**10,560**	**1,944**

detailed overview of the evolving size of our dataset after each data collection step is available in our online replication package [5].

3.2.1 Finding PaC Files. We use GitHub's Code Search API[8] to find PaC files in OSS repositories. For each PaC tool, we construct a query that looks for files based on file extensions (e.g., `.rego` for OPA). For tools that use generic file formats, such as YAML or JSON, we extend the query with tool-specific keywords. To illustrate, the query for Azure PaC searches for JSON files whose contents contain "`policyRule`". Note that for Checkov, we aim to find files containing policy implementations and thus exclude repositories that only use Checkov's built-in scanning rules without defining their own, and therefore do not implement PaC. To this end, we use two separate queries as Checkov policies can be written in either Python or YAML. An overview of the queries is available in our replication package [5].

As GitHub's Code Search is limited to 1000 results and may produce unreliable results, such as inconsistent result counts across pages [74], we replicate an approach applied by prior work [74]. This approach partitions the search queries based on file size until fewer than 1000 results are returned, and retries search queries to reconcile inconsistent results. We conducted these searches on April 29, 2025, taking nearly 27h, and identified a total of 105,436 potential PaC files across 26,398 repositories.

3.2.2 Removing Forks. Our use of the GitHub Code Search API already omits forks based on GitHub metadata. Nonetheless, our dataset still contains duplicate repositories caused by forks that are not accurately reflected in the GitHub metadata, which may introduce bias into our results. Therefore, we remove these additional forks by grouping repositories based on their first commit and retaining the repositories with the most stars. This removes 994 repositories and 14,483 files from the dataset.

3.2.3 Validating Files. We automatically validate each remaining file to remove incorrectly identified files, which can occur when file extensions are used by unrelated tools or keywords are incorrectly matched by the query. We also aim to remove files that, although related to PaC tools, do not contain policies themselves (e.g., a Kubernetes deployment for a policy engine). To this end, we implement validators for each PaC tool.

For Open Policy Agent, Sentinel, and CloudFormation Guard, we parse files using the tools' official parsers and consider the file invalid if parsing fails. We also account for special cases where valid PaC files fail to parse, which occurs in several security scanning tool repositories that use OPA policies that are parametrised using a template language. For Azure PaC, Cloud Custodian, and Checkov YAML policies, we check whether the JSON or YAML data contains the elements that are required according to the tools' documentation. For Checkov policies written in Python, we check whether the file imports Checkov's abstractions for custom rules. Finally, for tools that use Kubernetes manifests, i.e., Kyverno, Gatekeeper, and jsPolicy, we validate whether the Kubernetes manifest contains a policy definition for the respective tool by checking the `apiVersion` and `kind` fields.

This validation removed 21,255 repositories and 48,775 files from our dataset. The vast majority of these were removed due to the Cloud Custodian validation, as the GitHub Code Search query was overly general for this tool and returned many invalid results.

Manual Validation. As the automated validation above can suffer from false positives and false negatives, we manually review a random sample of the retained and rejected files for each tool. We determine statistically significant sample sizes for each sample using Cochran's formula adjusted for small populations [8], using a 95% confidence, 5% margin of error, and 50% estimated proportion. Then, one author performed a lightweight manual review of each file in each sample. For retained files, they checked whether the file contained a policy for the tool in question, whereas for rejected files, they checked whether the files were either not related to the PaC tool, contained syntax errors, or lacked policy definitions.

We assessed a total of 1,923 rejected and 2,890 retained files. Among these, we identified 27 incorrectly retained files which were related to PaC tools but did not contain policies, and only three incorrectly rejected files that are valid and contain policies. We also observe that the validation was effective at removing files that relate to PaC tools but do not contain policies. In summary, we find that the validation achieves a remarkably low error rate and effectively safeguards the quality of the dataset.

3.2.4 Filtering Repositories. The previous steps aimed to identify all repositories on GitHub that use one of the PaC tools, but some repositories may be of low quality. To remove such repositories, we apply quality criteria suggested by prior work [1, 29]. Specifically, we aim to retain (i) *maintained, non-toy repositories* based on the commit count and age of the repository; (ii) *collaborative, non-personal repositories* based on contributor, issue, and pull request counts; and (iii) *practical repositories* by omitting demo or example repositories. However, as this is the first study on PaC, we cannot rely on prior work to establish thresholds as they may not be applicable to the PaC domain. After experimenting with different thresholds, we determined that using the medians of each metric as a minimum threshold is a simple yet effective approach. Therefore, we remove repositories of which any metric falls under the median value. We calculate the thresholds separately for each tool to account for differences (e.g., age, popularity) between tools. The exact thresholds used can be found in our online replication package [5].

Applying these criteria removes 3,523 repositories containing 29,125 files from our dataset. Specifically, 2,099 repositories were removed because they contain too few commits, 454 repositories do not pass the age threshold, and, 415, 465, and 30 repositories were

[8]https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28#search-code

removed because of too few contributors, issues, and pull requests, respectively. Moreover, we omit 57 repositories because their names contain "example", "playground", "template", "sample", or "demo", and three repositories that became unavailable after they were first identified. Finally, from the remaining repositories, we eliminate any file whose path contains "test" (2,073 files) or "example" (420 files) to remove PaC files that are not used in practice. This removes a further 127 repositories that no longer contain PaC files.

Our final dataset therefore comprises 10,560 PaC files across 499 repositories, as described in the "**Repos**" and "**Files**" columns of Table 1. Note that a repository may use multiple PaC tools, causing the totals to be less than the sum of the values of individual tools.

3.2.5 Cloning Repositories. We clone all 499 repositories and check out each repository to the latest commit at the time the repository was added to the dataset. This ensures that for subsequent analyses the PaC files are in the state they were in when originally identified. We successfully cloned all repositories, comprising 44 GB of compressed data.

3.2.6 Sampling PaC Files. We create a stratified random sample of PaC files that will be manually analysed in RQ2. We determine sample sizes for each PaC tool separately using Cochran's formula adjusted for small populations [8] with a 95% confidence, 5% margin of error and 50% population proportion. Column "**Sample**" of Table 1 depicts the resulting sample sizes.

3.2.7 Dataset Exploration. Finally, we compute the proportion of PaC files in each repository and find that they represent a median of 0.65% of the files in repositories, ranging from 0.1% (jsPolicy) to 4.8% (Cloud Custodian) per tool. Nonetheless, several repositories contain reusable PaC files and thus form outliers, with 30% to 77% of the repositories' files being PaC files. In terms of size, we find that PaC files contain a median of 45 non-empty lines of code, with 78.4% of files containing less than 100 non-empty lines of code.

4 EMPIRICAL STUDY

Having collected our dataset, we now empirically explore the use of PaC in OSS repositories. The diversity of tools highlighted in Section 3.1 prompts us to investigate the characteristics of PaC adoption (**RQ1**) and the types of policies that PaC files implement (**RQ2**). Moreover, Section 3.2.7 shows that most repositories contain a low proportion of PaC files, leading to further questions on how PaC files are maintained over time (**RQ3**). This section describes the motivation for, the approach to answer, and the findings of each RQ.

