university of groningen

faculty of science and engineering

From Terraform to AWS CloudFormation: A Study of Cost Patterns and Antipatterns

Allia-Iasmina Neamț

university of groningen

faculty of science and engineering

University of Groningen

From Terraform to AWS CloudFormation: A Study of Cost Patterns and Antipatterns

Bachelor's Thesis

To fulfill the requirements for the degree of
Bachelor of Science in Computing Science
at the University of Groningen under the supervision of
dr. D. Feitosa (Computing Science, University of Groningen)
and
Prof. dr. V. Andrikopoulos (Computing Science, University of Groningen)

Allia-Iasmina Neamț (S4796756)

August 1, 2024

Abstract

Context: As one of the most popular software deployment environments, cloud computing presents a wide range of advantages. However, even though a major factor in any project, it is not yet established how much of a role cost plays in the decision between multiple cloud computing services. Rather, it might be the case that cost concerns are initially overlooked, to later emerge and negatively influence the project.

Objective: This study aims to investigate software developers' cost awareness when using cloud computing services, as well as the potential implications of lack thereof. The focus of the research is the AWS CloudFormation service, a popular tool for the provisioning of the infrastructure as code for software deployment.

Methods: Through a process of data mining, we build a dataset of relevant public GitHub repositories that use the AWS CloudFormation service and discuss cost-related concerns in the commit messages. Further, we apply thematic analysis to identify patterns and antipatterns for cost optimization, as well as correlate them with the results from the predecessor study performed in the context of the Terraform service.

Results: With a dataset of 206 commits that make cost-related changes, we demonstrate the applicability of the predecessor Terraform patterns and antipatterns for AWS CloudFormation. Additionally, we identify two new patterns: the usage of cost reports and the implementation of preventative cost reduction templates.

Contents

List of Tables

List of Figures

1 | Introduction

In the software engineering competition for speed and efficiency, cloud computing emerged as an innovative solution for increasing scalability and flexibility. With a large variety of tools and services, companies can now opt to use the cloud instead of managing their own private hardware, while developers are enabled to focus on solutions rather than their limitations [1].

Among the crucial factors that have ensured the popularity of cloud computing is the reduction of costs in comparison to on-premise computing [2]. The initial promise of the cloud has been of fast and convenient services, cheaper than the competitors' [3].

However, after the striking success of the first cloud computing services [4], numerous solutions have emerged from opposing companies [5]. As expected, a wide range of pricing rates has followed, with various payment methods, such as subscription-based plans, pay-as-you-go plans and reserved instances [6]. Contrasting and comparison between cloud computing services became increasingly difficult as assessing the final costs required a comprehensive understanding of the different payment methods, as well as their long-term implications [7]. Additionally, several services involved hidden costs, including data storage and transfer payments that would be requested only after the initial commitment. In numerous cases, by the time such costs are revealed, too many resources have already been invested in the project to consider alternative cloud computing services [8].

Due to the increased difficulty of properly assessing costs, this factor appears to be overlooked when selecting a cloud computing service, as software developers prioritize other elements, including convenience, efficiency and speed [9]. Nevertheless, the failure to accurately estimate expenses can have a significant negative impact on a project by potentially leading to budget overruns, completion delays or span limitations [10].

Given the broad spectrum of cloud computing services, ranging from serverless computing to DevOps tools and more, a scope reduction for our project is in order. Since the focus of the project are cost considerations of cloud computing in general, the analysis of a specific programming language or environment would not provide sufficiently broad results, as external validity might be threatened. Instead, we choose to inspect Infrastructure as Code (IaC) tools, a fairly used practice for provisioning cloud infrastructure by using machine-readable descriptor files [11]. With the main advantage of increased automation, IaC tools also facilitate consistency, scalability and reusability of projects.

As the complexity of cloud-based applications increases, so does the difficulty of manually managing the infrastructure and deployment of such systems due to scaling issues [12], security considerations [13] and human error. Infrastructure as code tools provide a more viable option by offering automation and efficiency. Despite the numerous advantages, similarly to all cloud computing services, these tools can pose several challenges, including cost-related considerations. If ignored and initially unmanaged, inefficient provisioning practices can have a severe negative impact on financial planning and become increasingly more difficult to spot and remedy.

Through this research, our first aim is to identify cost-related discussions within the IaC development community to further highlight the underlying issues that lead to such undesired outcomes. The project is a direct continuation of the study investigating cost awareness in the context of the open-source IaC tool Terraform[1] [14]. The authors identify expenses as a relevant point of concern for software developers, as well as observe the actions taken to reduce these costs. As a successor study, we intend to ensure consistency by following the same methodology while also increasing generalizability by expanding our research to a provider-specific tool. For this study, the specific cloud computing solution under scrutiny is the Amazon Web Services (AWS) CloudFormation[2], a service for the definition and provisioning of Infrastructure as Code through declarative YAML or JSON templates.

Following the same methodology as [14], an initial dataset of cost-related commits is collected as part of a team contribution involving three students. We further develop a software solution that collects a significant dataset by mining data from public GitHub repositories. The projects of interest are the ones that make use of AWS CloudFormation and discuss cost-related topics within the commits. Additionally, since our goal is to uncover the insights of software developers truly familiarized with AWS CloudFormation, we consider projects that simply use the default configuration templates as irrelevant. Thus, the set of projects is filtered to only include the repositories that modify the AWS CloudFormation configuration files. After creating the dataset, we manually validate and label the information.

While the theoretical understanding of the extent of cost awareness in the context of cloud computing services is an essential foundation for future research, identifying concrete solutions can have a practical impact and encourage positive change within the developers' community. After the collective effort of creating the initial dataset of commits, the individual component of the research focuses on uncovering the existence and implications of such a practical solution: software patterns and antipatterns.

Software patterns are design-level reusable solutions to common software development problems. Also referred to as design patterns, these blueprints for improving specific code issues originally emerged in the context of object-oriented design [15], but quickly extended to a wide range of connected fields, such as data management [16] or system design [17, 18]. In contrast, software antipatterns are frequently used counterproductive or ineffective practices that can lead to negative consequences or unsatisfactory outcomes. Where patterns define proven solutions to be followed, antipatterns highlight frequent pitfalls and bad practices that should be avoided if possible. Both patterns and antipatterns can be used to define code quality standards with regard to different aspects, including security [19, 20], efficiency [21] and fault detection [22].

Publicly available information regarding patterns and antipatterns can have a major preventative effect during the development of a project as programmers are enabled to avoid common mistakes. Moreover, even after the completion of a task, continuous access to new information on the topic of good practices can encourage code verification and improvement [23], as well as foster consistency and maintainability [24, 25].

With the individual contribution, we strive to extend another work that also branched out from the original Terraform Cost Awareness study [14], the Terraform Catalogue of Cost Patterns and Antipatterns [26]. Whereas the base study focused on the cost-related messages of Terraform commits, the Catalogue aimed to increase applicability by interpreting the concrete code changes and grouping them into themes to be further refined into cost-optimization patterns. For the Terraform IaC tool, three patterns and seven antipatterns emerged. To increase generalizability, we evaluate the applicability of these results for AWS CloudFormation and, potentially, observe the existence of new themes. The aforementioned labelled commits dataset obtained through the team effort constitutes the basis for the individual pattern and antipattern extraction.

The main goal of our research is therefore *to extend our knowledge regarding cost patterns and antipatterns for Infrastructure as Code tools, and in particular of AWS CloudFormation, in order to offer more thorough insights into practical optimization solutions*. Through prevention and education, we aim to encourage developers to

[1] https://www.terraform.io/
[2] https://aws.amazon.com/cloudformation/

learn from common mistakes and proactively optimize the cost component of cloud computing tools.

Throughout the remainder of the paper, we first review in Section II relevant related works on multiple topics we are interested in, including cost considerations, challenges of IaC tools, specifications of AWS CloudFormation, data mining and (anti)pattern detection. Further, in Section III, we describe the methodology and study design used for the collection and analysis of cost-related AWS CloudFormation commits from GitHub. In Section IV, we present our results regarding patterns and antipatterns, as well as their co-occurrences and comparison to the predecessor study. We further interpret our findings, discuss the implications for both practitioners and researchers and identify threats to validity in Section V. Lastly, Section VI summarizes the observations and describes our potential future works.

2 | Related work

For the state of the art, several general subtopics are of interest for the project: cost in the context of cloud computing as the general focus of the paper, IaC tools as the more specific focus and AWS CloudFormation as the service under investigation. Additionally, as the main techniques required for the team contribution and the individual contribution described in the previous section, data mining and pattern and antipattern detection, respectively, are also to be investigated.

2.1 Cost assessment in cloud computing

Widely popular solutions, cloud computing services are preferred by software developers due to their scalability, flexibility and efficiency. As concluded by recent studies, factors that contribute to this decision can be highly varied, such as the type of the organization or its level of digitalization [27]. Consequently, the choice between cloud computing services has a subjective component, as each software developer intends to best fulfil their needs and expectations.

In the past, cost has been regarded as a highly relevant factor in the context of migration to the cloud [28, 29], as well as an essential consideration for an optimal selection between cloud computing services [30]. However, correctly estimating the expenses of a service appears to be difficult, both in research [31] and in practice. There are little to no previous works regarding the actual degree to which programmers are aware of cost-related issues when selecting between multiple cloud computing services and the impact that lack of consideration might have on a project.

