The Journal of Systems and Software 215 (2024) 112112

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

# Mining for cost awareness in the infrastructure as code artifacts of cloud-based applications: An exploratory study☆

Daniel Feitosa *, Matei-Tudor Penca, Massimiliano Berardi, Rares-Dorian Boza, Vasilios Andrikopoulos

*Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands*

ARTICLE INFO

*Keywords:*
Cloud computing
Cost awareness
Mining software repositories
Cloud orchestration

ABSTRACT

**Context:** Cloud computing's rise as the primary platform for software development and delivery is largely driven by the potential cost savings. However, it is surprising that no empirical evidence has been collected to determine whether cost awareness permeates the development process and how it manifests in practice.
**Objective:** This study aims to provide empirical evidence of cost awareness by mining open source repositories of cloud-based applications. The focus is on Infrastructure-as-Code artifacts that automate software (re)deployment on the cloud.
**Methods:** A systematic examination of 152 735 repositories yielded 2 010 relevant hits. We then analyzed 538 relevant commits and 208 relevant issues using inductive and deductive coding and corroborated findings with discussions from Stack Overflow.
**Results:** The findings indicate that developers are not only concerned with the cost of their application deployments but also take actions to reduce these costs beyond selecting cheaper cloud services. We also identify research areas for future consideration.
**Conclusion:** Although we focus on a particular Infrastructure-as-Code technology (Terraform), the findings can be applicable to cloud-based application development in general. The provided empirical grounding can serve developers seeking to reduce costs through service selection, resource allocation, deployment optimization, and other techniques.

## 1. Introduction

Cost reduction is one of the main drivers of cloud adoption (Andrikopoulos et al., 2013). Cost savings for the cloud consumers accrue due to two phenomena. First, access to any kind of computational resources (both hardware and software) is on-demand and is being billed utilities-style (Mell et al., 2011). This means that scaling up and down the amount of these resources to meet the needs of the current load leads to higher efficiency in comparison with a fixed infrastructure such as e.g. in a traditional data center. Compounding this, there is also no need for upfront capital expenses for the acquisition of these resources, e.g. to cope with unforeseen demand. This means a nearly complete transfer of the focus from the management of capital expenses to operational ones, also known as the CAPEX-to-OPEX shift (Armbrust et al., 2010). Second, and on top of that, the economies of scale realized by the cloud service providers, and especially by the ones such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform, collectively known as *hyperscalers* due to their ability to enable scaling to virtually infinite levels of demand, allow these providers to offer access to these resources for almost (always) declining prices (Harms and Yamartino, 2010). This makes cloud computing very attractive to all kinds and sizes of organizations and enterprises.

Further testimony to the importance of cost for adopters of cloud computing is the amount of related work on areas investigating how to minimize and/or manage this cost. This can be achieved, for example, from the perspective of cloud consumers through optimal cloud service provider selection (Tricomi et al., 2020; Hosseinzadeh et al., 2020), and from the perspective of the providers by means of optimized task scheduling (Arunarani et al., 2019) or other profit optimization techniques such as energy consumption minimization (Cong et al., 2020). Whats more, every cloud service provider offers in one form or another

☆ Editor: Shane McIntosh.

* Corresponding author.
*E-mail addresses:* d.feitosa@rug.nl (D. Feitosa), matei.penca1@gmail.com (M.-T. Penca), massimiliano.berardi93@gmail.com (M. Berardi), raresboza@gmail.com (R.-D. Boza), v.andrikopoulos@rug.nl (V. Andrikopoulos).
*URLs:* https://feitosa-daniel.github.io (D. Feitosa), https://vandriko.github.io (V. Andrikopoulos).

https://doi.org/10.1016/j.jss.2024.112112
Received 18 December 2023; Received in revised form 10 April 2024; Accepted 22 May 2024
Available online 24 May 2024

cost calculator tools, allowing their users to get a quote on their service consumption based on their foreseen computational, storage, network etc. needs. Consulting on cloud cost management has developed into its own line of business, with even cloud service providers themselves offering such services to their users. In all cases, cost is referring to the monetary expenses of hosting and running software in one of the cloud deployment models as defined by NIST (Mell et al., 2011).

However, and to the extent of our knowledge, no empirical evidence exists on whether and in what form the cost of cloud-based software projects is discussed among the involved developers. While on the surface such concerns appear to be outside of the remit of software development per se, the situation in practice is quite different. The fact, for example, that the DevOps paradigm became popular and widely adopted almost in the same timeframe as cloud computing hints that software developers cannot easily ignore the operational aspects of the code they produce, including its cost. Furthermore, the aforementioned utilities-like billing of cloud services means that developers are now in a position to be held effectively accountable for the generated revenue of the software that they produced, deployed, and ran on the cloud infrastructure. As such, and further bolstered by the amount of anecdotal evidence, we do expect software developers to be actively aware and concerned about the cost of their software, and we set out to collect evidence of this.

The objective of this study is therefore clear: *to examine to what extent software developers are aware of the cost of deploying and operating cloud-based software, and what kind of concerns and action initiatives they are having about it.* We choose MSR (Mining Software Repositories) as the means to answer this question empirically. Among its other uses, MSR allows to empirically study otherwise subjective or external phenomena in combination with (or through) large-scale systematic mining of development artifacts. Fields such as green software engineering (Hindle, 2013; Pereira et al., 2021), risk assessment (Choetkiertikul et al., 2015; da Costa et al., 2017; Choetkiertikul et al., 2018), and software classification (Howard et al., 2013; LeClair et al., 2018; Sas and Capiluppi, 2022) have advanced noticeably due to MSR. In a similar fashion to these works, we hypothesize that *the amount and diversity of costs-related information in cloud-based software project repositories is sufficient to produce meaningful insights*.