4.1 RQ1 (Adoption): When is PaC introduced and enforced?

Motivation. Characterising when PaC is introduced and how policies are enforced in repositories can provide insights to practitioners about adopting PaC. We focus on three PaC adoption aspects:

- Identifying **the development stages at which PaC is first introduced** can inform practitioners about when to adopt PaC in their own repositories.
- Understanding **which strategies are used to enforce policies**, e.g., CI pipeline jobs or run-time checks, provides insights into how PaC can be integrated into development workflows.
- Uncovering frequently **co-occurring IaC tools** can aid practitioners in decision-making when choosing PaC tools that fit within their existing infrastructure deployment processes.

Approach. To analyse when PaC is first adopted in a repository, we traverse the commit history of each of the 499 repositories in search for the oldest commit that created any of the PaC files identified in Section 3.2. We compute the proportion of commits that occurred before the introduction of the first PaC file to enable comparison across repositories with different lifespans.

To investigate which strategies are used to enforce policies, we systematically examine the code and documentation of a sample of repositories for evidence of policy enforcement. As this is a labour-intensive process, we adopt a saturation sampling approach [21] in which we first explore 50 random repositories. In each subsequent iteration, we explore 20 more repositories until no new strategies emerge and saturation is reached. We focus on artefacts that are typically associated with policy enforcement, such as CI pipeline files, Kubernetes admission controllers, deployment scripts, and production application logic. Initially, one co-author of the paper with nine years of programming experience applied multi-label classification using open card sorting [6], allowing enforcement strategies to emerge during labelling. In total, the rater labelled 70 repositories, achieving saturation after the second iteration. The obtained strategies were discussed with a second co-author with 13 years of programming experience, who then independently labelled the sample of repositories anew. The raters agreed totally on 72.9% of the repositories and partially (i.e., sharing at least one label) on 10%. To assess the reliability of our labelling, we compute inter-rater agreement using Cohen's Kappa [9] by treating each unique label combination as a category, and obtain a value of 0.66, indicating "substantial" agreement [33]. Finally, the two raters resolve all disagreements through discussion.

To investigate which IaC tools co-occur with PaC usage in repositories, we scan the latest version of all 499 repositories for the presence of files associated with IaC tools. We focus on widely used tools across four categories [23, 74]: *provisioning* (Terraform, Pulumi, AWS CloudFormation, AWS CDK, CDKTF, Azure Resource Manager, Bicep, Vagrant), *configuration* (Ansible, Chef, Puppet, Salt), *orchestration* (Kubernetes), and *templating* (Packer). We identify these files through naming conventions (e.g., `.tf` files for Terraform) and keyword or patterns within file contents (e.g., `apiVersion` for Kubernetes manifests). To ensure accuracy, we manually inspect a sample of matched files for each IaC tool to validate correct identification. We also review a sample of repositories in which no IaC tool was detected to ensure that tools were not missed due to limitations in the patterns. We found no evidence of either case.

Results. **Observation 1. PaC is introduced throughout all stages of repository lifecycles.** The distribution depicted in Figure 1 shows that only 28% of repositories adopt PaC early (i.e., within the first 20% of development). Across all repositories, PaC files are introduced after a median of 43.4% of development has elapsed. Introduction times vary across PaC tools, with Sentinel

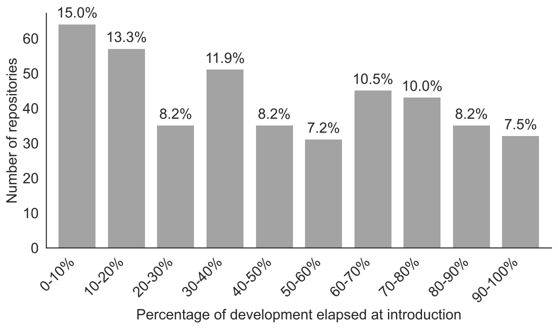

Figure 1: Distribution of PaC introduction time.

Table 2: Identified policy enforcement strategies.

ID	Enforcement strategy & Description	(%)
ES1	**Admission Time.** Policies are enforced during Kubernetes admission through admission controllers.	34.29%
ES2	**User-Invoked.** Policy enforcement is invoked manually by developers or operators.	22.86%
ES3	**Continuous Integration.** PaC checks are automatically executed as part of CI pipelines (e.g., GitHub Actions).	21.43%
ES4	**Event-Triggered.** Enforcement occurs in response to runtime events.	20%
ES5	**Scheduled.** Policies are enforced periodically via cron jobs or scheduled scripts.	5.71%

tending to be adopted the earliest at a median of 14.5% of development, whereas CloudFormation Guard is adopted later at a median of 67.4% of development. However, as PaC has only recently gained in popularity, it may have been impossible for older repositories to have adopted PaC earlier. Nonetheless, these findings suggest that PaC can be adopted at any stage of development, both by emerging and mature repositories.

Observation 2. Most repositories use one of five enforcement strategies. Table 2 summarises the identified strategies. Note that the total frequency exceeds 100% as a repository may use multiple enforcement strategies. We also encountered four repositories in which we found no evidence of policy enforcement, e.g., because the PaC files formed test data and did not follow naming conventions that led to their removal from the dataset (cf. Section 3.2.4). We describe each enforcement strategy below. Note that the reported frequency is limited to the sample of examined repositories.

ES1. Admission Time (34.29% of sampled repositories). The first enforcement strategy validates policies at Kubernetes admission time, typically using tools such as Kyverno, Gatekeeper, or custom admission webhooks, to evaluate resource manifests as they are applied to the cluster. For example, `toboshii/home-ops`[9] integrates Kyverno policies to reject Kubernetes resources that violate security or resource allocation constraints.

ES2. User-Invoked (22.86%). In this strategy, policy checks need to be manually integrated or triggered by developers or operators. Most of the repositories using this strategy offer reusable catalogues of policies, such as policies that are checked as part of code analysis tools. For example, `conforma/policy`[10] provides a set of OPA policies for a command-line tool to verify software integrity.

ES3. Continuous Integration (21.43%). Repositories using this strategy contain dedicated CI pipelines to enforce policies on infrastructure or configuration files (e.g., Terraform manifests) at every commit or pull request. For example, the GitHub Actions workflows contained in `govuk-one-login/authentication-api`[11] validate Terraform files using custom Checkov policies at every commit.

ES4. Event-Triggered (20%). This strategy enforces policies in response to specific runtime events, such as API calls or resource modifications, and is typically observed in the context of access control. For instance, `shiblon/entroq`[12] integrates OPA policies to perform request-level authorisation at runtime.

ES5. Scheduled (5.71%). The final strategy enforces policies on a recurring schedule through cron jobs or scheduled CI pipelines, independent of external events. This may be useful in compliance-heavy repositories to detect misconfigurations over time. For instance, `mitodl/ol-infrastructure`[13] runs Cloud Custodian checks on an AWS infrastructure every 24 hours.