Whereas no outside studies relate entirely to our topic, it must be noted that this research is a direct continuation of the study by Feitosa et al. [14]. Throughout their paper, the authors focus on uncovering cost-related issues and considerations programmers have in the context of the Terraform cloud computing service. By using a process of data mining, the authors have obtained a substantial database of commits and issues from public GitHub repositories that discuss cost-related topics. The data was further statistically analyzed and interpreted through topic modelling and a knowledge graph. As conclusions of the predecessor study, it was observed not only that software developers have significant concerns about the expenses of the Terraform service but also that clear actions are taken in order to reduce these costs. The same methods will be utilized to implement our project, with the significant distinction of focusing on the AWS CloudFormation tool instead of the Terraform tool.

2.2 Challenges of IaC tools

With highly different specifications, IaC tools are a broad class of cloud computing services, still undergoing changes and improvements. Based on the coupling level between the IaC tool and its cloud service provider, two main categories can be identified: provider-specific IaC tools and provider-agnostic IaC tools [32]. The former category employs a tight coupling, as the tools are designed to interact with their particular cloud service providers. Examples include the Amazon Web Service (AWS) CloudFormation and the

Azure Resource Manager Templates[1]. The latter category allows for a looser coupling and the tools are designed in a vendor-neutral manner that facilitates the provisioning of cloud infrastructure, regardless of the used service provider. Examples of provider-agnostic IaC tools are Terraform, Pulumi[2] and OpenTofu[3], an open-source fork of Terraform.

Despite the numerous advantages of provisioning automation, IaC tools also pose disadvantages, many still unexplored throughout current research or lacking awareness within the community. A relevant recent study [33] deployed an approach similar to ours by investigating software developers' discussions on the Stack Overflow platform to highlight the challenges faced when using various tools, such as Terraform and Ansible. The conclusion of the research is straightforward and seven main topics of concern are identified: server configuration, policy configuration, networking, deployment pipelines, variable management, templating and file management.

Moreover, while investigating common strategies for adopting IaC tools and their shortcomings, a previous work [34] reveals a limited level of support and documentation of provisioning techniques. As implied by the research, this leads to developers testing novel but ultimately faulty implementation methods, instead of relying on established good practices. A similar point is also raised in another related study [35], as one of the main concerns identified regarding the usage of IaC tools is the reliability and resilience of the code itself. Improper or undocumented changes to the infrastructure are further portrayed as a significant threat. Within the same research, more suggestions towards mitigating challenges of IaC tools are presented, including access management to limit the misuse of the functionalities caused by lack of information, as well as standardization and reliable implementations for reduced maintenance efforts. The existence of previous works focused on the various hidden challenges of IaC tools proves that the topic is indeed of interest and further research on elements such as cost considerations would be relevant.

2.3 AWS CloudFormation

The Amazon Web Service CloudFormation, released in 2011, is a service that allows users to describe and provision Infrastructure as Code in a declarative manner. The tool has become a popular cloud computing solution, as it decreases the difficulty of infrastructure management by using standardized templates while ensuring reusability and scalability. Additionally, due to its compatibility with a large variety of services provided by Amazon Web Service, CloudFormation is relatively simple to implement and maintain, at least within the AWS context.

However, despite the numerous advantages, the service has several drawbacks and risks that are becoming more visible due to recent research on the topic. One relevant previous work unmasks the hidden vulnerabilities that emerge during updates [36]. In another study regarding IaC tools, the authors reveal that issues of AWS CloudFormation include insufficient transaction management of stack creation and update, as well as the absence of sharing for templates [37].

Whereas it appears clear that there is a general interest in the shortcomings of the AWS CloudFormation service, its expenses are not directly highlighted. There are certain costs associated with the service that might not be justifiable for every project, especially not in comparison to the alternative cloud computing solutions. Research contrasting between multiple such services [38] reveals cost as a relevant differentiation factor and indicates possible shortcomings of the AWS CloudFormation service.

Moreover, as a highly integrated part of AWS, projects that use CloudFormation or any other AWS services are challenging to transfer to other environments due to additional data transfer costs. Despite the cost-related disadvantages, there is no clear indication of software developers adopting a cautious approach to budget planning when selecting the cloud computing service for a project. Given its prominent

[1]https://azure.microsoft.com/en-us/products/arm-templates
[2]https://www.pulumi.com/
[3]https://opentofu.org/

level of popularity and few officially recognized concerns regarding expenses, cost awareness appears to be lacking in the context of the AWS CloudFormation service as well. This behaviour is somewhat surprising, as AWS also provides the AWS Pricing Calculator[4], a web-based solution for estimating the costs of using the services within the AWS environment. The tool is highly customizable and can assess expenses based on input configurations, allowing developers to understand their specific spending scenario and, theoretically, remedy potential issues early on.

While the AWS Pricing Calculator is ideal for pre-deployment financial planning, another tool, the AWS Cost Explorer [5], is also available for management and analysis of costs and usage after deployment. Moreover, external to the environment, but specifically designed to be easily coupled with cloud providers like AWS, the Infracost tool can be particularly useful since it can calculate expected spendings directly from Infrastructure as Code configurations[6]. However, neither solution is automatically integrated with AWS CloudFormation, increasing the difficulty of cost assessment, as it would require time-consuming manual data entry into the tools or research and implementation of other services to automate the process. Additionally, a lack of previous works on the topic highlights a gap in our knowledge of whether developers are aware of such cost estimation options prior to getting started or only after significant budgeting issues are encountered. As the second situation implies a later stage of development, the configuration of AWS CloudFormation might be too complex to easily integrate the cost management tools.

2.4 Data mining

As a process of discovering patterns, trends or meaningful information within large data sets, data mining has become a relevant tool used for highly diverse statistical tasks such as pattern recognition and predictive analysis. Due to the transparency and collaborative view of open science, public repositories can be utilized as a rich and insightful source of information during the process of data mining. Implementing this tool has become increasingly popular and has provided the logistical basis of numerous recent studies. The topics of interest and the datasets to be mined have a wide range, with some relevant examples focused on social media data [39], clinical data [40, 41] and public education data [42].

Research within the computing science field has also benefited from data mining techniques, mostly centered around various question and answers forms, such as Stack Overflow [33, 43], Reddit [44] and GitHub [45, 46].

On the specific topic of data mining for the assessment of cost awareness, the only relevant previous research to be considered is the Terraform study [14]. However, other predecessors have implemented the technique of mining data from public repositories with the purpose of assessing awareness on different yet related topics. Several works analyzing energy consumption awareness [47, 48], as well as the research on performance-related concerns [49], have successfully collected relevant datasets through data mining and provided clear insights into their respective topics. Additionally, the aforementioned related studies have concluded that increased awareness in general can prove to be beneficial.

2.5 (Anti)pattern extraction

As one of the logical operations to be performed on mined data, pattern and antipattern recognition allows the definition and interpretations of practices to be followed or, on the contrary, to be avoided for optimization purposes. In the context of Infrastructure as Code tools, this technique has been successfully used for the optimization of different aspects that can, in turn, improve the overall quality of the project. Multiple previous works propose a similar approach of initially investigating common code changes regarding a specific factor to further extract patterns and antipatterns to potentially remedy the identified

[4]https://calculator.aws/#/
[5]https://aws.amazon.com/aws-cost-management/aws-cost-explorer/
[6]https://www.infracost.io/

issues. One such factor frequently encountered in Infrastructure as Code tools research is code quality, for which multiple pattern extraction methods have been deployed, including quantitative analysis of data mined from public software repositories [50, 51], historical commits interpretation through clustering [52] and code smell examination [53, 54]. As a second example, previous works have also relied on pattern and antipattern extraction to improve software security through methods such as association rule mining to co-locate issues [55] or control and data flow in code smell detection [56].

Whereas pattern and antipatterns extraction is a highly used data interpretation technique, there is little to no related work regarding the optimization of the cost factor, with the only relevant previous research being the predecessor study to this one of the Terraform Catalogue of Patterns and Antipatterns [26]. However, the existence of extensive research using this technique suggests a certain practicability and reliability of patterns and antipatterns that we can further apply to the gap of cost considerations.

3 | Study design

3.1 Objective and Research Questions

As previously mentioned, our first goal is to extend the predecessor Terraform Catalogue of Cost Patterns and Antipatterns [26] through evaluating its applicability for AWS CloudFormation and, thus, the first research question emerges:

- **RQ1:** How applicable are the Terraform cost reduction patterns in the context of AWS CloudFormation?

Moreover, as defined by the predecessor study, a pattern can only be deemed relevant if observed in at least three different repositories. As this was not the case with all commits identified and analyzed for the Terraform Catalogue, the addition of the AWS CloudFormation results might result in previously irrelevant themes passing the threshold. Our second research question is:

- **RQ2:** Do the patterns that failed the relevance check in the context of Terraform become relevant after the addition of the AWS CloudFormation results?

Lastly, since the Terraform Catalogue may not cover all possible patterns, especially in the context of different Infrastructure as Code tools, the third research question aims to increase the generalizability of the set of results:

- **RQ3:** What other patterns emerge?

To answer the research questions, we start from the basis of the Terraform Catalogue of Cost Patterns and Antipatterns and apply a similar methodology. An overview can be found in Figure 3.1 and each step is further detailed.

3.2 Initial dataset

As a team effort, we collect commits discussing costs from repositories that implement AWS CloudFormation. To appropriately reflect what the average software developer is interested in, we gather information from public GitHub repositories, a rich source of diversity. The results are further used for analysis in three individual works, as shown in Figure 3.2. The initial dataset collection can be divided into two stages detailed below.

3.2.1 Data mining

To obtain the desired set of commits, data is mined from public GitHub repositories through an automated code base developed by the team. The implementation has been broken down into several smaller units validated through unit testing. Most of the system logic has been taken over from the methodology of the Terraform study [14], including Python as the programming language. An overview can be found in Figure 3.3 and each unit is explained further.