Cloud-based application development, however, covers a very wide range of application types and development activities, and this creates a question of scope in this study. Selecting for a specific programming language or ecosystem as in other MSR studies does not produce meaningful results here since these are orthogonal concerns to the use of cloud infrastructures. Instead, we scope our search for evidence to *cloud orchestrator artifacts* included in open source projects. Cloud orchestrators are *Infrastructure as Code* (IaC) solutions that provide an abstraction layer over the self-service management APIs of the various cloud service providers, with the intention of flattening out the differences between them (de Carvalho and de Araujo, 2020). This is usually achieved by means of *descriptor files*, i.e., configuration files that when interpreted by the orchestrator ensure that both the underlying infrastructure is made available, and the tasks required for the (re)deployment of software on this infrastructure is executed correctly. Descriptor files are usually semi-structured documents in a machine-readable format that is easy to process such as YAML or JSON, and like any other configuration files they are added to code repositories to be managed by the respective version control system. Orchestrators are either (cloud service) provider-specific, such as Amazon Web Services CloudFormation, or provider-agnostic, such as Terraform, Cloudify, Apache Heat, and others as discussed for example in Tomarchio et al. (2020). For reasons that will be discussed in Section 3, we focus our work specifically on Terraform artifacts.

In summary, this paper aims to report on the first work that attempts to perform cost awareness mining for cloud-based application development, starting with IaC artifacts. In addition, we fortify the findings of the mining process by also investigating to what extent the identified related topics are discussed on a popular developer forum, Stack Overflow,[1] in relation to the same type of artifacts. The analysis of the involved posts confirms the fitness of our result to purpose. The contributions of this work can therefore be summarized as follows:

- we collect and present *empirical evidence* of the existence of cost-related information pertinent to Terraform artifacts as it appears in (open) source code repositories;
- we triangulate and augment this information extraction by identifying Stack Overflow posts where pertinent discussions take place;
- we make publicly available a *curated dataset* of the artifacts identified through this evidence-collection process and the scripts we used during the data collection, see Feitosa et al. (2024);
- we define a set of *actionable items for future research on cost awareness* based on our preliminary analysis of this dataset.

The rest of this paper is structured as follows. Some related works are presented in Section Section 2. Section 3 discusses the study design, including the definition of research questions to investigate. Section 4 presents our findings, and Section 5 discusses our effort to triangulate them through developers' interactions on a public forum. Section 6 offers a discussion on the implications of these findings for practitioners and researchers, including the formulation of a research agenda for future work. Finally, Sections 7 and 8 close this paper with a reflection on the threats to validity to this study, and a summary of its main findings, respectively.

## 2. Related work

As mentioned in the previous section, and to the best of our knowledge, there is no existing study gathering empirical evidence about how developers deal with the cost of deploying cloud-based or otherwise software, and definitely none collected through repository mining. The closest works in spirit in this direction are instead studies on mining energy consumption awareness on the developer's side such as the work by Moura et al. (2015) and Bao et al. (2016). Other works such as the one by Pinto et al. (2014) on the same topic, or Das et al. (2016) analyzing documented performance-related issues can be also considered somewhat related to ours.

However, that is not to say that there are no research efforts for supporting the management of cost in such systems. In fact, that is a flourishing line of research approached from different perspectives. Despite being recognized early on as a major concern when migrating existing systems to the cloud (Andrikopoulos et al., 2013), for example, and being a crucial component in many migration support approaches (Jamshidi et al., 2013), estimating the cost of deploying and running software in the cloud remains an open research challenge (Shuaib et al., 2019). Cost of deployment and operation of target applications is one of the common factors taken into consideration in works researching mechanisms for efficient decision making on which cloud service provider(s) to use (Hosseinzadeh et al., 2020). This also appears to be the case for the related problem of optimizing the selection of services from potentially across service providers, commonly known as cloud service composition (Amato and Venticinque, 2016; Vakili and Navimipour, 2017). Managing the cost (and energy consumption) is also identified as one of the focus points of architecting cloud-based software as discussed in the survey of Chauhan et al. (2017) on the topic.

## 3. Study design

In this section, we elaborate on the methods employed to achieve the objective presented in Section 1. In particular, we discuss the derived research questions, the required data and its collection, and how the data is analyzed to provide the necessary answers.

[1] https://stackoverflow.com/

### 3.1. Research questions

To explore developers' cost awareness and how it manifests in project repositories, we define three main questions:

**$RQ_1$** What kind of relevant information can we extract from commits on IaC artifacts?

**$RQ_2$** How can we augment this information further based on issues raised in the respective repositories?

**$RQ_3$** How can we organize this information so we can gain deeper insights from it?

In absence of any prior study establishing a link between activities in repositories and cost awareness, we define these questions on a purely exploratory base. We strive to find information related to actions taken on cloud-based software projects on the basis of impacting their deployment cost. In an initial exploration, we seek stronger evidence of such actions and, thus, focus on changes to code connected with acknowledged impact to cost by means of commit messages ($RQ_1$). Next, we expand the search scope for discussions that may not incur in changes but that are relevant nevertheless ($RQ_2$). In the case of this study, we explore entries in the issue trackers of projects that were identified in the previous research question. At this phase, it is also relevant to understand how the topics of discussion differ (if at all) compared to the information on commit messages. This information can guide future research and development efforts (see Section 6).