Observation 3. PaC tools often co-occur with orchestration and provisioning IaC tools, but less with configuration or templating IaC tools. We identify 402 repositories that use an IaC tool alongside a PaC tool. The most frequent co-occurrences are Kubernetes (315 repositories) and Terraform (141). Many of the pairings are ecosystem-specific, such as AWS' CloudFormation and CloudFormation Guard, HashiCorp's Terraform and Sentinel, and Azure's Resource Manager or Bicep with Azure PaC. Moreover, Kubernetes-specific PaC tools (Kyverno, Gatekeeper, and jsPolicy) appear exclusively in Kubernetes-based repositories. These patterns suggest that PaC tools are adopted to complement their corresponding IaC tools.

Overall, while we find that PaC frequently co-occurs with orchestration and provisioning tools, such as Kubernetes and Terraform, we observe substantially fewer uses of PaC with configuration or templating tools such as Ansible or Packer. Although we find 50 repositories that use Ansible alongside a PaC tool, we note that co-occurrence does not always indicate that PaC policies relate to the IaC definitions. Indeed, 29 repositories combine Ansible with Kyverno, which targets Kubernetes resources rather than Ansible scripts, and 14 repositories pair Ansible with OPA, but a manual inspection shows that the policies rarely target the Ansible configuration. Moreover, Packer appears in only 11 repositories, Salt in two, Puppet in one, and Chef in none. Although tools like `conftest` and `ansible-policy` support PaC-based validation for such types of IaC tools, we find little evidence of their use in practice.

[9]https://github.com/toboshii/home-ops

[10]https://github.com/conforma/policy

[11]https://github.com/govuk-one-login/authentication-api

[12]https://github.com/shiblon/entroq

[13]https://github.com/mitodl/ol-infrastructure

Table 3: Example of the open coding process.

Policy Description	Object	Category
"Ensure that Cloud KMS cryptokeys are not anonymously or publicly accessible"	crypto	Security
"Produce, control and distribute symmetric cryptographic keys"	crypto-key	Security
"Ensure IAM password policy requires minimum length of 14 or greater"	password	Security

RQ1. *When is PaC introduced and enforced?*

PaC has been introduced throughout all stages of development, often co-occurring with orchestration or provisioning IaC tools. Repositories use varying policy enforcement strategies, including at Kubernetes admission time, in CI pipelines, invoked manually by users, triggered by events, and on recurring schedules.

4.2 RQ2 (Purpose): What types of policies are implemented using PaC?

Motivation. While RQ1 explored the strategies used to integrate policy enforcement in repositories, this RQ investigates *which types* of policies are enforced by PaC files. An overview of the types of policies present in real-world repositories provides insights into the nature of policies specified by developers and the reasons why PaC is used. We investigate two aspects:

- A **taxonomy of policy categories** based on the types of resources they govern can inform practitioners and researchers about the types of policies that are adopted in practice.
- The **tool coverage of the taxonomy's categories** quantifies the capabilities of PaC tools, enabling informed tool choices for practitioners and highlighting potential gaps in tool capabilities for researchers and tool builders.

Approach. We construct a taxonomy of policies using a qualitative coding process. To ensure reliability and reproducibility, we apply multiphase open coding [77], which consists of two rounds of open coding [27, 51, 77] conducted by two co-authors, each with over ten years of experience in software engineering qualitative analysis. We extract the textual descriptions of all policies contained in the 1,944 sampled PaC files (cf. Section 3.2.6), yielding 3,957 policy descriptions that we analyse in two phases.

Synchronised Open Coding. In the first round, we apply synchronised open coding to the first 1,979 policy descriptions. The two raters independently review each entry to assess (i) whether it defines a constraint on a specific object (e.g., cloud service); (ii) the type of object being targeted; and (iii) thematic similarities across object types to form broader categories. Table 3 illustrates this process. From the descriptions in the first column, we identify three objects, namely "*crypto*", "*crypto-key*", and "*password*", which we grouped under the "*Security*" category. The two raters held weekly discussions over two months to reconcile their findings and refine the emerging categories. This phase achieved a Cohen's Kappa [9] of 0.95, indicating "almost perfect" agreement [33]. Disagreements on 97 entries were resolved through discussion, resulting in a preliminary taxonomy of 11 policy categories.

Independent Open Coding. In the second round, we analyse the remaining 1,978 policy descriptions using independent open coding. In this phase, each rater labels the entries individually, without discussion or coordination, using the criteria from the previous phase. This round yields a Cohen's Kappa of 0.65, which corresponds to "substantial agreement" [33]. Disagreements on 344 entries were resolved through post-coding discussion. This phase also identified one additional category not captured in the synchronised round, resulting in a final taxonomy of 12 categories.

In all, after the two rounds of open coding the raters mapped 2,251 of the 3,957 descriptions to a category. The remaining 1,706 entries were too generic (e.g., "*check*") to be mapped to a category.

Tool Coverage. To complement the taxonomy, we calculate the extent to which each PaC tool covers the categories. For each tool-category pair, we use the formula below to compute the proportion of the category's policies that are implemented using the tool.

$$\text{Coverage}(t, c) = 100\% \times \frac{\#\text{ policies in } c \text{ implemented using } t}{\text{Total policies in } c}$$

Results. **Observation 4. PaC policies are primarily concerned with infrastructures and security.** Table 4 depicts our taxonomy, showing that three categories, i.e., *Cloud Util* (40.38%), *Virtualisation* (24.97%), and *Security* (11.37%), account for 76.7% of all policies. *Cloud Util* includes policies for resource tagging, instance provisioning, and serverless functions, *Virtualisation* concerns containers, virtual machines, and their orchestration, and policies in the *Security* category enforce requirements on cryptography and access control. Other policies target operational and development needs, such as policies related to *Monitoring*, *SQL* database configurations, *Networking*, and *Software Development*. We also encounter several rare categories, such as policies that limit cloud expenses, govern intellectual property rights, or validate policy metadata itself.

Observation 5. Tool support for policy categories varies widely. Figure 2 presents a heatmap of tool coverage per category. Tools such as Sentinel and Azure PaC offer broad support, covering eight or more categories. In contrast, tools such as Gatekeeper or Cloud Custodian offer partial or no support for less common categories, such as SQL or Intellectual Property. Note that jsPolicy is excluded, as none of its policies could be categorised using our method.

RQ2. *What types of policies are implemented using PaC?*

We identify 12 categories of PaC policies. While most policies target infrastructure (e.g., containers, compute instances) and security, a small fraction addresses broader needs, such as cost limits, and IP compliance. Tool support is uneven across categories, with stronger coverage for infrastructure-related policies.

4.3 RQ3 (Maintenance): How are PaC files maintained?

Motivation. We aim to understand how PaC files are maintained after PaC is adopted in real-world repositories. Our analysis focuses on three key aspects of maintenance:

- Studying the **frequency and magnitude** of PaC file changes enables practitioners to estimate the maintenance efforts required.
- Identifying **who maintains PaC files** helps understand whether PaC maintenance requires a small group of experts or involves most developers in a repository.

Table 4: Taxonomy of identified policy categories.