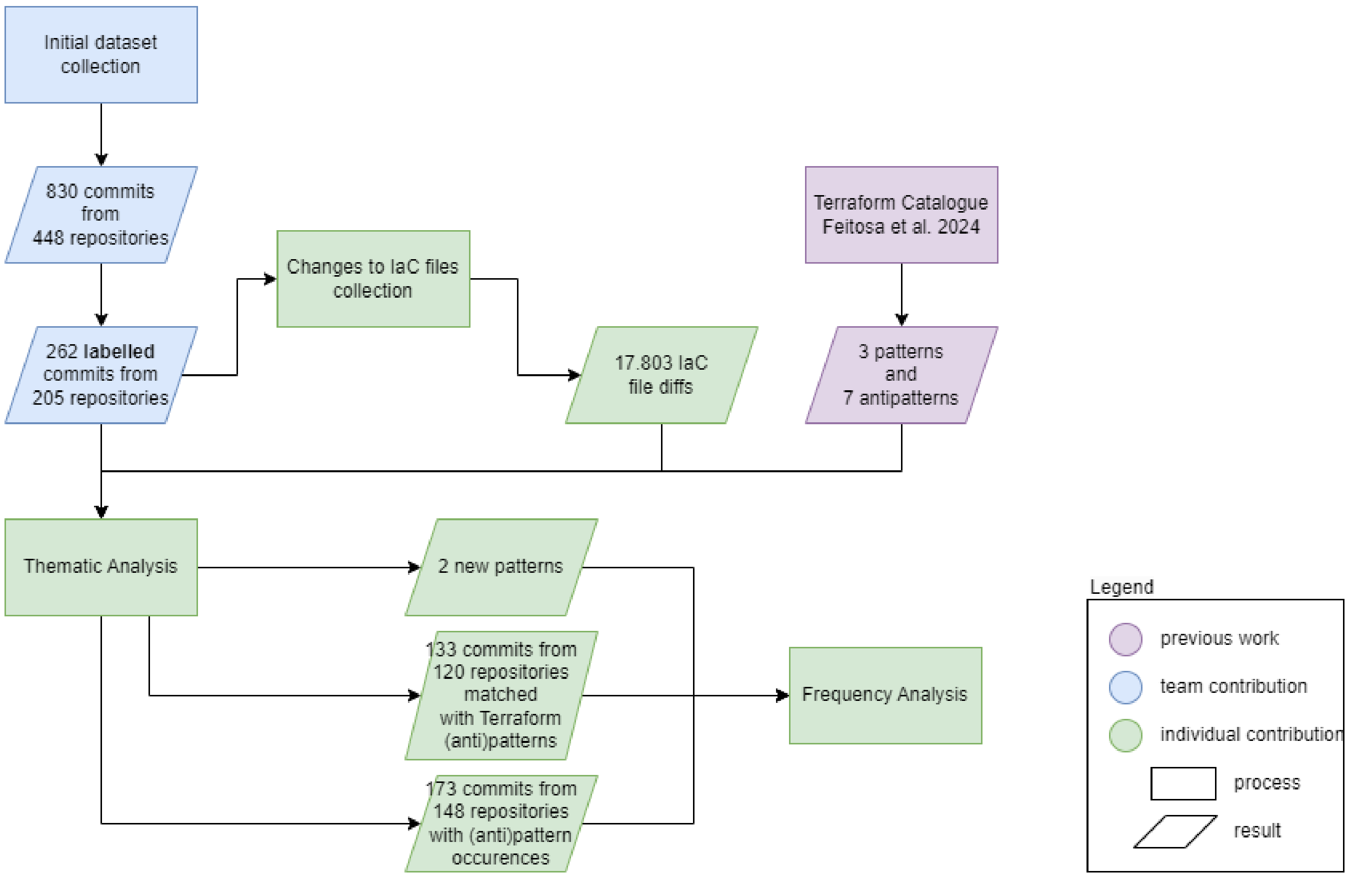
Initial dataset collection
830 commits from 448 repositories
262 labelled commits from 205 repositories
Changes to IaC files collection
17.803 IaC file diffs
Terraform Catalogue Feitosa et al. 2024
3 patterns and 7 antipatterns
Thematic Analysis
2 new patterns
133 commits from 120 repositories matched with Terraform (anti)patterns
173 commits from 148 repositories with (anti)pattern occurences
Frequency Analysis
Legend
previous work
team contribution
individual contribution
process
result

Figure 3.1: Overview of the data collection and thematic analysis

Unit 1 - Data retrieval

The first step is to identify relevant repositories and save them in a dataset in the form of links. Whereas AWS CloudFormation cannot be identified at this point since it does not have a unique extension or mark, we know that the repositories must contain YAML or JSON files for the declarative provisioning of the Infrastructure as Code. After a preliminary analysis of data samples with the two extensions, we chose to only focus on YAML as we observed a lower number of false positives. Further, the GitHub API code search querying method can be used to identify files with the YAML or YML extension. Since such a request gives as a result 1.000 random files that fit the criteria, we increased the likelihood of true positives by writing multiple queries, each also searching for the name of an AWS CloudFormation resource (e.g., "AWS::EC2::Instance") in addition to the extension. Further, we identified the repositories the resulting files belong to and saved the unique links in the dataset. We have run this script multiple times. As part of the GitHub API logic, the random files given by the code search are slightly different each time.

After a unique repository link is saved in the dataset, the next step is for the Python script to make a clone request to the GitHub API and save the actual repository. This step is required since the filtering defined in the next unit needs access to the concrete files of the project. The commits are automatically pulled when the repository is cloned.

Unit 2 - Information filter

For the initial dataset collection, only the commit messages are under scrutiny and we are interested in identifying the ones that contain cost-related information. We find such relevant commits by searching for specific keywords in the messages. For consistency, we have used the set of keywords from the Terraform Cost Awareness Study [14] with the following term stems: bill, cheap, cost, efficient, expens and pay. Commits that do not include any keyword are eliminated from the dataset as the probability of identifying

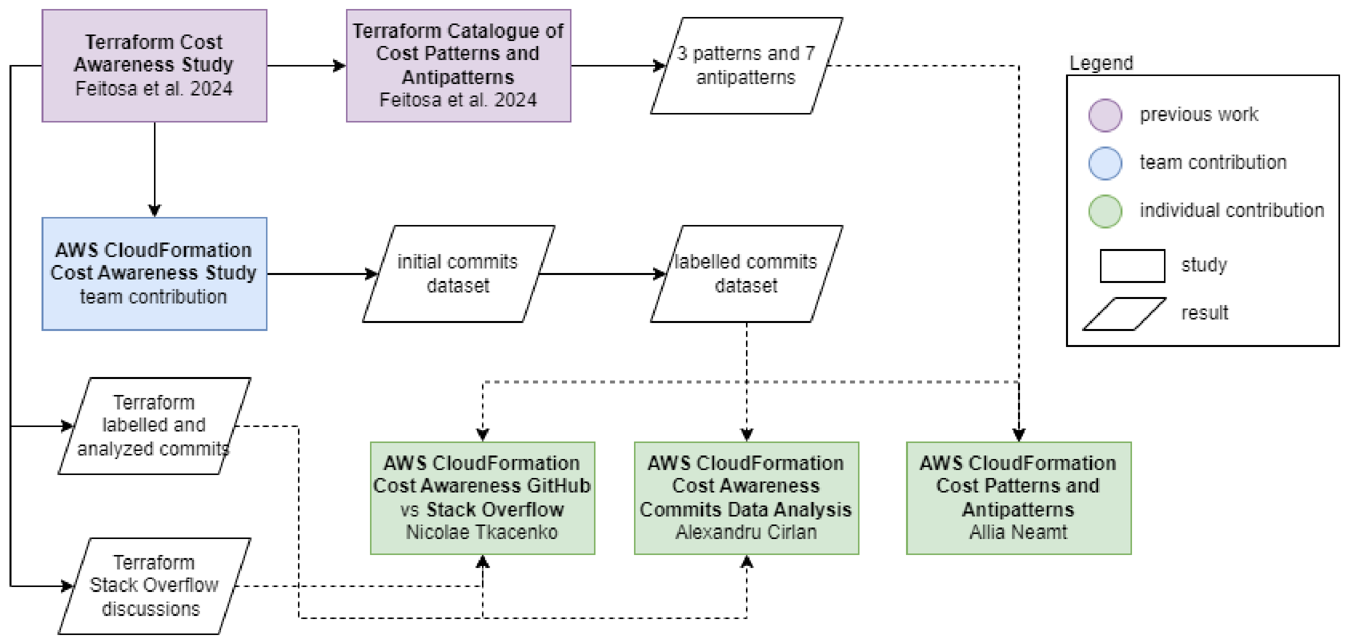
Terraform Cost Awareness Study
Feitosa et al. 2024
Terraform Catalogue of Cost Patterns and Antipatterns
Feitosa et al. 2024
3 patterns and 7 antipatterns
AWS CloudFormation Cost Awareness Study
team contribution
initial commits dataset
labelled commits dataset
Terraform labelled and analyzed commits
Terraform Stack Overflow discussions
AWS CloudFormation Cost Awareness GitHub vs Stack Overflow
Nicolae Tkacenko
AWS CloudFormation Cost Awareness Commits Data Analysis
Alexandru Cirlan
AWS CloudFormation Cost Patterns and Antipatterns
Allia Neamt
Legend
previous work
team contribution
individual contribution
study
result

Figure 3.2: Overview of the predecessor and current related works on IaC cost optimizations

cost considerations in their messages decreases drastically.