The information collected in the previous RQs can inform future research and practitioner decisions. However, this 'rawer' format of the data may often require examining the dataset in more depth to make connections between the identified core concepts and more informed decisions. Thus, it is instrumental to understand how the knowledge evolving from the information extracted in the previous research questions can be structured for this purpose ($RQ_3$).

### 3.2. Case selection

As with previous studies on mining software repositories, we direct our efforts to open source repositories. The reasoning here is that using such an open collection of repositories minimizes the selection bias on our side and therefore increases the robustness of our possible findings. Among other qualities, we seek a source that can provide a *sizable* and *diverse* population. Also, from a population that meets the quality criteria, we must find the *breadth* and *depth* of the data that one can derive. As a source of repositories, we choose GitHub mainly due to the volume and diversity of software projects that are available in it. Moreover, we aim at maximizing the data pool while maintaining a systematic and repeatable approach and, therefore, GitHub's search features and API are instrumental.

We also already put forward in the introductory section our particular interest in investigating projects that use cloud orchestrators. Since we seek to identify evidence of developers' discussions over cloud infrastructure matters (in this case, cost), it is natural to narrow down our search scope to projects that use IaC. The commit messages and issues involving descriptor files have the potential to bring up concerns we are interested in. For the purposes of this study, we choose to work with Terraform[2] as it is one of the most notable and widely adopted orchestrators (de Carvalho and de Araujo, 2020). Terraform is known for providing an open-source version, cloud services compatibility, interface accessibility and mature API (de Carvalho and de Araujo, 2020).

We clarify that there are other viable options, Cloudify[3] being a well known one, marginally behind Terraform in terms of performance (Kovács and Kacsuk, 2017). Hyperscalers are also offering their own IaC solutions, with AWS' CloudFormation being a particularly popular one. In principle, therefore, we could execute this study with artifacts of multiple cloud orchestrators taken into account. However, the complexity of analyzing multiple platforms was deemed prohibitive since this kind of study is a resource-demanding endeavor as-is. Moreover, we found the number of projects on GitHub mentioning Cloudify (876 repositories with around 79K commits) to be significantly fewer than that of projects mentioning Terraform (171K with around 1M commits). Cloudify has some of its features locked behind a paywall (Kovács and Kacsuk, 2017), potentially turning away many small time and open-source developers, which might explain this difference in numbers. Furthermore, provider-specific cloud orchestrators would need a deeper understanding on our part of the cloud services being used which would detract from the focus of the study. Nevertheless, we acknowledge the relevance of investigating other platforms in future work (see Section 6).

In summary, the cases under consideration for this study comprise *GitHub projects that use Terraform as their cloud orchestrator, and address matters related to cost in commit messages and issues discussions*. In the following we describe how we actually apply the latter criterion.

### 3.3. Data collection and analysis

Following best practices in evidence-based software engineering (Wohlin et al., 2012), we characterize the population of this study in terms of *unit of analysis*. The units of this study are commits or issues in repositories that contain evidence of cloud cost awareness. For each valid unit, we extract the triplet `<unit-id;unit-content;label>`, where: (a) `unit-id` refers to the repository and commit hash or issue id, (b) `unit-content` is the commit message or issue text, and (c) `label` is a descriptor highlighting the main theme(s) derived from `unit-content`.

The process to extract and analyze the units is summarized in Fig. 1. Locating Terraform descriptor files is relatively straightforward since they are by default written in the HashiCorp Configuration Language (HCL), a language that is indexed by GitHub, and carry the `.tf` or `.tf.json` extension as per the Terraform documentation.[4] Furthermore, Terraform's first release was in 2014, allowing us to constrain our search further for repositories after this date. To automate the interaction with GitHub's API, we use PyGitHub.[5] Therefore we set PyGitHub to search, on a day-to-day basis,[6] until the end of May 2022 (when the data collection for this study took place), for repositories that contain HCL files created after 2014. The search returns a candidate set of 156 585 repository links. Removing repositories from this set that do not include `.tf` or `.tf.json` files reduces this set to 152 735 repositories for further consideration.

We then use a list of keyword stems to match against commit messages and further search in this set for cost-awareness. However, it is worth noticing that since there are no previous works on the same topic we cannot reuse their keywords list and we have to come up with our own. After some piloting, and considering the research questions we aim to answer, we decide on the following keyword stems,[7] (in alphabetical order):

`bill`, `cheap`, `cost`, `efficient`, `expens`, and `pay`.

[2] https://www.terraform.io/

[3] https://cloudify.co/

[4] https://www.terraform.io/language/files

[5] https://pygithub.readthedocs.io/

[6] This was done to avoid the limitations in the amount of results by the GitHub API per request.

[7] Treating them as stems means that `expens` for example will match against `expense` `expenses`, `inexpensive`, and `inexpensively`.