Category (%)	Definition and Sub-categories	Example(s)
— Infrastructure and Security Policies		
Cloud Util (40.38%)	Policies for cloud resources and utilities. **Sub-categories:** Instance, Serverless, Tagging	"Require compute instances use allowed machine types." "Check if X-Ray is enabled for Lambda." "Ignore specific tag values during evaluation."
Virtualisation (24.96%)	Policies for virtualisation and containerisation. **Sub-categories:** Container, Orchestration, VMs	"Remediate privileged container violations." "Restrict Kubernetes deployments without namespaces." "Enforce VM disk usage limits."
Security (11.37%)	Policies for confidentiality, integrity, and availability.	"Ensure admin users are not tied to API keys."
— Operational and Development Policies		
Monitoring (5.73%)	Policies for logging and alerts on resource health.	"Log an alert if OSTBytesAvailable is below 15%."
Software Dev (5.64%)	Policies for software development processes. **Sub-categories:** Dependencies, PRs	"Use HTTP import to list registry modules." "Require two approvals for PRs."
SQL (5.61%)	Policies for SQL database configurations.	"Enable geo-redundant backups for PostgreSQL."
Networking (5.51%)	Policies on network settings and traffic control.	"Disable IP forwarding on network interfaces."
— Rare Policy Categories (<1%)		
Jobs (0.31%)	Policies for scheduled or batch jobs.	"Schedule must use valid Cron syntax."
Config Scripts (0.31%)	Policies for validating IaC scripts.	"Use tfplan import to enforce Terraform version."
Metadata (0.09%)	Policies about the metadata of the policy itself.	"The display name must be under 128 characters."
Intellectual Property (0.04%)	Policies enforcing IP ownership and usage rights.	"Require compliance with intellectual property rights."
Expense (0.04%)	Policies limiting cloud expenses.	"Monthly cost must be below $100 for dev team."

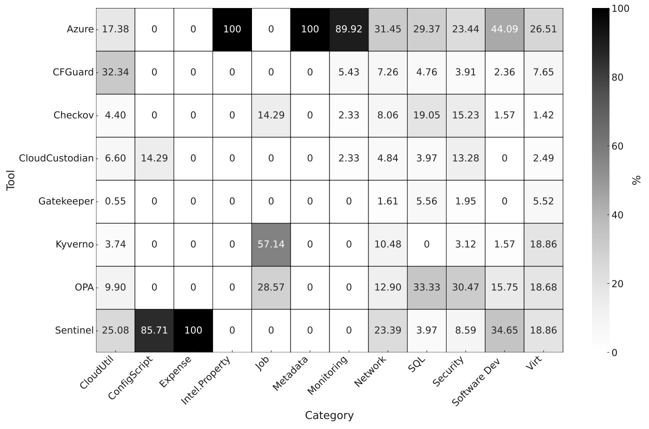

Figure 2: Tool coverage for policy categories. Each cell shows the proportion of policies in that category implemented using the corresponding tool.

- Analysing the **reasons why PaC files are modified** offers insights into the goals of PaC maintenance, e.g., whether policies are made stricter or more lenient.

Approach. We traverse the Git history of all 499 repositories in the dataset to collect commits containing changes to PaC files present in the latest revision of the repository, which we refer to as *PaC commits.* As we are interested in PaC maintenance, we focus on commits that modify existing PaC files, not those that create or delete PaC files. We also exclude merge commits, as they may aggregate changes from other commits, including those unrelated to PaC. This results in 6,103 PaC commits, which we subject to both quantitative and qualitative analyses.

Quantitative analysis. As a proxy for maintenance effort, we measure the frequency of PaC commits relative to all commits created after PaC was introduced into repositories. We also measure the number of lines changed in PaC and non-PaC files within PaC commits to better understand the magnitude of PaC-related changes. Finally, we calculate the proportion of developers involved in PaC maintenance, relative to the number of developers who have contributed to the repository after PaC was introduced, to understand how PaC maintenance tasks are distributed within development teams.

Qualitative analysis. To investigate the motivation for PaC maintenance activities, we conduct a qualitative analysis of the changes and messages contained in PaC commits. We construct a sample of 363 PaC commits using Cochran's formula with 95% confidence and 5% margin of error [8]. Then, following a similar approach to RQ2, two authors conduct two rounds of open coding. In the first round, they apply open coding to 50 commits using multi-label classification, after which they reconcile their findings and refine the labels to establish a shared understanding [38]. In the second round, the same raters analyse the remaining 313 commits individually, without discussion or coordination. They agreed totally on 78.3% of the second round's cases and partially on 1.9% of the cases (i.e., they shared at least one label, but each also included a label the other did not), demonstrating a substantial agreement. Cohen's Kappa, computed by sorting and concatenating labels into unique categories, was 0.66, indicating "substantial agreement" [33]. To resolve the disagreements, a third author checked the conflicting cases and the corresponding annotations, and then recommended a final annotation. The recommendations were discussed with the first two authors until full agreement was reached. The three annotators have 17, 20, and 13 years of programming experience, respectively.

Results. **Observation 6. In most repositories, PaC changes constitute only a minimal fraction of overall development activity.** Figure 3 shows the distribution of PaC maintenance frequency across the analysed repositories, reflecting the percentage of commits that modify PaC files relative to the total number of commits created after PaC was introduced in each repository. It demonstrates a pronounced right-skewed pattern with a sharp peak near zero. The majority of repositories exhibit very low PaC maintenance, with the median of approximately 2.5% of total commits.

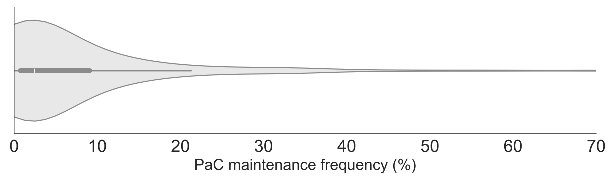

Figure 3: Distribution of PaC maintenance frequency.

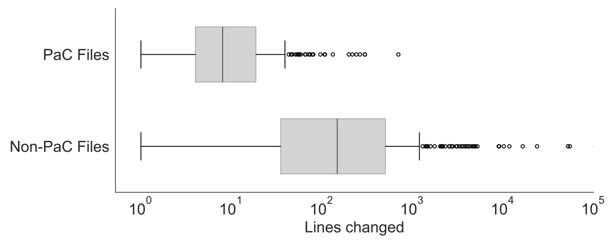

Figure 4: Distribution of lines modified by PaC commits in PaC and non-PaC files.

Nonetheless, the long tail, reaching up to 70%, reveals the existence of outliers where PaC maintenance constitutes a substantial portion of development activity. We observe that many of these outliers are repositories in which PaC files represent an above-average proportion of all files (cf. Section 3.2.7). Specifically, in 14 of the 26 repositories in which PaC maintenance frequency exceeds 25%, PaC files represent over 10% of all files. The high proportion of PaC files and PaC maintenance suggests these repositories may be compliance-heavy or provide catalogues of reusable policies.

Observation 7. PaC commits seem to involve substantial changes to both PaC and non-PaC code. Figure 4 depicts the number of lines changed by PaC commits, comparing between lines changed in PaC files and non-PaC files (e.g., production or infrastructure code). It shows that PaC commits involve substantial modifications to both PaC and non-PaC files. PaC commits seem to modify non-PaC files significantly more than PaC files, as the median number of lines changed in non-PaC files is 148, an order of magnitude higher than in PaC files with a median of around 8 lines. This suggests that PaC files are often changed as part of larger maintenance activities, possibly with surrounding code being modified in response to changing policies, or vice versa.