Unit 3 - Service filter

Lastly, the cloned repository dataset is filtered to include only the commits that change AWS CloudFormation configuration files. Whereas this service does not have a specific mark, we know that its YAML file must contain a specific structure with three nested keys: first "Resources", then, under it, a key with any name and, finally, the "Type" key. The value of the latter must be the name of an AWS CloudFormation resource, meaning it will start with the string "AWS::". Through a Python script, the dataset is filtered so only the commits that contain this specific structure in at least one YAML file are kept. Additionally, we compare each such file with a set of official AWS CloudFormation templates [1]. If there is a perfect match with one of the templates, the file is not taken into account.

3.2.2 Data validation and labelling

Team members have manually analyzed the initial dataset by understanding the meaning of the commit message and assigning one or more labels to describe it. For consistency reasons, the same labels from the Terraform Cost Awareness study [14] are used, with minimal additions or refinements further elaborated in Alexandru Cirlan's thesis, as shown in Figure 3.2. The purpose of the data validation stage is to establish the meaning of each data point and its relevance for cost awareness. To increase reliability, two team members have independently checked each element. In case of conflicts, a third team member was required to make an impartial choice. We have identified 262 cost-related commits from 205 repositories.

3.3 Commits data collection

In the initial dataset we have collected commits that discuss cost considerations in their messages and change AWS CloudFormation files. As mentioned before, we have also assigned labels to each commit describing, if applicable, the cost-related change as suggested by the message. For the individual contribution, we use this initial dataset for pattern and antipattern extraction. However, moving forward, the concrete code changes of the commits are required since the messages alone do not provide sufficient information regarding the used practices and their implications.

[1] https://aws.amazon.com/cloudformation/resources/templates/

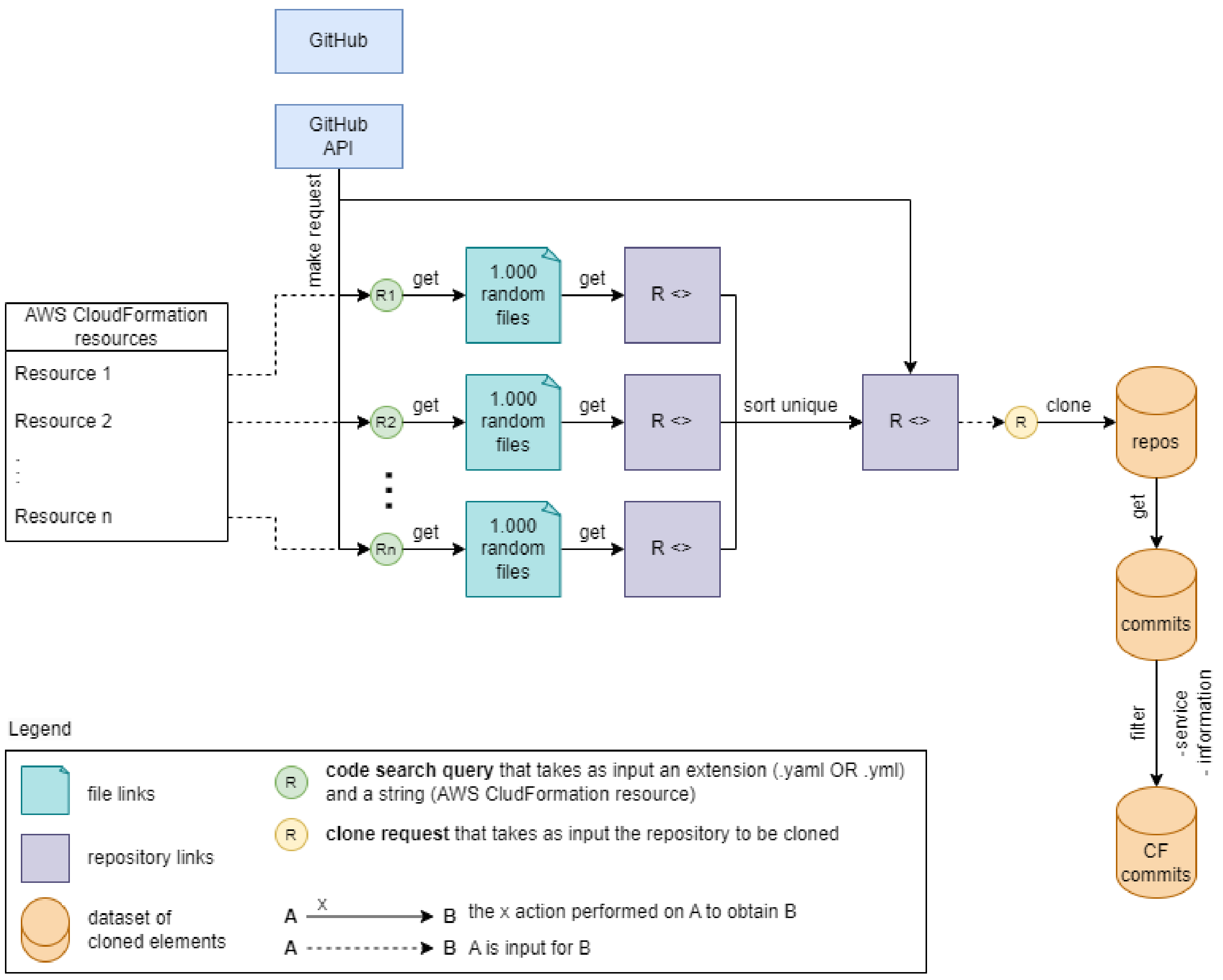
GitHub
GitHub API
make request
AWS CloudFormation resources
Resource 1
Resource 2
Resource n
R1
R2
Rn
get
1.000 random files
get
R <>
sort unique
R <>
R
clone
repos
get
commits
filter
- service
- information
CF commits
Legend
file links
repository links
dataset of cloned elements
R
code search query that takes as input an extension (.yaml OR .yml) and a string (AWS CludFormation resource)
R
clone request that takes as input the repository to be cloned
A x B the x action performed on A to obtain B
A B A is input for B

Figure 3.3: Steps of the data collection team contribution

For each commit previously labelled as cost-related, we have retrieved the changed AWS CloudFormation files. Between the initial dataset collection and the commit changes collection, one repository has become unavailable and its corresponding one commit has been omitted. From the remaining 261 commits, we have collected 17.803 code changes in 388 files.

3.4 Thematic Analysis

To identify cost patterns and antipatterns, we apply thematic analysis, a process of systematically coding and categorizing data to observe the emergence of themes. Through carefully grouping and regrouping the data based on similarities, this method facilitates pattern recognition and has been successfully used for commit analysis by previous works [48].

To ensure consistency with the predecessor study, the set of three patterns and seven antipatterns described in the Terraform Catalogue [26] is utilized as a basis to which we deductively attempt to match the new AWS CloudFormation commits. Additionally, if the previous patterns do not correspond, an inductive approach is used to directly identify similarities between commits to further group them into potential new themes. For a thorough and reliable analysis aligned with the previous study, the following steps are implemented:

1. **Familiarization with the data:** The first step is to manually examine each commit while taking into consideration the message, the concrete code changes from AWS CloudFormation configuration files

and, additionally, the labels assigned during the team contribution.

2. **Generating initial codes:** For each commit, codes are assigned to describe the nature of the cost-related changes. Such a code can be given to multiple commits, whereas some commits that do not directly affect costs receive a no-change code. To ensure consistency, we have started from the Terraform Catalogue [26] by first attempting to match each commit with one or more of the previously defined codes. In case none were deemed appropriate, new codes were generated while maintaining the naming style from the predecessor study. Each new code has been defined and added to the Terraform Catalogue set of codes to allow for consistent future expansions.

3. **Validating themes:** After the generation and definition of codes, the commits can be further grouped based on similarities to observe broader themes. However, to better respond to RQ1 that questions the applicability of the Terraform patterns for AWS CloudFormation, we first attempt to match each commit to one of the previous three patterns or seven antipatterns. As a further step in case no Terraform patterns are appropriate, we group the commits based on the assigned codes to identify recurring trends, as well as outliers in the form of rarely used practices. Aligning with the predecessor study, a theme must occur in at least three different repositories to be considered relevant. Additionally, consolidation discussions are performed to reduce the risk of an interpretation bias and increase reliability.

4. **Defining and naming themes:** Whereas some commits are directly matched with previous Terraform patterns and antipatterns, the new themes observed are further analyzed and defined into new patterns for solutions to common issues or, respectively, antipatterns for ineffective practices to be avoided. To create a coherent set of results with both Terraform and AWS CloudFormation, the documentation of the new patterns and antipatterns includes a brief description, a contextual interpretation of the issues, the observed solution and an example.

3.5 Frequency Analysis

We perform frequency analysis to compare our results to the Terraform Catalogue [26] findings as part of RQ1 and RQ2 and properly analyze the new patterns for RQ3. By first calculating the occurrences and co-occurrences of patterns in both commits and repositories, we aim to further understand their interactions and implications.

4 | Results

4.1 Dataset

Following the methodology described in the previous chapter, we manually analyzed the code changes from the 262 commits with relevant cost-related messages, as established during the team data labelling phase. Furthermore, we used codes to describe the nature of these changes for the thematic analysis process. To ensure consistency, we started from the basis of predecessor codes defined for the Terraform Catalogue of Patterns and Antipatterns and attempted to assign one or more codes to describe the changes in each commit. As a result, we identified the cost-related predecessor codes 104 times in 93 commits from 86 repositories, whereas the *no cost change* code was assigned to 56 commits from 47 repositories, deeming them irrelevant to our research. Additionally, to better tailor to the uniqueness of the AWS CloudFormation IaC tool, we defined new codes to describe changes that did not previously occur in the context of Terraform. We uncovered 98 new codes, corresponding to 133 commits from 109 repositories. Overall, when considering both new and old codes, our results reveal a total of 206 commits from 167 repositories labelled as containing at least one cost-related change. This number represents 78.6% of the total amount of commits with relevant messages we investigated through thematic analysis.