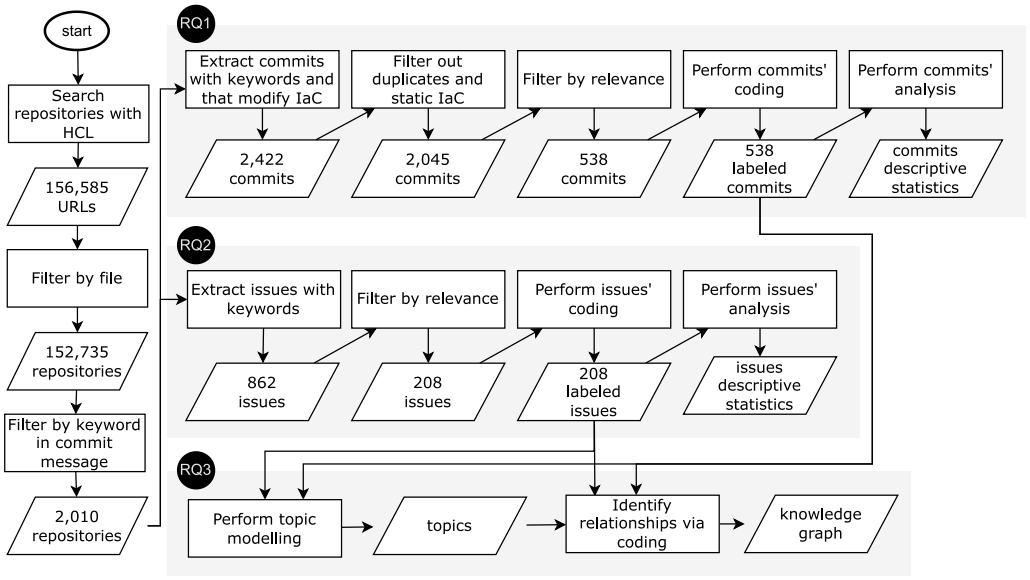


**Fig. 1.** Steps of data collection and analysis.

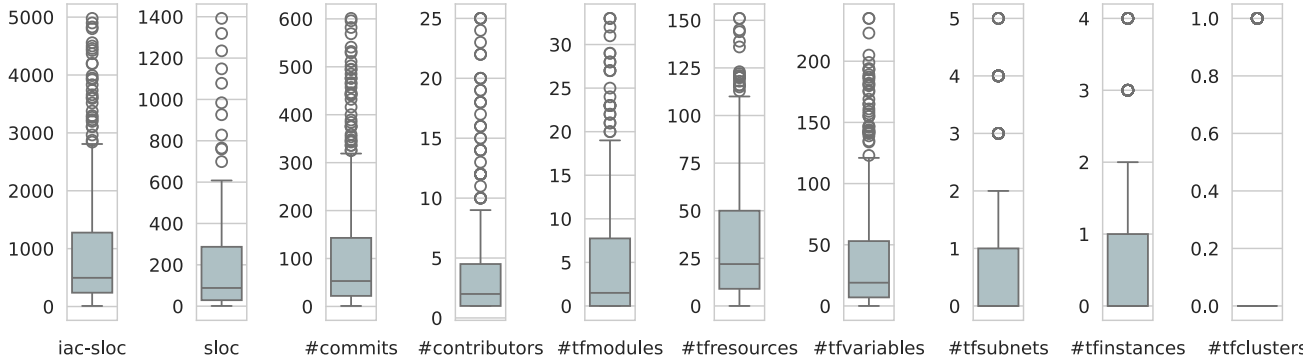


**Fig. 2.** $RQ_1$ repository demographics (90th percentile).

Using this list, we instruct PyDriller (Spadini et al., 2018) to process all commit messages in the candidate repositories for the presence of one or more these keywords and we output the repository name, commit hash and message. PyDriller is a Python library used for analyzing Git repositories. We chose this tool for its easy-to-use API and broad adoption by the MSR community, which we interpret as an additional sign of quality.

After this step, a much more manageable set of 2010 repositories containing 6116 potentially related commits is identified as a result. This set of commits and their respective repositories serve as input to collect the data and perform the analysis for each research question.

### 3.3.1. $RQ_1$

To answer the first research question, we must identify what kind of information related to cloud cost management can be extracted from the commits. Out of this set of 6116 commits, 377 are from forked repositories already in the set. After filtering out these commits and those that do not modify any Terraform files, we are left with 1162 repositories and 2045 related commits. The selected commits are then inspected manually to decide their actual relevance. For that, each commit message is checked by two researchers and validated by a third one. Any conflicts are resolved by the entire team in consolidation meetings. The output of this process results in the identification of 538 relevant commits. The selected units come from 434 distinct repositories.

In Fig. 2, we characterize the repositories in terms of IaC artifacts' size (`iac-sloc`, max. 89K), source code size (`sloc`, max. 1210K), number of commits (`#commits`, max. 17K), contributors (`#contributors`, max. 596), number of Terraform modules (`#tfmodules`, max. 1.4K), resources (`#tfresources`, max. 2.5K), variables (`#tfvariables`, max. 5K), subnets-related definitions (`#tfsubnets`, max. 32), instances-related definitions (`#tfinstances`, max. 36), and number of Terraform clusters-related definitions (`#tfclusters`, max. 21). The max values are not visible in Fig. 2 since we needed to trim the top 10% values of each variable to better visualize the plot. We also note that 65% (284) of the population comprise repositories that contain (and manage) IaC artifacts only (i.e., they do not contain source code). These characteristics suggest that the population is varied with a tendency for repositories maintained by few contributors and concentrated in repositories of more Terraform and other IaC files than source code. This is possibly an indication of developers taking over a "Terraform expert" role in teams that become responsible also with dealing with the operational cost aspect of each project.

We then proceed to collect what kind of information related to cloud cost management can be extracted from these commits. This is primarily a manual task, to be performed using open coding, a form of inductive coding (Corbin and Strauss, 2014). For this task, each data point (for this RQ, commit message) is labeled first by two researchers and validated by a third. The labels refer to central

ideas in the discussion and their characteristics. Any conflicts (incl. disagreements) are resolved by all five researchers in a consolidation meeting. In this meeting, we verify the rationale for the codes and analyzed the evidence again (i.e., the complete commit message), and argued to a consensus. We did not have cases where a consensus was not met after this process.