Observation 8. PaC maintenance is typically concentrated among a subset of developers within each repository. Figure 5 shows the distribution of the proportion of a repository's developers that have modified PaC files. The distribution is highly right-skewed with a median of 25%, demonstrating that in most repositories, relatively few developers engage in PaC maintenance. While the distribution extends from near 0% to 100%, the bulk of repositories cluster at the lower end, with the majority having less than half of their developers involved in PaC-related work.

Observation 9. Most PaC changes involve refactorings aimed at improving structure without altering behaviour. When behaviour does change, policies tend to become stricter more often than more lenient. We identified seven main reasons behind PaC file modifications: *Refactoring* (30.9% of commits, to improve structure without altering behaviour); *Stricter policy* (21.2%, to tighten constraints or add checks); *Bugfix* (17.4%, to correct faulty or unintended behaviour); *Documentation or metadata changes* (16.3%, to update auxiliary information without affecting enforcement); *More relaxed policy* (16.3%, to loosen constraints or remove checks); *Dependencies/builds* (8.3%, to update tooling/versions or integration setup); and *Performance* (0.6%, to improve policy efficiency). Bug fixes were identified when the change clearly fixed an issue (e.g., a syntax error) or the commit message explicitly indicated a bug fix. Only 2.2% of bug fixes also made the policy stricter, and 0.6% made it more relaxed. An example of a *Stricter policy* change is shown in Listing 2, in which the highlighted condition was added to also raise a violation if a virtual machine's Linux-specific configuration does not explicitly disable password authentication. This change reflects a stricter security requirement, preventing scenarios where password authentication remains possible even if SSH password settings appear compliant at a higher level.

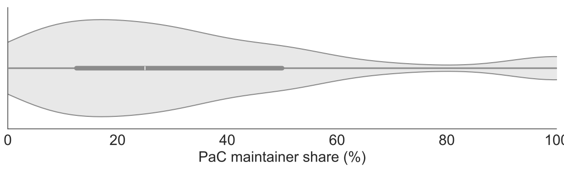

Figure 5: Distribution of the proportion of a repository's developers that modified PaC files.

```
CxPolicy[result] {
   vm := input.document[i].playbooks[k].azure_rm_virtualmachine
   is_linux_vm(vm)
   not vm.ssh_password_enabled == false
+  not vm.linux_config.disable_password_authentication == false
   result := {
      "documentId": input.document[i].id,
      ...,
      "keyExpectedValue": sprintf("'azure_rm_virtualmachine[%s]'
     should be using SSH keys for authentication", [vm.name]),
      "keyActualValue": sprintf("'azure_rm_virtualmachine[%s]' is
     using username and password for authentication", [vm.name]),
   }
}
```

Listing 2: Example of a *Stricter policy* change[14].

RQ3. *How are PaC files maintained?*

PaC files are maintained through relatively infrequent yet often substantial code changes, primarily carried out by a subset of developers. Maintenance activities predominantly involve refactoring to improve readability and maintainability; however, when behavioural changes occur, policies tend to become stricter rather than more lenient.

5 DISCUSSION

This section describes the implications and limitations of our study.

[14]https://github.com/Checkmarx/kics/commit/49018cb

5.1 Implications

We first discuss the implications of our findings for practitioners, researchers, and PaC tool builders.

5.1.1 Implications for Practitioners. Our findings motivate **earlier and wider adoption of PaC in practice**. Observation 1 shows that PaC can be integrated at any point in development, while Observations 6 and 8 suggest that PaC files require minimal maintenance by few developers once introduced. Therefore, emerging projects may benefit by adopting PaC to enforce compliance and security best practices early on, while maturing projects could implement PaC to tackle growing operational complexity.

Our results may also **help practitioners who are adopting PaC**. The taxonomy of policy categories constructed in RQ2 provides a set of use cases to consider for a project, ranging from common categories related to infrastructure and security to rare categories, such as intellectual property and expenses (Observation 4). Observation 2 furthermore provides several enforcement strategies that practitioners can replicate. Our insights can also serve as a reference point for practitioners to assess their PaC usage against the state of the practice, e.g., by comparing their maintenance frequency or the types of policies implemented. Finally, our results can support practitioners in deciding which PaC tools to adopt, with Observation 3 highlighting PaC and IaC tools that are commonly used together, and RQ2 mapping tool support for each policy category. For instance, Observation 5 shows that Microsoft Azure's PaC solution may be an attractive option for organisations that wish to cover a wide range of policy categories.

5.1.2 Implications for Researchers. The mapping of policy descriptions to categories constructed for RQ2 provides the foundation for future research on **automated policy generation**. For example, researchers could use the mapping to construct generative AI models to automatically create policies for a given tool and category. This mapping can also be extended into a benchmark to evaluate models on their policy generation capability. Moreover, Observations 7 and 9 revealed that changes to PaC files may impact a large amount of code, with most changes involving refactorings. Researchers could use our labelled dataset of PaC changes to further **study and automate refactoring operations** on PaC files.

5.1.3 Implications for Tool Builders. Our results reveal several **gaps in the capabilities of current PaC tools**. As shown in Observation 3, while infrastructure provisioning and orchestration appear to be widely covered by PaC tools, we find little evidence of PaC files that enforce best practices in configuration management or server templating scripts. Observation 5 also reveals gaps in tool coverage of the different policy categories. These findings may motivate PaC tool builders to expand the capabilities of their tools. Finally, our taxonomy may serve to motivate new PaC tools covering domains not contained in the taxonomy, such as (generative) AI models.

5.2 Threats to Validity

We summarise the threats to validity of our findings as follows.

Conclusion Validity. The results of our manual analyses are limited to the sampled PaC files and repositories, and may be influenced by the omission of other PaC files. We mitigated this threat by using representative samples, determined using Cochran's formula or saturation sampling. Furthermore, as policy descriptions may not always suffice to determine a policy's category, we may have missed categories that only become evident at run-time.

Construct Validity. Our data is mined from open-source repositories, which may raise quality concerns. We mitigated this threat by applying established filtering criteria to retain high-quality repositories. Furthermore, the use of automated tools to identify PaC and IaC files may lead to misclassifications, which we mitigated by conducting a comprehensive manual review of the classifications. Moreover, manual labelling during the qualitative analyses may introduce subjective bias, which we mitigated by using multiple labellers and calculating inter-rater agreement scores. Finally, we acknowledge that the presence of repositories offering reusable catalogues of policies (cf. Section 3.2.7) may influence our findings. However, we explicitly discuss such repositories in RQ1 and RQ3, whereas in RQ2, we argue that they help to provide a broader perspective of policy purposes.

Internal Validity. For RQ3, we assume that commits represent distinct development activities. Nevertheless, it is possible that commits mix PaC and non-PaC development, which we mitigated by omitting known problematic commits (e.g., merge commits). Moreover, during our subsequent manual commit analysis, we found no evidence of systematic problems that could invalidate our findings. Similarly, in RQ1, the co-occurrence of an IaC and a PaC tool may not imply causality, which we mitigated by reviewing dubious cases.