After the coding process, the commits could be further grouped into themes based on similarities. As part of the first research question of observing the relevance of the predecessor Terraform Catalogue results, we first attempted to use the assigned codes to match each AWS CloudFormation commit with one or more of the ten patterns and antipatterns already defined. However, if none were deemed appropriate, we grouped the commits in the hopes of uncovering themes and found two new candidate patterns. As a result, we matched 133 distinct commits from 120 repositories with the predecessor (anti)patterns and 49 commits from 38 repositories with our two new patterns. It must be noted that the intersection of the two sets of numbers reveals 9 commits and, respectively, 10 repositories that contain both old and new (anti)patterns. Additionally, we investigated the co-occurrences of the AWS CloudFormation (anti)patterns and compared them to the predecessor Terraform results in order to better understand their interactions and differences.

Moreover, to ensure consistency with the predecessor Terraform Catalogue, we also implemented the relevance check of each new theme only being taken into account if it occurs in at least three different repositories. Following this restriction, we uncovered 26 commits from 24 repositories with assigned cost-related codes that could not be grouped into new (anti)patterns. Additionally, as part of the second research question, we considered the possibility that the patterns that did not pass the relevance check in the context of Terraform might gain significance after the addition of the AWS CloudFormation results since the total number of distinct repositories with occurrences could reach three. However, none of the codes we identified but could not group into new (anti)patterns could be matched with the Terraform ungrouped codes either.

Our findings have been added to the Catalogue and our observations including co-occurrences of (anti)patterns are presented in detail below.

4.2 Catalogue

The Catalogue of Patterns and Antipatterns for IaC tools contains diverse cost-optimization practices identified through the combined analysis of the Terraform and AWS CloudFormation tools. As a successor of the Terraform Catalogue [26], we briefly revisit the predecessor (anti)patterns with their applicability for AWS CloudFormation and further define the new themes we uncovered. The dataset is also available online[1].

4.2.1 Patterns

As effective practices to be followed, patterns can improve efficiency and reduce the overall spendings of a project. The table below contains the names and descriptions of the three predecessor patterns as defined in the Terraform Catalogue, as well as the number of distinct commits and repositories in which we identified each pattern while analyzing AWS CloudFormation.

Table 4.1: Predecessor Terraform patterns with numbers of occurrences in distinct commits and repositories for AWS CloudFormation

Name	Description	Commits	Repositories
Budget	Use budgets to receive alerts about charged and forecasted costs and control spending.	14	14
Object storage lifecycle rules	Define lifecycle rules for object storage to move objects to cheaper storage or drop them entirely.	13	12
Spot instances	Use spot instances to run interruptible workloads for significant cost savings compared to regular instances.	1	1

Additionally, as a result for our third research question, we identified two new patterns further defined in terms of an introduction, context, solution and an example.

1. **Cost Report -** cost report elements can be used to obtain information on the actual spendings over a period of time.

   Context: Given the complexity and scale of cloud computing projects, cost management might be overlooked and, in turn, lead to overspending.

   Solution: Major cloud providers offer cost reporting tools such as CUR buckets and lambda rules to collect and later visualize and analyze cost and usage information. These tools enable cost tracking, allow programmers to gain insights into spending trends and might uncover cost-related issues.

   We have identified the Cost Report pattern in 38 distinct commits from 27 repositories[2].

   Example:

```
@@ -0,0 +1,24 @@
+ AWSTemplateFormatVersion: '2010-09-09'
+ Description: 'Monthly Cost and Usage Report'
+ Parameters:
```

[1] https://search-rug.github.io/iac-cost-patterns/

[2] https://search-rug.github.io/iac-cost-patterns/cost-report/

```
+    Bucket:
+      Type: String
+    Frequency:
+      Type: String
+      Default: "monthly"
+      AllowedValues:
+      - "monthly"
+      - "weekly"
+      - "hourly"
+  Resources:
+    MonthlyCostReport:
+      Type: "AWS::CUR::ReportDefinition"
   # ...
```

2. **Preventative Template -** templates with predefined configurations that enforce cost-optimizing methods can proactively manage and reduce future expenses.

   Context: Retroactively implementing cost-saving changes can be inefficient and expensive, especially in the context of large systems.

   Solution: Preventative templates can be designed to prioritize cost-optimizations and lead to a proactive enforcement of good practices further on.

   We have identified the preventative template pattern in 23 distinct commits from 21 repositories[3].

   Example:

```
@@ -0,0 +1,111 @@
+ AWSTemplateFormatVersion: 2010-09-09
+ Description: Template for reducing costs on PN
+
+ Parameters:
+   StopEc2FuctionTagName:
+     Type: String
+     Default: 'tostop'
+     Description: Name of EC2 Tag that will be stopped
+
+   StopEc2FuctionTagValue:
+     Type: String
+     Default: 'true'
+     Description: Value of EC2 Tag that will be stopped
+
+   StopEc2FuctionTagNameCronExpression:
+     Type: String
+     Default: 'cron(0 22 * * ? *)'
+     Description: Cron expression when ec2 stop
+
+ Resources:
+   StopEc2Fuction:
+     Type: AWS::Lambda::Function
  # ...
```

4.2.2 Antipatterns

In contrast to patterns, antipatterns highlight counterproductive methods that lead to undesired results and should be omitted from projects. The seven Terraform antipatterns from the Bolhuis et al. study are further briefly described, while the number of occurrences we identified in both commits and repositories are shown.

[3] https://search-rug.github.io/iac-cost-patterns/preventative-template/

Table 4.2: Predecessor Terraform antipatterns with numbers of occurrences in distinct commits and repositories for AWS CloudFormation

Name	**Description**	**Commits**	**Repositories**
Expensive instance	Compute instances are often overprovisioned even when a cheaper instance would suffice.	13	12
Expensive monitoring	Monitoring solutions are expensive and might not be needed.	5	5
Old generation	Using newer resource generations gives similar performance for lower cost.	19	19
Expensive storage type	More expensive storage types are often used even when cheaper storage types would be sufficient.	4	4
Expensive network resource	Network resources like NAT gateways, elastic IP addresses and subnets tend to be expensive while not being strictly needed.	10	8
Overprovisioned resources	Resources like RAM, storage and CPU utilization are often overprovisioned even when lower values are acceptable.	26	25
AWS - Expensive DynamoDB	AWS DynamoDB4 tables often use features that carry cost but are not required, especially for infrequently accessed tables.	42	37

We have not identified any new antipatterns for AWS CloudFormation.

4.3 (Co-)occurrences

To better understand the behaviour and interactions of (anti)patterns, we analyze their co-occurrences within the same commit and, respectively, the same repository, as well as perform comparisons with the results from the predecessor Terraform Catalogue. The co-occurrences are summarized by the UpSet plots in Figure 4.1.

The ten Terraform patterns and antipatterns have a different distribution for AWS CloudFormation. The most significant difference we observe is the increased preference for proactivity and prevention of AWS CloudFormation users, with patterns having higher overall occurrence rates than in the case of Terraform. The two previously most frequent results, the *Expensive instance* and the *Expensive network resource* are encountered significantly less often, coming in seventh and ninth in our list of results. However, while only the fifth most identified Terraform theme, the *AWS-Expensive DynamoDB* antipattern was observed the most for AWS CloudFormation, with 42 commits in 37 repositories. The frequency increase of this antipattern is expected since, as an AWS database service, DynamoDB is logically more likely to be paired with AWS CloudFormation rather than Terraform. Additionally, in the predecessor results, *Budget* was the most frequent pattern with the fourth highest number of occurrences, while the other two patterns had the last and second-to-last rates. In contrast, according to our results, the two new patterns, the *Cost Report* and *Preventative template*, are ranked second and fourth, with the *Budget* pattern being sixth.

By analyzing the relationships between themes, we discover that two or more (anti)patterns are more