As it will be discussed further in Section 4, all collected units are analyzed to identify relevant characteristics such as the prevalence of the various labels and distribution of the units among repositories. In addition, meta-information such as the `unit-id` is used to analyze the repositories, e.g., regarding the number of source lines of code (SLOC) and number of contributors.

#### 3.3.2. $RQ_2$

For the second research question, we wish to explore what kind of additional information related to cloud cost awareness can be extracted from entries in issue trackers that adds to that extracted from commit messages. As shown in Fig. 1, we start from the list of 1339 repositories obtained from the first filtering of commits based on keyword. We use this list because discussion may take place before actual changes happen (as demonstrated by modification to Terraform files), and we would like to capture them too. Also, this set of repositories allows us to narrow down the population to an amount of issues that we can feasibly extract from GitHub since they already contain some evidence that cost may be a concern for the project.

To extract the issues, we provide GrimoreLab's Perceval (Dueñas et al., 2018) with the repository owner username, the repository name, and a GitHub API token. Perceval is a Python-based tool that can collect data from various sources, including issue trackers. We chose Perceval for its versatility and validation within the MSR community. The tool then returns us a list of issue objects that contain every single detail pertaining the issue. We then extract the issue objects that contain one or more of the defined keywords in the title, body or any of the comments; this results in an initial set of 862 entries.

Next, we apply a process similar to that described for $RQ_1$ to filter for relevant issues and then label them. Since issues may contain long discussions, the coding is focused on the context, i.e., the sentence where the keyword appears and the surrounding ones (when needed). If multiple keywords appear on the same issue and they refer to different contexts, multiple checks are performed. We note that the labels are applied to whole issues based on evidence found in any of the content elements (i.e., title, body or comments). This process culminates in the selection of 208 units belonging to 89 distinct repositories. In Fig. 3, we characterize the repositories in terms of the same metrics as in Fig. 2: `iac-sloc` (max. 43K), `sloc` (max. 1 486K), `#commits` (max. 6K), `#contributors`, (max. 156), `#tfmodules` (max. 373K), `#tfresources` (max. 646K), `#tfvariables` (max. 3.5K), `#tfsubnets`, (max. 39), `#tfinstances` (max. 34), and `#tfclusters`, max. 27). We again trimmed the top 10% values of each variable to better visualize the plot. This population of repositories is considerably smaller than that collected for $RQ_1$, and contains a lower amount of projects that contain (and manage) Terraform files (only 44%). Nevertheless, the descriptive statistics summarized by Fig. 3 suggest a fair distribution of data points, with higher averages (by approx. a factor of two).

Similar to $RQ_1$, the collected units are analyzed to identify relevant characteristics such as the prevalence of the various labels and distribution of the units among repositories. Moreover, the results are compared to those obtained from $RQ_1$, to reflect on the value added by exploring issues. Both of these issues are to be discussed further in Section 4.

#### 3.3.3. $RQ_3$

In the final research question, we want to examine the dataset in more depth and make contextual connections between the identified core concepts. This process is done in two steps, starting with topic modeling (Blei, 2010) and followed by a second coding activity, culminating in the creation of a knowledge graph which combines the results of the previous steps. We describe these steps in the following.

Topic modeling is a statistical learning tool that is well-suited for abstracting connections between words (into topics) from a corpus of documents (Blei, 2010). We apply Latent Dirichlet Allocation (LDA) (Blei, 2010), a popular topic-modeling technique used in a number of MSR studies e.g. Zimmerle et al. (2022), Al Alamin et al. (2021), Chen et al. (2012), Hindle et al. (2011).

Before applying LDA, a number of steps must be undertaken for cleaning and preparing the corpus. GitHub content contains a mixture of Markdown and HTML code scattered throughout the text that we mined. Thus, we first convert any markdown syntax to HTML, remove the content in *pre*, *code* and *blockquote* elements, and remove the markup and URLs. We note that we remove code snippets because we are interested in developers' discussions and the code may bias the LDA to find irrelevant topics. Next, we prepare each document using Stanza's (Qi et al., 2020) neural network NLP pipeline.[8] In particular, we tokenize it into sentences containing lists of tokens, extract part-of-speech (PoS) tags of each token, and lemmatize them. The latter step is especially relevant to improve the validity of the bag of words used for topic modeling. From the prepared tokens, we filter the PoS tags that may contain relevant words, i.e., nouns, adjectives, adverbs and verbs. We also remove tokens according to a stop-word list that we built by merging the Gensim's (Řehůřek and Sojka, 2010) list for English with terms deemed irrelevant for our analysis. We perform the same steps for both commits' and issues' text.

With the prepared corpus, we use Gensim to convert it into a bag of words, build a TF-IDF (term frequency–inverse document frequency) model and use the two to create LDA models. To build an LDA model, we must find a suitable number of topics ($K$), as it impacts the granularity of the results (Abdellatif et al., 2020; Han et al., 2020). The quality of the model is also affected by other hyperparameters, such as $\alpha$ (referring to document-topic density) and $\beta$[9] (referring to topic-word density) are some of the most relevant (Campbell et al., 2015; Treude and Wagner, 2019). Following the related literature (Reboucas et al., 2016; Al Alamin et al., 2021; Zimmerle et al., 2022), we experimented with the ranges $K = \{5, 6, \ldots, 34, 35\}$, $\alpha = \beta = \{50/K, 0.01\}$. We also varied the *chuncksize* (number of documents for each training mini-batch)[10] $S = \{1, 2, 4, 8, \ldots, 1024\}$ as it can yield positive impact on the model (Hoffman et al., 2010). Although the quality of the model is ultimately assessed by us, we relied on the coherence (Abdellatif et al., 2020; Al Alamin et al., 2021) and perplexity (Treude and Wagner, 2019; Zimmerle et al., 2022) metrics to narrow down the number of candidate models to be manually inspected. Finally, we train models using 100 iterations for hyperparameter exploration, and use 1000 iterations to train the models selected for manual inspection. After this process, we settled with $K = 12$, $\alpha = 50/K$, $\beta = 0.01$, and $S = 32$ for the model based on the commits dataset model; and $K = 5$, $\alpha = 0.01$, $\beta = 50/K$, and $S = 2$ for the model based on the issues dataset.