External Validity. The results presented in this paper are obtained from open-source repositories on GitHub and may not generalise to proprietary PaC projects. Furthermore, our findings are limited to PaC tools that are often mentioned online and may not generalise to other PaC tools not considered.

6 RELATED WORK

Infrastructure as Code. PaC is closely related to IaC, which has been studied extensively in recent years [7, 53]. Researchers have studied several aspects of IaC development, including maintainability [10, 13, 31, 43, 46, 68, 70, 79], defect prediction [11, 12, 50, 58], testing [57, 73], and dependencies [42, 45]. Prior work has also empirically studied the occurrence of defects [17, 26, 52, 57, 84] and proposed approaches to detect them [24, 28, 69, 75]. Numerous studies have also investigated security in IaC scripts, from taxonomies of security weaknesses for various IaC tools [54–56] to the identification of new types of weaknesses [32, 34, 63] and improvements to weakness detection techniques [44, 65, 66].

While IaC has been studied extensively, we observe a lack of research studying PaC due to its nascency. IaC-related findings are also not directly applicable to PaC, which governs various operational aspects, including, but not limited to, infrastructures. The only existing research related to PaC is by Verdet et al. [80], who derived a taxonomy of security policies offered by built-in checks of two security scanning tools, including Checkov, and studied their adoption in Terraform repositories. In contrast, our paper derives a more general taxonomy of policies beyond security and studies their adoption and maintenance across the wider PaC domain, covering nine popular PaC tools.

Policy Analysis in Software Engineering. As policies have numerous applications in software engineering, researchers have proposed several techniques to analyse them. Examples include techniques to assess adherence to policies, particularly privacy policies, in mobile apps [3, 4, 15, 37, 72, 82, 86], mini apps [35], and voice assistants [36, 83], as well as policy generation [85], analysis [30, 71], and visualisation [48]. Others have synthesised policy-related guidelines for practitioners [25, 47, 81]. Researchers have also proposed detection and repair techniques for access control policy violations in cloud-based services [16, 18, 19]. Our study differs from this prior research, as we focus on PaC, a practice to implement and enforce policies through codified checks, rather than policies in general.

DevOps. Researchers have conducted numerous studies on DevOps, concerning aspects such as leadership [49], adoption barriers [78], implementation-related challenges [64], quality assurance [2, 40], and security best practices [60, 61]. While organisations report positive experiences with DevOps, there is a notable lack of quantitative data supporting these claims [20]. As PaC is an under-explored yet important DevOps practice, our study of its use in practice contributes to addressing this knowledge gap.

7 CONCLUSION

This paper presented an empirical study of Policy as Code (PaC), an emerging DevOps practice to enforce policies using declarative code. We collected a dataset of over 10,000 PaC files from 499 open-source repositories spanning nine PaC tools. We found that PaC is introduced at all stages of repository lifecycles, and observed five enforcement strategies used in practice. From a qualitative analysis of 1,944 PaC files, we constructed a taxonomy of 12 policy categories, ranging from infrastructure and security constraints to intellectual property and expense requirements. By analysing the change histories of all repositories, we found that PaC file maintenance is infrequent, but that changes may be substantial, with most changes involving refactoring. Our findings motivate widespread and earlier adoption of PaC. Supporting this, our taxonomy of policy categories, catalogue of enforcement strategies, and insights into PaC tool capabilities can inform practitioners on how to integrate PaC in their projects.

ACKNOWLEDGMENTS

ruben ▹ *TODO: Credit Shonan.* ◃ Ruben Opdebeeck is funded by the "Cybersecurity Research Programme Flanders". Akond Rahman was partially funded by the U.S. National Science Foundation (NSF) Award # 2312321.

REFERENCES

[1] Amritanshu Agrawal, Akond Rahman, Rahul Krishna, Alexander Sobran, and Tim Menzies. 2018. We don't need another hero? the impact of "heroes" on software development. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 245–253. https://doi.org/10.1145/3183519.3183549

[2] Ahmad Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, and Antonio Filieri. 2021. Quality-Aware DevOps Research: Where Do We Stand? *IEEE Access* 9 (2021), 44476–44489. https://doi.org/10.1109/ACCESS.2021.3064867

[3] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. 2019. PolicyLint: investigating internal privacy policy contradictions on google play. In *28th USENIX security symposium (USENIX security 19)*. 585–602.

[4] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. 2020. Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*. 985–1002.

[5] Anonymous Authors. 2025. Verifiability Package for Paper. https://figshare.com/s/3ef2982ddd8fe38930ee. [Online; accessed 16-July-2025].

[6] Kathy Charmaz. 2014. *Constructing grounded theory* (2nd ed.). Sage.

[7] Michele Chiari, Michele De Pascalis, and Matteo Pradella. 2022. Static Analysis of Infrastructure as Code: a Survey. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. 218–225. https://doi.org/10.1109/ICSA-C54293.2022.00049

[8] William Gemmell Cochran. 1977. *Sampling Techniques* (3rd ed.). John Wiley & Sons.

[9] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46. https://doi.org/10.1177/001316446002000104

[10] Ting Dai, Alexei Karve, Grzegorz Koper, and Sai Zeng. 2020. Automatically Detecting Risky Scripts in Infrastructure Code. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. ACM, 358–371. https://doi.org/10.1145/3419111.3421303

[11] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726.

[12] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian A. Tamburri. 2022. Within-Project Defect Prediction of Infrastructure-as-Code Using Product and Process Metrics. *IEEE Transactions on Software Engineering* 48, 6 (2022), 2086–2104. https://doi.org/10.1109/TSE.2021.3051492

[13] Stefano Dalla Palma, Chiel van Asseldonk, Gemma Catolino, Dario Di Nucci, Fabio Palomba, and Damian A Tamburri. 2023. "Through the looking-glass..." An Empirical Study on Blob Infrastructure Blueprints in TOSCA. *Journal of Software: Evolution and Process* (2023).

[14] Fred D Davis, RP Bagozzi, and PR Warshaw. 1989. Technology acceptance model. *J Manag Sci* 35, 8 (1989), 982–1003.

[15] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Guoai Xu, and Shaodong Zhang. 2018. How do mobile apps violate the behavioral policy of advertisement libraries?. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*. 75–80.

[16] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. 2006. Specifying and Reasoning About Dynamic Access-Control Policies. In *Automated Reasoning*. Springer, 632–646.