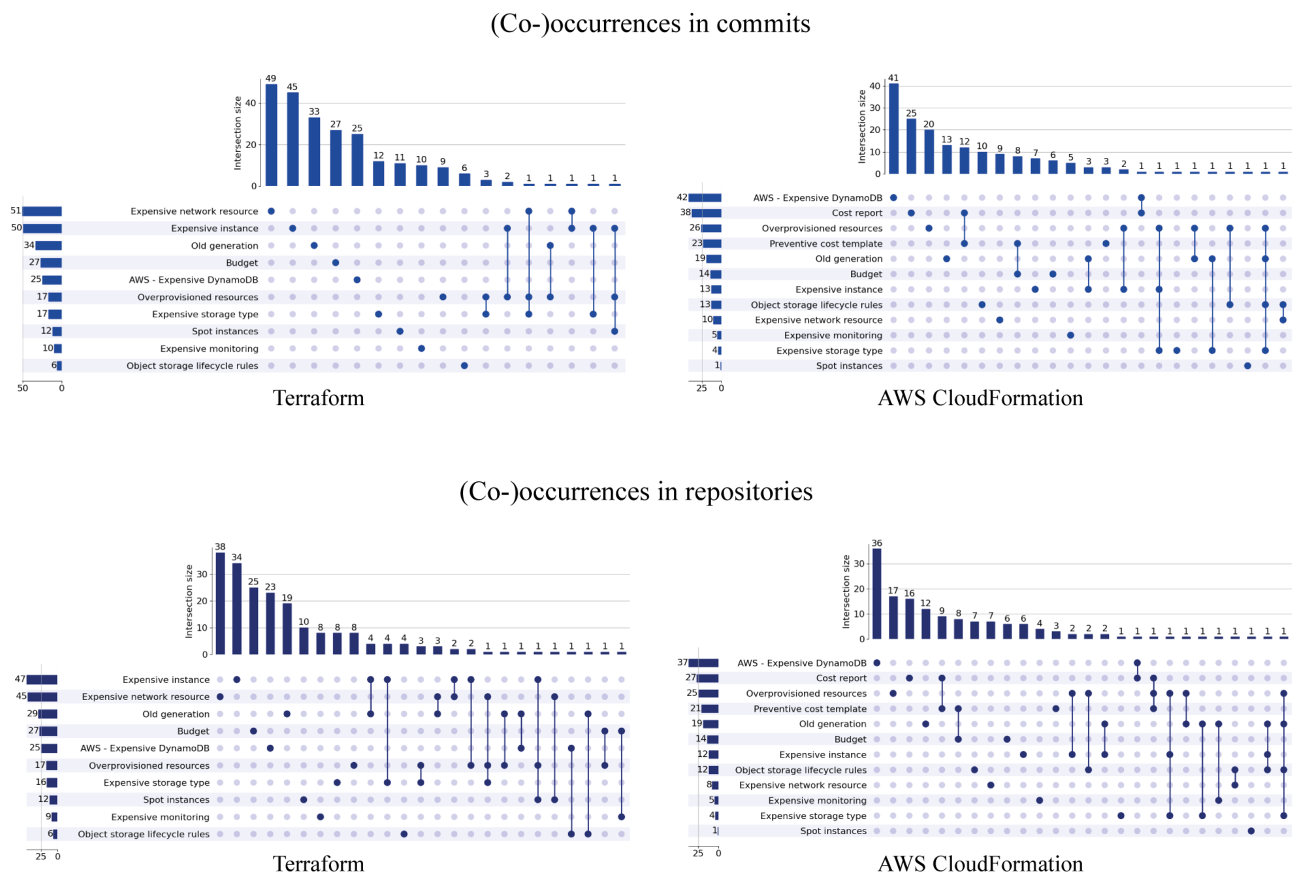

Figure 4.1: Unique co-occurrences of (anti)patterns within single commits and repositories, for both Terraform and AWS CloudFormation

likely to co-occur in the context of AWS CloudFormation than Terraform. This is the case for both commits and repositories, with the *Cost Report – Preventative template* combination being the fourth most frequent overall. Moreover, the three most encountered co-occurrences are composed of either two patterns or two antipatterns, suggesting consistency in mentality and approach, with a low likelihood of developers being preventative and corrective at the same time. However, despite the slightly higher rates of co-occurrences in comparison to the predecessor study, our results still reveal single pattern occurrences as more frequently encountered.

As also observed during the Terraform predecessor study, we encounter multiple implementations of the same (anti)pattern in different commits from the same repository. The most representative example we have found in this sense is the Sage AWS Organizations[4] repository with occurrences of the *Cost Report* pattern in five commits from four different dates between 2022 and 2023. The first commit [link] integrates the usage of cost reports into the project with the required AWS Cost and Usage Report buckets, the second commit [link] adds a cost report lambda function, while the remaining three commits ([link], [link], [link]) implement cost categories. Since clearly connected, the changes could have been grouped into one single, more structured commit. The recurring occurrence of the pattern suggests inconsistency and lack of planning when addressing cost concerns, with a rather chaotic approach. As the predecessor Terraform results also support this observation, it appears that, when using IaC tools in general, developers struggle with the structure for cost optimisations and are prone to making repetitive mistakes.

[4]https://github.com/zaro0508/organizations-infra

5 | Discussion

With the results presented throughout the previous Section, we further interpret them and attempt to understand the hidden implications, following the structure of the research questions. Moreover, to ensure the credibility of our research, we acknowledge the limitations of our study design and present the methods we have used to mitigate the threats. Lastly, we describe the implications of our results for both practitioners and researchers in the hopes of encouraging the integration of better practices, as well as more work on the topic of cloud computing costs.

5.1 Interpretation of Results

When analyzing the applicability of the ten predecessor Terraform cost-optimization (anti)patterns for AWS CloudFormation (**RQ1**), we observe a fairer distribution of patterns and antipatterns. As implied by the previous results, Terraform users are mostly in a reactive state of addressing spending issues only after they emerge and rarely deploy preventative techniques in the form of patterns. AWS CloudFormation users, on the other hand, are almost equally likely to prefer either behaviour, with a steady balance between prevention and correction. Whereas in the context of Terraform antipatterns had significantly higher occurrence rates than patterns, our results reveal a more even distribution of the two, indicating that preferred practices can vary greatly across different IaC tools, with neither being more relevant.

Furthermore, by comparing the co-occurrences of patterns and antipatterns for the two IaC tools, it becomes apparent that it is more frequent for AWS CloudFormation users to address multiple issues at the same time than it had been previously observed for Terraform. As the more significant difference, our results reveal co-occurrences in 18.5% of the total intersection size of the commits (32 out of 173), substantially higher compared to the 4.2% identified for the predecessor (10 out of 237). Similarly, in the case of repositories, we find pattern combinations in 21.6% of the total intersection size (32 out of 148), while the Terraform study only reports 13.2% (27 out of 204). It might be the case that the rigidity of the provider-specific tool and the restrictions of operating within the AWS environment actually encourage developers to rely more on research when making changes. This preventative approach increases the chances of uncovering unforeseen shortcomings or better practices, leading, in turn, to unplanned refactorings. More interestingly, the most frequent co-occurrences are each composed of two patterns, the *Preventative Template* and *Cost Report* and, respectively, the *Preventative Template* and *Budget*, suggesting that an overall proactive mindset can lead to a broader view and increase the likelihood of identifying potential risks early on. The reason behind the preference for combinations of patterns rather than antipatterns, as well as the long-term implications and benefits of prevention, is the subject for future investigation.

Despite the significant differences between the co-occurrences of themes for Terraform and AWS CloudFormation, we also observed a lack of a systematic approach for cost-optimizations, as the same (anti)patterns have been identified in multiple distinct commits of the same repository. The prevalence of this observation in both the predecessor research and our results leads to the conclusion that developers struggle to manage the complexity of IaC tools and could benefit from automated frameworks to enforce more structured strategies.

By identifying each Terraform (anti)pattern at least once in the context of AWS CloudFormation, we strengthen the applicability of the Catalogue for IaC tools in general. However, as discussed in Section 4.1, none of the Terraform patterns that failed the relevance check before passed the threshold after the combination of the results either (**RQ2**), highlighting the fact that each IaC tool has unique particularities and implications which might need to be omitted from our set of patterns and antipatterns in order to increase generalizability. A Catalogue of cost-optimization practices collected from multiple environments can create a useful basis for problem identification and solving, but more specificity can only be gained from detailed research on the IaC tool of interest.

While the investigation of new themes (**RQ3**) reveals two new cost-optimization patterns in comparison to the results of the predecessor Terraform Catalogue, no new antipatterns are identified. Therefore, AWS CloudFormation users might be slightly more preventative in comparison to Terraform users, which could be the case due to the difference in the nature of the two IaC tools. As a provider-specific tool, AWS CloudFormation is designed specifically to function within the AWS environment. Especially for inexperienced developers, this restriction could generate the need to understand the AWS services and possibilities prior to getting started. Through research, patterns and good practices are more likely to be found. On the other hand, Terraform is a provider-agnostic tool that allows coupling with other services. This higher level of freedom could result in developers being more willing to try different methods and fix errors as they are encountered instead of relying on preventative research. Thus, it might be the case that the rigidity of AWS CloudFormation generates a cautious behaviour and leads to the identification of patterns, while the flexibility of Terraform allows developers to make and correct mistakes through antipatterns. However, more research on the significant differences between preferred practices across IaC tools and the potential correlation with the nature of the service, is needed in order to deepen our observations.

5.1.1 Anti-practices

Moving further than our initial research questions, we have also identified multiple instances of a specific behaviour of AWS CloudFormation users that cannot be classified as neither a pattern nor an antipattern and which has not been observed for the Terraform tool either. We have found 19 commits from 18 repositories with changes that directly oppose one or more of our cost-optimization (anti)patterns and, in doing so, lead to price increases. However, the uniqueness of these observed anti-practices is that their commit messages reflect the developer's knowledge of the cost impact, while the reason for the choice is also provided. In comparison to accidental changes with unintentional price implications, the commits in question clearly show a certain degree of cost awareness, but a different factor is prioritized, such as speed or efficiency.

As an example, we identified a commit [link] that changes the storage solution from the cheaper gp3 volumes to the more expensive io1. The commit message reveals an understanding of the price increase ("We stopped using io1 volumes for cost reasons") and the prioritization of a different factor, namely the branch synchronization time ("..., but masters take a really long time to catch up without them"). Furthermore, the commit highlights an overall awareness of cost implications that leads to strategic decision making and financial planning ("This reintroduces it for masters only, which shouldn't be a huge cost given that masters restart rarely.")

While cost is not always the most important concern, the existence of anti-practices proves that multiple AWS CloudFormation developers understand the budget implications and, by doing so, are enabled to make informed decisions and avoid unpleasant surprises. Therefore, through more research and increased awareness on the topic of cost, we expect to not only encourage spendings reductions, but to also promote the importance of educated choices as negative consequences can be better ameliorated when properly understood.

5.2 Implications to Practitioners and Researchers

The Catalogue of Cost Patterns and Antipatterns we defined offers valuable insights into different practices and their implications. Through the combination of the specific results from both Terraform and AWS CloudFormation, we increase the applicability of our observations and encourage developers to make more informed decisions and avoid common pitfalls. The collection of solutions to recurrent issues of provisioning IaC tools, with both explanations and examples, represents a blueprint for simple cost-optimization techniques we expect to help improve code quality. Whereas patterns are intended to promote prevention and caution, antipatterns allow developers to remedy errors in order to avoid the negative consequences efficiently.