Next, we aim to build a knowledge graph by making informed connections between relevant words. The set words come from both the coding performed for $RQ_1$ and $RQ_2$ as well as from the interpretation of the topics of both models. To interpret the topics, we used the open card sorting technique (Abdellatif et al., 2020; Zimmerle et al., 2022) and analyzed the top words of a topic via a random sample of documents dominated by it (Ahmed and Bagherzadeh, 2018). The sample size varied from 10 to 20 documents per topic, depending on the number of documents connected to a given topic (we aimed for 5%–10% of the number of documents connected to the topic). Three researchers

[8] https://stanfordnlp.github.io/stanza/pipeline.html
[9] In Gensim, the parameter 'eta' refers to $\beta$.
[10] https://radimrehurek.com/gensim/models/ldamodel.html

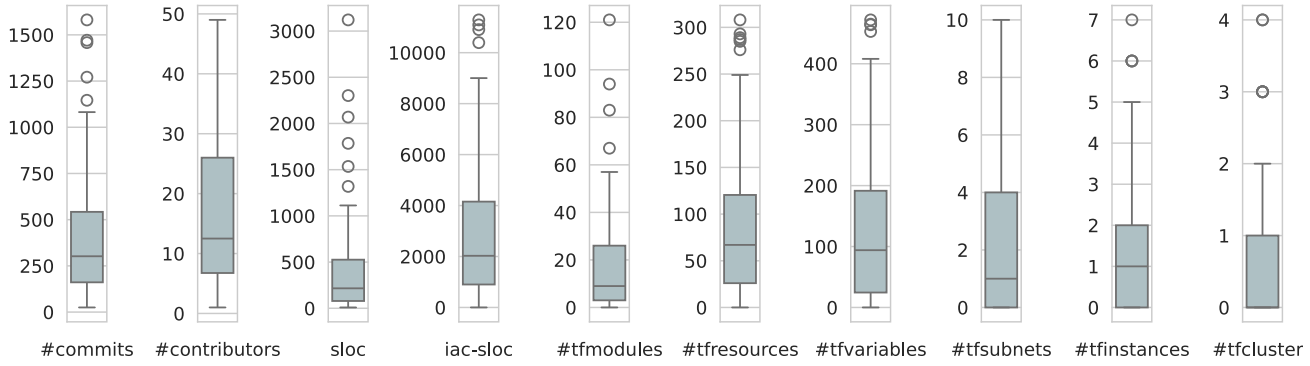


**Fig. 3.** $RQ_2$ repository demographics (90th percentile).

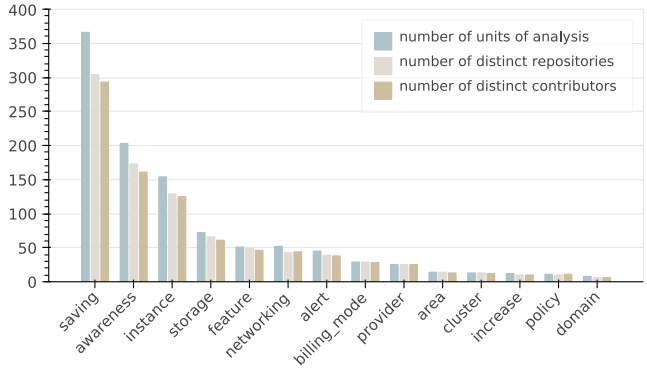


**Fig. 4.** Coding demographics of commits.

applied the technique, with the other two researchers validating the results and resolving disagreements. Finally, the relationship between words is defined by applying axial coding and selective coding (Corbin and Strauss, 2014) on the preexisting labels (from $RQ_1$ and $RQ_2$) and words from topics. Axial coding is a combination of inductive and deductive coding with the goal of relating codes (e.g., finding emerging categories). Selective coding is a similar process, but to find the core set of codes and categories. Group categories and relationship between labels can be identified from a topic (if a word can describe the topic well) and from manual inspection of labels and their linked documents.

## 4. Results

The presentation of the results follows the RQs defined in Section 3.

### 4.1. Cost Awareness in Commits ($RQ_1$)

As a result of the coding process, we obtain a set of 14 distinct labels related to the 538 commits. Each commit has between one and five labels assigned to it based on the message content. Fig. 4 lists the collected labels, and shows their recurrence among units (i.e., commits) and distinct repositories, and number of distinct contributors associated with them. We provide a description of each label in Table 1 and elaborate on the most recurrent ones in the following.

The most popular label by all metrics is `saving`. A notorious example from the dataset is:

> `Removed the default use of detailed monitoring. (#17) * Reduces CloudWatch costs for metrics by 80%`
> *blinkist/terraform-aws-airship-ecs-cluster (commit hash: d7aa6599)*

This example showcases the fact that monitoring solutions provided by the cloud service providers might offer deep insights into the billing of their services, but are also incurring expenses for their usage, as with any other cloud service. This being the most common label indicates not only awareness on behalf of the developers, but also specific cost reducing actions as an effect of this awareness. As a matter of fact, approximately 70% of all commits as shown in Fig. 4 document concrete actions to save cost. Characterizing the types of actions taken is outside of the scope of this work, but can be easily achieved by further processing the commits in the dataset.