[17] Georgios-Petros Drosos, Thodoris Sotiropoulos, Georgios Alexopoulos, Dimitris Mitropoulos, and Zhendong Su. 2024. When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 359 (Oct. 2024), 31 pages. https://doi.org/10.1145/3689799

[18] William Eiers, Ganesh Sankaran, and Tevfik Bultan. 2023. Quantitative Policy Repair for Access Control on the Cloud. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 564–575. https://doi.org/10.1145/3597926.3598078

[19] William Eiers, Ganesh Sankaran, Albert Li, Emily O'Mahony, Benjamin Prince, and Tevfik Bultan. 2022. Quantifying permissiveness of access control policies. In *Proceedings of the 44th International Conference on Software Engineering*. ACM, 1805–1817. https://doi.org/10.1145/3510003.3510233

[20] F. M. A. Erich, C. Amrit, and M. Daneva. 2017. A qualitative study of DevOps usage in practice. *Journal of Software: Evolution and Process* 29, 6 (2017), e1885. https://doi.org/10.1002/smr.1885

[21] Jill J Francis, Marie Johnston, Clare Robertson, Liz Glidewell, Vikki Entwistle, Martin P Eccles, and Jeremy M Grimshaw. 2010. What is an adequate sample size? Operationalising data saturation for theory-based interview studies. *Psychology and Health* 25, 10 (2010), 1229–1245.

[22] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology* 106 (2019), 101–121. https://doi.org/10.1016/j.infsof.2018.09.006

[23] Michele Guerriero, Martin Garriga, Damian A. Tamburri, and Fabio Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 580–589. https://doi.org/10.1109/ICSME.2019.00092

[24] Oliver Hanappi, Waldemar Hummer, and Schahram Dustdar. 2016. Asserting Reliable Convergence for Configuration Management Scripts. *SIGPLAN Not.* 51, 10 (oct 2016), 328–343. https://doi.org/10.1145/3022671.2984000

[25] Mujtaba Hassan, Muzammil Hussain, Maham Irfan, et al. 2019. A policy recommendations framework to resolve global software development issues. In *2019 International Conference on Innovative Computing (ICIC)*. IEEE, 1–10.

[26] Md Mahadi Hassan, John Salvador, Shubhra Kanti Karmaker Santu, and Akond Rahman. 2024. State Reconciliation Defects in Infrastructure as Code. *Proc. ACM Softw. Eng.* 1, FSE, Article 83 (jul 2024), 24 pages. https://doi.org/10.1145/3660790

[27] Gary Hickey and Cheryl Kipping. 1996. A multi-stage approach to the coding of data from open-ended questions. *Nurse researcher* 4, 1 (1996), 81–91.

[28] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. 2013. Testing Idempotence for Infrastructure as Code. In *Middleware 2013*. Springer, 368–388.

[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 92–101. https://doi.org/10.1145/2597073.2597074

[30] Stephen Kaplan, Dylan Bulmer, Avery Gosselin, and Sepideh Ghanavati. 2021. Lattice-based Contextual Integrity Analysis of Social Network Privacy Policies . In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. IEEE, 394–399. https://doi.org/10.1109/REW53955.2021.00070

[31] Shoma Kokuryo, Masanari Kondo, and Osamu Mizuno. 2020. An Empirical Study of Utilization of Imperative Modules in Ansible. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*. 442–449. https://doi.org/10.1109/QRS51102.2020.00063

[32] Indika Kumara, Zoe Vasileiou, Georgios Meditskos, Damian A. Tamburri, Willem-Jan Van Den Heuvel, Anastasios Karakostas, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2020. Towards Semantic Detection of Smells in Cloud Infrastructure Code. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*. ACM, 63–67. https://doi.org/10.1145/3405962.3405979

[33] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174.

[34] Julien Lepiller, Ruzica Piskac, Martin Schäf, and Mark Santolucito. 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, 105–123.

[35] Wei Li, Borui Yang, Hangyu Ye, Liyao Xiang, Qingxiao Tao, Xinbing Wang, and Chenghu Zhou. 2024. MiniTracker: Large-Scale Sensitive Information Tracking in Mini Apps. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 2099–2114. https://doi.org/10.1109/TDSC.2023.3299945

[36] Song Liao, Christin Wilson, Long Cheng, Hongxin Hu, and Huixing Deng. 2020. Measuring the effectiveness of privacy policies for voice assistant applications. In *Proceedings of the 36th Annual Computer Security Applications Conference*. 856–869.

[37] Xing Liu, Jiqiang Liu, Sencun Zhu, Wei Wang, and Xiangliang Zhang. 2019. Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. *IEEE Transactions on Mobile Computing* 19, 5 (2019), 1184–1199.

[38] Kirsti Malterud. 1993. Shared Understanding of the Qualitative Research Process. Guidelines for the Medical Researcher. *Family Practice* 10, 2 (07 1993), 201–206. https://doi.org/10.1093/fampra/10.2.201

[39] Davood Mazinanian, Ameya Ketkar, Nikolaos Tsantalis, and Danny Dig. 2017. Understanding the use of lambda expressions in Java. *Proc. ACM Program. Lang.* 1, OOPSLA (2017), 85:1–85:31. https://doi.org/10.1145/3133909

[40] Alok Mishra and Ziadoon Otaiwi. 2020. DevOps and software quality: A systematic mapping. *Computer Science Review* 38 (2020), 100308.

[41] Kief Morris. 2016. *Infrastructure as Code: Managing Servers in the Cloud* (1st ed.). O'Reilly.

[42] Ruben Opdebeeck, Bram Adams, and Coen De Roover. 2025. Analysing Software Supply Chains of Infrastructure as Code: Extraction of Ansible Plugin Dependencies. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2025*. IEEE, 181–192. https://doi.org/10.1109/SANER64311.2025.00025

[43] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2022. Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime. In *Proceedings of the 19th International Conference on Mining Software Repositories*. ACM, 61–72. https://doi.org/10.1145/3524842.3527964

[44] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2023. 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)*. 534–545. https://doi.org/10.1109/MSR59073.2023.00079

[45] Ruben Opdebeeck, Ahmed Zerouali, Camilo Velázquez-Rodríguez, and Coen De Roover. 2021. On the Practice of Semantic Versioning for Ansible Galaxy Roles: An Empirical Study and a Change Classification Model. *J. Syst. Softw.* 182, C (dec 2021), 21 pages. https://doi.org/10.1016/j.jss.2021.111059

[46] Stefano Dalla Palma, Majid Mohammadi, Dario Di Nucci, and Damian A. Tamburri. 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*. ACM, 31–36. https://doi.org/10.1145/3416505.3423563

[47] Lisa Parker, Tanya Karliychuk, Donna Gillies, Barbara Mintzes, Melissa Raven, and Quinn Grundy. 2017. A health app developer's guide to law and policy: a multi-sector policy analysis. *BMC medical informatics and decision making* 17 (2017), 1–13.

[48] Ioannis Paspatis, Aggeliki Tsohou, and Spyros Kokolakis. 2020. AppAware: a policy visualization model for mobile applications. *Information & Computer Security* 28, 1 (2020), 116–132.

[49] Juanjo Pérez-Sánchez, Saima Rafi, Juan Manuel Carrillo de Gea, Joaquín Nicolás Ros, and José Luis Fernández Alemán. 2025. A theory on human factors in DevOps adoption. *Computer Standards and Interfaces* 92 (2025), 103907. https://doi.org/10.1016/j.csi.2024.103907

[50] Giovanni Quattrocchi and Damian Andrew Tamburri. 2022. Predictive maintenance of infrastructure code using "fluid" datasets: An exploratory study on Ansible defect proneness. *Journal of Software: Evolution and Process* 34, 11 (2022), e2480. https://doi.org/10.1002/smr.2480

[51] Akond Rahman, Dibyendu Brinto Bose, Raunak Shakya, and Rahul Pandita. 2023. Come for Syntax, Stay for Speed, Understand Defects: An Empirical Study of Defects in Julia Programs. *Empirical Software Engineering* 28, 93 (2023), 33.