Furthermore, as part of a group of connected projects, we aim to stimulate more work on the topic of challenges of IaC tools and, eventually, cloud computing in general. Given the complexity and abstraction level of such services, it becomes difficult to obtain a comprehensive view of the implications of small choices, increasing the likelihood of mistakes with substantial effects. Our research provides the groundwork for multiple branches of related potential investigations. As strongly implied throughout our interpretation of the results, there are clear differences between Terraform and AWS CloudFormation regarding the distribution of (anti)patterns. Future research to uncover the reasons behind this observation and its applicability to other environments can provide significant progress towards understanding the particularities and complications of IaC tools in general. Moreover, by expanding our suggestion that co-occurrences of patterns highlight an overall preventative mindset and risk aversion, subsequent work can further identify the factors that encourage developers to adopt and maintain proactivity.

Lastly, despite AWS being an elaborated environment that offers cost estimation and management solutions, we highlighted a lack of awareness and several difficulties in handling expenses in the context of CloudFormation. The reasons why developers neglect to integrate cost services early on, as well as their level of awareness regarding different optimization tools, are compelling directions for future work. Such research questions could further lead to the broader exploration of the general challenges of complex cloud computing systems or even the development of automated solutions for efficiently operating within these environments.

5.3 Threats to validity

To avoid potential risks to the quality of our research, we identified several limitations of the study design and attempted to mitigate them. Both internal and external validity, as well as reliability are further investigated.

5.3.1 External validity

As the main goal of the group of connected projects is to eventually create a collection of cost-related observations and (anti)patterns for IaC tools in general, a broad applicability is an essential feature. With numerous providers and styles, IaC tools are diverse, not only with regards to provisioning methods and requirements, but also in terms of maintenance and cost-optimization techniques, posing a threat to generalizability. However, as a successor of the two Terraform works [14, 26], we expand the same research questions and methodology to a second IaC tool, thus improving the external validity of our combined results. AWS CloudFormation and Terraform present numerous differences, including the provider, the provisioning language and the type, with the former being provider-specific and the latter provider-agnostic. Given the clear dissimilarities between the two, as well as the consistency of using the same research approach in our successor work, we expect our combined results to have a high applicability for IaC tools in general. However, it must also be noted that the growing diversity of cloud computing solutions cannot be fully represented by the two tools we have chosen to investigate, as unique particularities might arise. Posing the same research questions in the context of other IaC tools as part of future work can increase generalizability even more.

Moreover, through the data collection and labelling process, we have obtained a relatively small set, with 206 commits from 167 repositories. This aspect can threaten the representativeness of our research, as the sample might not appropriately reflect the broader population. However, the comparison and addition of the Terraform results increase the statistical power of our observations, while a future expansion of the dataset to include more entities might also contribute to the external validity.

5.3.2 Internal validity

For processing and analyzing the datasets, several choices could have threatened internal validity and could have been investigated further. Due to time constraints, as well as our interest in active projects, we have only mined the last 10.000 commits of a repository and therefore relevant changes triggered by the maturation of the data might have been neglected. Additionally, both the predecessor Terraform study and our work exclusively focus on public code from the GitHub platform, which can present unforeseen particularities and increase the likelihood of a selection bias. The future exploration of code from different platforms and environments could potentially mitigate the threat.

Furthermore, as part of a trade-off between external and internal validity, we focused on generalizability, leading to a slight reduction in the specificity of our results. This behaviour can be observed in the choice to first attempt to match the AWS CloudFormation new commits with the predecessor Terraform (anti)patterns to increase applicability, while potentially neglecting unique features. Additionally, by only including in the Catalogue patterns that occur in at least three different repositories, we ensure relevance but might disregard very specific yet impactful practices.

As part of the open science approach, we aim to increase transparency and encourage collaboration by thoroughly documenting the process and results. We expect peer review and potential future related works to contribute to mitigating the remaining threats to validity.

5.3.3 Reliability

Since our work involves manual labour and data interpretation throughout multiple stages, it becomes essential to ensure the consistency and stability of the measurements. As part of both the team data labelling process and the individual thematic analysis, we foresaw the possibility of an interpretation bias skewing the results based on the expectations or preferences of the researcher. Additionally, human error in the form of mistakes or oversights could also alter the outcomes. In order to combat both risks, multiple researchers have independently interpreted and labelled each element, while the inter-rater reliability has been calculated and verified to match a predefined standard. However, even with the taken precautions, it must be noted that open coding and thematic analysis are slightly uncertain methods. For this reason, our labelling process might be susceptible to judgement errors and omissions of data points that contain implicit or vague cost considerations.

6 | Conclusions

As part of a team contribution, we mined commits from public GitHub repositories and manually labelled the entities with regards to the cost-related topics mentioned in the messages. As a result, we identified 262 commits that use AWS CloudFormation and discuss cost considerations. Further, in terms of individual research contribution, we performed thematic analysis on the resulting dataset by first assigning codes to describe the concrete code changes of the commits previously deemed relevant. 206 of the 262 commits received at least one cost-related code. Afterwards, we attempted to match the new commits with one of the three patterns and seven antipatterns from the Terraform predecessor study. Each previous theme was identified at least once in the context of AWS CloudFormation, strengthening the applicability of the Terraform Catalogue for IaC tools in general.

Moreover, while implementing the predecessor relevance check of a theme only emerging if encountered in at least three different repositories, we revealed 26 commits with cost-related codes that could not be grouped into new (anti)patterns. Besides, the commits in question could not be matched with any of the Terraform commits that failed the relevance check either, highlighting the existence of very specific particularities of each IaC tool that cannot be generalized. Finally, by grouping the unmatched commits, we identified two new patterns: the usage of cost reporting elements, which we found in 38 distinct commits and the implementation of preventative cost optimization templates, recorded in 23 commits. Therefore, we strengthen the observations of the predecessor study and, through our new results, observe a more preventative and structured approach in CloudFormation repositories.

Regarding future work, we aim to normalize the methodology and analysis procedure throughout the group of connected projects to present a consistent set of combined results. As one of our preliminary choices, we only focused on the YAML provisioning option of AWS CloudFormation and, therefore, we intend to broaden the validity of our results by also analyzing the JSON option. Further, we aim to expand the applicability and reach of our Catalogue by posing the same research questions for different popular IaC tools as dictated by their trends and evolution. By employing this approach, we expect to either confirm the relevance of our (anti)patterns or, on the contrary, identify new practices and expand our set of results even more. Lastly, the growing Catalogue can be further used to create automated tools, such as linters or plugins, for more systematic cost-optimization strategies.

Bibliography

[1] N. Dimitri. "Pricing cloud IaaS computing services". In: *Journal of Cloud Computing* 9.1 (2020). DOI: 10.1186/s13677-020-00161-2.

[2] H. Rehman, S. Majumdar, and M. Rajkumar. "Benefit and risk factors influencing organizations to migrate from On-Premise to cloud computing model of software product". In: *Smart Intelligent Computing and Applications* (2019), pp. 185–202. DOI: 10.1007/978-981-32-9690-9_19.

[3] A. Chalker et al. "Cloud and on-premises data center usage, expenditures, and approaches to return on investment: A survey of academic research computing organizations". In: *Practice and Experience in Advanced Research Computing* (2020), pp. 26–33. DOI: 10.1145/3311790.3396642.

[4] J. Emeras et al. "Amazon Elastic Compute Cloud (EC2) versus in-House HPC platform: A cost analysis". In: *IEEE Transactions on Cloud Computing* 7(2) (2019), pp. 456–468. DOI: 10.1109/tcc.2016.2628371.

[5] R. Kelley et al. "Choosing the right compute resources in the cloud: An analysis of the compute services offered by Amazon, Microsoft and Google". In: *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery* (2020), pp. 214–223. DOI: 10.1109/cyberc49757.2020.00042.

[6] H. S. Abusaimeh, A. A. A. Sharabati, and S. M. Asha. "Using cloud computing services to enhance competitive advantage of commercial organizations". In: *International Journal of Data and Network Science* 7(3) (2023), pp. 1349–1360. DOI: 10.5267/j.ijdns.2023.4.003.

[7] L. Liu et al. "A practical, integrated multi-criteria decision-making scheme for choosing cloud services in cloud systems". In: *IEEE Access* 9 (2021), pp. 88391–88404. DOI: 10.1109/access.2021.3089991.

[8] H. Park, G. R. Ganger, and G. Amvrosiadis. "MIMIR: Finding cost-efficient storage configurations in the public cloud". In: *Proceedings of the 16th ACM International Conference on Systems and Storage* (2023), 22–34. DOI: 10.1145/3579370.3594776.

[9] R. R. Papalkar, P. R. Nerkar, and C. Dhote. "Issues of concern in storage system of IoT based big data". In: *International Conference on Information, Communication, Instrumentation and Control* (2017), pp. 1–6. DOI: 10.1109/icomicon.2017.8279126.

[10] A. S. Dunk and A. Kilgore. "Top management involvement in RD budget setting: The importance Of financial factors, budget targets, and RD performance Evaluation". In: *Advances in Management Accounting* 11 (2003), pp. 191–206. DOI: 10.1016/s1474-7871(02)11008-2.

[11] Y. Zhang et al. "Lifting the fog of uncertainties". In: *Proceedings of the 2023 ACM Symposium on Cloud Computing* (2023), 48–64. DOI: 10.1145/3620678.3624646.