The next most popular label is `awareness`, which does also imply action, e.g.:

> `nat gateway is verry [sic] expensive`
> *stealthHat/k8s-terraform (hash: 681a3f8b)*

This particular example identifies a well known issue with Amazon Web Services' NAT Gateway service with respect to cost accruing easily out of control that is even the subject of online memes and frequently recurring Twitter threads.[11]

Finally, the label `instance` understandably figures among the top recurring labels. An example from the dataset is:

> `Move from m4.large to m5.large. The new gen have more CPU and are cheaper`
> *alphagov/govuk-aws (hash: 6cfda6ad)*

In this respect, this label can be located anywhere between the previous two ones: it can identify awareness and intention of action at the same time. On a related note, it is worth pointing out that, although less prevalent, we also identified information explicitly discussing cost increases due to a prior change (in roughly 2% of the units).

Altogether, we notice developers' consciousness of consequences of decisions in deployment. Moreover, labels such as `instance` and `storage` point to specific aspects of the deployment that are or can be tuned to manage cost. From Fig. 4, we cannot infer or speculate over which of such aspects are more often treated this way. However, considering the configuration options offered by platforms such as

[11] See e.g. https://twitter.com/quinnypig/status/1440301033314349062

**Table 1**
Label descriptions.

| Label | Description |
|---|---|
| alert | text expressing concerns related to billing alarms enforcing an upper threshold on costs. |
| area | text expressing concerns related to server or instance geographical location. |
| awareness | text simply mentioning concerns with cost (without necessarily implying action). |
| billing_mode | text expressing concerns related to the type of billing plan being used (e.g., on-demand for development or normal plan for production). |
| cluster | text expressing concerns related to cluster configuration. |
| domain | text expressing concerns related to domain name system and IP addresses. |
| feature | text expressing concerns related to various features such as logging, load balancers or usage of third party libraries. |
| increase | text expressing concerns related to increase in cost due to a change. |
| instance | text expressing concerns related to computational instances (e.g., Amazon AWS t2.micro) used in the deployment. |
| networking | text expressing concerns related to networking configuration. |
| policy | text expressing concerns related to the implementation of general rules to prevent excessive charges. |
| provider | text expressing concerns related to choosing a service providers (e.g., Amazon, Azure, Google). |
| saving | denotes mentioned changes made to save costs. |
| storage | text expressing concerns related to storage solutions (e.g., Amazon gp2) used in the deployment. |

Terraform, the data suggest a broad understanding of these options. At the same time, the number of repositories in our dataset compared to the total amount of projects using Terraform may also suggest that only a small percentage of developers are interested or aware of the cost-saving possibilities in IaC configuration. We note that this observation is also speculative as it requires further investigation to e.g. discard test or template projects. Further insights can be gained by analyzing the dataset in more depth for e.g. characterizing the types of repositories with respect to the labels used in their commits, identifying the relation between labels and contributors and so on. This kind of analysis is left as future work and as part of a call to the wider community, as discussed below in Section 6.

### 4.2. Cost Awareness in Issues ($RQ_2$)

The coding of the 208 issues resulted in recognizing labels we identified in the commit-units of analysis, without adding new ones. We note that one label, namely `policy`, was not identified among issues. Each issue discussion has between one and four labels assigned to it based on the text in the title, body and comments. Fig. 5 describe the labels in terms of their recurrence among units (i.e., issues) and distinct repositories, and number of distinct creators and commenters.

At a first glance, we notice considerable similarity between Figs. 4 and 5. We explored this observation further by inspecting the content of a random sample of units from both datasets that are tagged with the same label. For that, we selected 50 commits and 20 issues, i.e. aiming for a representativeness of 10%. In general, we notice that issues contain more information around the cost-related matter at hand, which is expected since they essentially provide a discussion forum. More importantly, we found the extra amount of information to be often related to decision-making around the cost matter. Actors may present hints on the current configuration of the deployment, the alternatives for change, the rationale and even potential feature requests. For example, see the following two samples from different issues showing the depth of information that can be potentially extracted:

> `By having a personal deployment, we are free to experiment and research without any limitation. The drawback is that it implies a cost for the cloud provider. Alternatively, we could imagine a ‘sandbox.qhub.dev‘ or ‘alpha.qhub.dev‘ or whatever`
> *Quansight/qhub (issues: #924)* **label**: `provider`

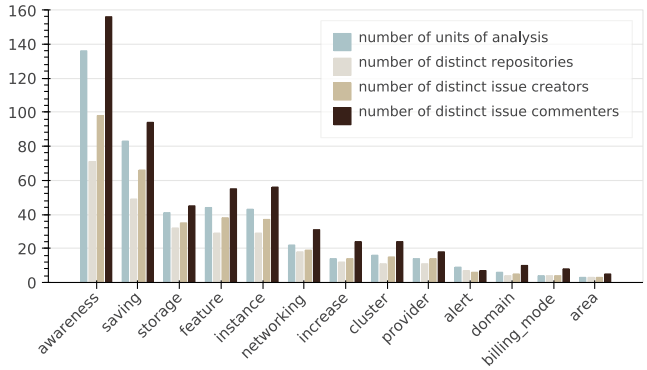


**Fig. 5.** Coding demographics of issues.

> `But the LBs (HTTP(s) and TCP) do not work because they only have the default/main worker pool as target pool, and in my setup its size is 0. So I am kinda paying for Global FW rules that have no use and I cannot delete them because they will get created again in the next ‘terraform apply‘.`
> *poseidon/typhoon (issue: #558)* **label**: `networking`