[52] Akond Rahman, Effat Farhana, Chris Parnin, and Laurie Williams. 2020. Gang of Eight: A Defect Taxonomy for Infrastructure as Code Scripts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ACM, 752–764. https://doi.org/10.1145/3377811.3380409

[53] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2018. A systematic mapping study of infrastructure as code research. *Information and Software Technology* (2018). https://doi.org/10.1016/j.infsof.2018.12.004

[54] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 164–175.

[55] Akond Rahman, Md Rayhanur Rahman, Chris Parnin, and Laurie Williams. 2021. Security Smells in Ansible and Chef Scripts: A Replication Study. *ACM Trans. Softw. Eng. Methodol.* 30, 1, Article 3 (Jan. 2021), 31 pages. https://doi.org/10.1145/3408897

[56] Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. 2023. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM Trans. Softw. Eng. Methodol.* 32, 4, Article 99 (May 2023), 36 pages. https://doi.org/10.1145/3579639

[57] Akond Rahman and Laurie Williams. 2018. Characterizing Defective Configuration Scripts Used for Continuous Deployment. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 34–45. https://doi.org/10.1109/ICST.2018.00014

[58] Akond Rahman and Laurie Williams. 2019. Source code properties of defective infrastructure as code scripts. *Information and Software Technology* 112 (2019), 148 – 163. https://doi.org/10.1016/j.infsof.2019.04.013

[59] Akond Ashfaque Ur Rahman, Eric Helms, Laurie Williams, and Chris Parnin. 2015. Synthesizing Continuous Deployment Practices Used in Software Development. In *2015 Agile Conference*. 1–10. https://doi.org/10.1109/Agile.2015.12

[60] Akond Ashfaque Ur Rahman and Laurie Williams. 2016. Security practices in DevOps. In *Proceedings of the Symposium and Bootcamp on the Science of Security*. ACM, 109–111. https://doi.org/10.1145/2898375.2898383

[61] Akond Ashfaque Ur Rahman and Laurie Williams. 2016. Software security in DevOps: synthesizing practitioners' perceptions and practices. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*. ACM, 70–76. https://doi.org/10.1145/2896941.2896946

[62] Jimmy Ray. 2024. *Policy as Code* (1st ed.). O'Reilly Media.

[63] Sofia Reis, Rui Abreu, Marcelo d'Amorim, and Daniel Fortunato. 2023. Leveraging Practitioners' Feedback to Improve a Security Linter. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, Article 66, 12 pages. https://doi.org/10.1145/3551349.3560419

[64] Leah Riungu-Kalliosaari, Simo Mäkinen, Lucy Ellen Lwakatare, Juha Tiihonen, and Tomi Männistö. 2016. DevOps adoption benefits and challenges in practice: A case study. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016*. Springer, 590–597.

[65] Nuno Saavedra and João F. Ferreira. 2023. GLITCH: Automated Polyglot Security Smell Detection in Infrastructure as Code. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, Article 47, 12 pages. https://doi.org/10.1145/3551349.3556945

[66] Nuno Saavedra, João Gonçalves, Miguel Henriques, João F. Ferreira, and Alexandra Mendes. 2023. Polyglot code smell detection for infrastructure as code with GLITCH. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2042–2045.

[67] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. Sage.

[68] J. Schwarz, A. Steffens, and H. Lichter. 2018. Code Smells in Infrastructure as Code. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. 220–228. https://doi.org/10.1109/QUATIC.2018.00040

[69] Rian Shambaugh, Aaron Weiss, and Arjun Guha. 2016. Rehearsal: A Configuration Verification Tool for Puppet. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 416–430. https://doi.org/10.1145/2908080.2908083

[70] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 189–200. https://doi.org/10.1145/2901739.2901761

[71] Parvaneh Shayegh, Vijayanta Jain, Amin Rabinia, and Sepideh Ghanavati. 2019. Automated approach to improve iot privacy policies. *arXiv preprint arXiv:1910.04133* (2019).

[72] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D. Breaux, and Jianwei Niu. 2016. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 25–36. https://doi.org/10.1145/2884781.2884855

[73] Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2024. Automated Infrastructure as Code Program Testing. *IEEE Trans. Softw. Eng.* 50, 6 (June 2024), 1585–1599. https://doi.org/10.1109/TSE.2024.3393070

[74] Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2024. The PIPr Dataset of Public Infrastructure as Code Programs. In *Proceedings of the 21st International Conference on Mining Software Repositories*. ACM, 498–503. https://doi.org/10.1145/3643991.3644888

[75] Thodoris Sotiropoulos, Dimitris Mitropoulos, and Diomidis Spinellis. 2020. Practical Fault Detection in Puppet Programs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ACM, 26–37. https://doi.org/10.1145/3377811.3380384

[76] Styra, Inc. 2023. *The State of Policy as Code Report: Insights as Organizations Scale Authorization Policies.* https://www.styra.com/resources/reports/policy-as-code/

[77] Angela Sweeney, Kathryn E Greenwood, Sally Williams, Til Wykes, and Diana S Rose. 2013. Hearing the voices of service user researchers in collaborative qualitative data analysis: the case for multiple coding. *Health Expectations* 16, 4 (2013), e89–e99.

[78] António Trigo, João Varajão, and Leandro Sousa. 2022. DevOps adoption: Insights from a large European Telco. *Cogent Engineering* 9, 1 (2022), 2083474.

[79] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 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)*. 164–174. https://doi.org/10.1109/SANER.2018.8330206

[80] Alexandre Verdet, Mohammad Hamdaqa, Leuson Da Silva, and Foutse Khomh. 2025. Assessing the adoption of security policies by developers in terraform across different cloud providers. *Empirical Software Engineering* 30, 3 (2025), 74.

[81] Denis Verdon. 2006. Security policies and the software developer. *IEEE Security & Privacy* 4, 4 (2006), 42–49.

[82] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. 2018. Guileak: Tracing privacy policy claims on user input data for android applications. In *Proceedings of the 40th International Conference on Software Engineering*. 37–47.

[83] Fuman Xie, Yanjun Zhang, Chuan Yan, Suwan Li, Lei Bu, Kai Chen, Zi Huang, and Guangdong Bai. 2022. Scrutinizing privacy policy compliance of virtual personal assistant apps. In *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*. 1–13.

[84] Yue Zhang, Muktadir Rahman, Fan Wu, and Akond Rahman. 2023. Quality Assurance for Infrastructure Orchestrators: Emerging Results from Ansible. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. 1–3. https://doi.org/10.1109/ICSA-C57050.2023.00073

[85] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. 2021. PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps.. In *28th Annual Network and Distributed System Security Symposium, NDSS*. 18.

[86] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel Reidenberg, N Cameron Russell, and Norman Sadeh. 2019. Maps: Scaling privacy compliance analysis to a million apps. *Proceedings on Privacy Enhancing Technologies* (2019).