[12] K. Patel. "Mastering Cloud Scalability". In: *Advances in computer and electrical engineering book series* (2024), 155–169. DOI: 10.4018/979-8-3693-0900-1.ch008.

[13] S. Jha and H. Gupta. "Cloud computing security challenges and related mitigation strategies". In: *AIP Conference Proceedings* (2024). DOI: 10.1063/5.0214142.

[14] D. Feitosa et al. "Mining for Cost Awareness in the Infrastructure as Code artifacts of cloud-based applications: An exploratory study". In: (2023). DOI: 10.2139/ssrn.4436874.

[15] E. Gamma et al. "Design Patterns: Elements of Reusable Object-Oriented Software". In: *Addison-Wesley Professional* (1994).

[16] F. Al-Hawari. "Software design patterns for data management features in web-based information systems". In: *Journal of King Saud University* 34(10) (2022), 10028–10043. DOI: 10.1016/j.jksuci.2022.10.003.

[17] M. Ozkaya and M. Kose. "Designing and Implementing Software Systems using User-defined Design Patterns". In: (2021). DOI: 10.5220/0010571404970504.

[18] D. Drozdov et al. "Utilizing Software Design Patterns in Product-Driven Manufacturing System: A Case Study". In: *Studies in computational intelligence* (2019), 301–312. DOI: 10.1007/978-3-030-27477-1_23.

[19] A. Van Den Berghe, K. Yskout, and W. Joosen. "A reimagined catalogue of software security patterns". In: (2022). DOI: 10.1145/3524489.3527301.

[20] K. Yskout, R. Scandariato, and . W. Joosen. "Do security patterns really help designers?" In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* 1 (2015), 292–302. DOI: 10.1109/ICSE.2015.49.

[21] M. E. Rana et al. "The Impact of Flyweight and Proxy Design Patterns on Software Efficiency: An Empirical Evaluation". In: *International Journal of Advanced Computer Science and Applications* 10(7) (2019). DOI: 10.14569/ijacsa.2019.0100724.

[22] N. K. L. Cheng, N. C. P. Chang, and N. C. P. Chu. "Software fault detection using program patterns". In: (2011). DOI: 10.1109/icsess.2011.5982308.

[23] T. Feng et al. "Software design improvement through anti-patterns identification". In: *IEEE International Conference on Software Maintenance* (2004), p. 524. DOI: 10.1109/ICSM.2004.1357866.

[24] S. Rochimah, P. G. Nuswantara, and R. J. Akbar. "Analyzing the Effect of Design Patterns on Software Maintainability: A Case Study". In: (2018). DOI: 10.1109/eeccis.2018.8692876.

[25] P. Hegedűs et al. "Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability". In: *Communications in computer and information science* (2012), 138–145. DOI: 10.1007/978-3-642-35267-6_18.

[26] K. Bolhuis, D. Feitosa, and V. Andrikopoulos. "A Catalog of Cost Patterns and Antipatterns for Infrastructure as Code". In: *2024 50th Euromicro Conference on Software Engineering and Advanced Applications*. 2024.

[27] R. Machuga. "Factors determining the use of cloud computing in enterprise management in the EU (considering the type of economic activity)". In: *Problems and Perspectives in Management* 18(3) (2020), 93–105. DOI: 10.21511/ppm.18(3).2020.08.

[28] V. Andrikopoulos et al. "How to adapt applications for the Cloud environment". In: *Computing* 95(6) (2012), pp. 493–535. DOI: 10.1007/s00607-012-0248-2.

[29] P. Jamshidi, A. Ahmad, and C. Pahl. "Cloud migration research: A systematic review". In: *IEEE Transactions on Cloud Computing* 1(2) (2013), pp. 142–157. DOI: 10.1109/tcc.2013.10.

[30] M. Hosseinzadeh et al. "Service selection using multi-criteria decision making: A comprehensive overview". In: *Journal of Network and Systems Management* 28(4) (2020), pp. 1639–1693. DOI: 10.1007/s10922-020-09553-w.

[31] M. Shuaib et al. "Why adopting cloud is still a challenge?—A review on issues and challenges for cloud migration in organizations". In: *Advances in Intelligent Systems and Computing* (2019), 387–399. DOI: 10.1007/978-981-13-5934-7_35.

[32] O. Tomarchio, D. Calcaterra, and G. D. Modica. "Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks". In: *Journal of Cloud Computing* 9(1) (2020), p. 49. DOI: 10.1186/s13677-020-00194-7.

[33] M. Begoug et al. "What do Infrastructure-as-Code practitioners discuss: An empirical study on Stack Overflow". In: *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2023), pp. 1–12. DOI: 10.1109/esem56168.2023.10304847.

[34] M. Guerriero et al. "Adoption, support, and challenges of Infrastructure-as-Code: Insights from industry". In: *IEEE International Conference on Software Maintenance and Evolution* (2019), pp. 580–589. DOI: 10.1109/icsme.2019.00092.

[35] G. N. Nedeltcheva et al. "Challenges towards modeling and generating Infrastructure-as-Code". In: *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (2023), 189–193. DOI: 10.1145/3578245.3584937.

[36] J. Lepiller et al. "Analyzing Infrastructure as Code to prevent intra-update sniping vulnerabilities". In: *Tools and Algorithms for the Construction and Analysis of Systems* (2021), pp. 105–123. DOI: 10.1007/978-3-030-72013-1_6.

[37] Y. Yamato et al. "Development of template management technology for easy deployment of virtual resources on OpenStack". In: *Journal of Cloud Computing* 3(1) (2014), p. 7. DOI: 10.1186/s13677-014-0007-3.

[38] Y. Al Moaiad et al. "Cloud service provider cost for online University: Amazon Web Services versus Oracle Cloud Infrastructure". In: *Lecture Notes in Computer Science* (2023), 302–313. DOI: 10.1007/978-981-99-7339-2_26.

[39] Y. Xue et al. "A LDA-based social media data mining framework for plastic circular economy". In: *International Journal of Computational Intelligence Systems* 17(1) (2024), p. 8. DOI: 10.1007/s44196-023-00375-7.

[40] R. Hellali et al. "Corticosteroid sensitivity detection in sepsis patients using a personalized data mining approach: A clinical investigation". In: *Computer Methods and Programs in Biomedicine* 245 (2024), p. 108017. DOI: 10.1016/j.cmpb.2024.108017.

[41] H. Liao et al. "Mining and fusing unstructured online reviews and structured public index data for hospital selection". In: *Information Fusion* 103 (2024), p. 102142. DOI: 10.1016/j.inffus.2023.102142.

[42] R. Cerezo et al. "Reviewing the differences between learning analytics and educational data mining: Towards educational data science". In: *Computers in Human Behavior* 154 (2024), p. 108155. DOI: 10.1016/j.chb.2024.108155.

[43] A. Diyanati et al. "A proposed approach to determining expertise level of StackOverflow programmers based on mining of user comments". In: *Journal of Computer Languages* 61 (2020), p. 101000. DOI: 10.1016/j.cola.2020.101000.

[44] T. Iqbal et al. "Mining Reddit as a new source for software requirements". In: *IEEE 29th International Requirements Engineering Conference* (2021), pp. 128–138. DOI: 10.1109/re51729.2021.00019.

[45] D. Atzberger et al. "CodeCV: Mining expertise of GitHub users from coding activities". In: *IEEE 22nd International Working Conference on Source Code Analysis and Manipulation* (2022), pp. 143–147. DOI: 10.1109/scam55253.2022.00021.

[46] C. Zimmerle et al. "Mining the usage of reactive programming APIs". In: *Proceedings of the 19th International Conference on Mining Software Repositories* (2022), 203–214. DOI: 10.1145/3524842.3527966.

[47] L. Cruz and R. Abreu. "Mining questions about software energy consumption". In: *Proceedings of the 11th Working Conference on Mining Software Repositories* (2014), pp. 22–31. DOI: 10.1145/2597073.2597110.

[48] I. Moura et al. "Mining energy-aware commits". In: *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories* (2015). DOI: 10.1109/msr.2015.13.

[49] T. Das, M. Di Penta, and I. Malavolta. "A quantitative and qualitative investigation of performance-Related commits in Android apps". In: *IEEE International Conference on Software Maintenance and Evolution* (2016), pp. 443–447. DOI: 10.1109/icsme.2016.49.

[50] A. Rahman, E. Farhana, and L. Williams. "The 'as code' activities: development anti-patterns for infrastructure as code". In: *Empir. Softw. Eng.* 25 (2020), pp. 3430–3467. DOI: 10.1007/s10664-020-09841-8.

[51] A. Rahman. "Anti-Patterns in Infrastructure as Code". In: *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018. DOI: 10.1109/icst.2018.00057.

[52] W. Chen, G. Wu, and J. Wei. "An Approach to Identifying Error Patterns for Infrastructure as Code". In: *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018. DOI: 10.1109/issrew.2018.00-19.

[53] T. Sharma, M. Fragkoulis, and D. Spinellis. “Does your configuration code smell?” In: *Proceedings of the 13th International Conference on Mining Software Repositories*. ICSE ’16. ACM, 2016. DOI: 10.1145/2901739.2901761.

[54] J. Schwarz, A. Steffens, and H. Lichter. “Code Smells in Infrastructure as Code”. In: *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*. IEEE, 2018. DOI: 10.1109/quatic.2018.00040.

[55] F. A. Bhuiyan and A. Rahman. “Characterizing co-located insecure coding patterns in infrastructure as code scripts”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*. ASE ’20. ACM, 2020. DOI: 10.1145/3417113.3422154.

[56] R. Opdebeeck, A. Zerouali, and C. De Roover. “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)*. IEEE, 2023. DOI: 10.1109/msr59073.2023.00079.