Thus, while commit messages are commonly more compact and may report cost-changing actions, issues may shed light on the decision-making process that developers undergo before applying a cost-affecting change. An example of this contrast can be seen in the following units both labeled as `instance`:

> `Change code to use the cheaper r4.xlarge instances type.`
> *cisagov/cyhy_amis (hash: 4e67a501)*

> `It would be really great if the new ‘t4g‘ instance, which are even cheaper than ‘t3.nano‘, would be supported as well`
> *Guimove/terraform-aws-bastion (issue: #124)*

While the commit message communicates the change for a cheaper instance, the issue hints on the current configuration and expresses the wish for a new feature (i.e., support of a different instance). We note that this example regards different repositories. We tried to find

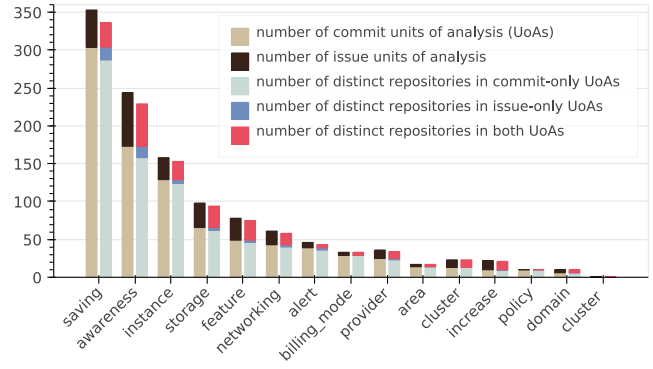


**Fig. 6.** Coding demographics of the combined dataset.

connections between commits and issues but our dataset does not contain any direct links to issue in commit messages. An alternative to explore this avenue is to study pull requests, which is outside of our scope but mentioned in our research agenda (see Section 6).

Continuing with the analysis, we notice some changes in the order of the labels in terms of recurrence. Fig. 6 shows a comparison of the two sets of units and helps to visualize their differences. `awareness` is more recurrent than `saving` among issues, which might be related to our observation that issues can be a more prominent platform for decision-making. On a similar note, units labeled with `increase` are also more recurrent among issues (compared to commit units). This might also be in line with the nature of issues, in this case, reporting or acknowledging cost increase. The example below depicts a situation of a seemingly unintentional increase.

> `Pods for some core services have migrated over to high-memory nodes, which have a much higher cost that the general nodes. I tried killing the pod hoping it would restart on a different node, but usually it just restarts on the same node.`
> *Quansight/qhub (issue: #321)*

From Fig. 6, we also observe that, despite the lower number of repositories among issue-units of analysis, the majority (54 out of 89, and an average of 85% per label) are unique to them. In conclusion, despite the similarities of the assigned labels, the results suggest that the information extracted from issues can complement that from commits. In particular, issues can provide more knowledge about the decision-making surrounding cost management in the deployment of cloud systems configured with IaC.

### 4.3. Knowledge Organization ($RQ_3$)

From the initial coding performed for $RQ_1$ and $RQ_2$, we obtained a set of 14 labels. Moreover, as we delved in the units' content to further understand what kind of information can be extracted, we found evidence of connections and additional relevant themes. For example, in the unit regarding *alphagov/govuk-aws* (hash: 6cfda6ad) presented in Section 4.1, we notice a clear link between `instance` and `saving`, as well as indication of what motivates the change (i.e., more CPU). This particular unit had already been tagged with both labels, but our labels are not fine-grained to the point of providing lower levels of detail.

We note that one main reason for not using finer-grained labels during coding was to avoid explosion of labels that, although meaningful, might not have been ultimately relevant (i.e., not recurrent enough) and could risk the quality of the procedure. Our goal is to guide future research and practitioners' decisions by providing a more generalizable and actionable knowledge based on developers' experience. Thus, we now take a step back to collect more relevant labels and establish connections between them. As described in Section 3.3.3, we started by modeling topics from our dataset, and then proceed to perform axial and selective coding based on the topics and the labels identified in our dataset.

During the investigation of $RQ_2$, we established that the text in both commit messages and issues' content is varied but complementary in nature. So we aimed to create one model per set of units to avoid the risk of not identifying relevant topics. As a result of our topic modeling, we identified 12 topics from commit units and five topics from issue units. The used hyperparameter configurations (see Section 3.3.3) yielded the most promising results, but not all topics were useful for our purposes.

In particular, only one topic derived from issues was considered. The relevant topics mention **cost-related terms** (e.g., 'cheap', 'expensive', 'budget', 'waste') and **actions** (e.g., 'change', 'move', 'add', 'test', 'upgrade') associated with various **properties of the deployment**, both general (e.g., 'VM', 'storage', 'disk', 'machine') and specific (e.g., 'dynamodb', 'CPU', 'NAT', 'EC2', 'RAM'). Some of the connections revealed through the topics are already observable in our coding for $RQ_1$ and $RQ_2$, in the form of co-occurring labels as summarized by Fig. 7. The figure presents these co-occurrence relationships as an UpSet plot (Lex et al., 2014). The labels (with their frequencies) are shown as rows on the bottom part, and the frequency of the various combinations of labels are represented as columns on the top part and described through the connected dots on the bottom part.

We then applied axial and selective coding to aggregate terms (from topics and labels) and inspect the relationships in the units. At the end of this process, we created the knowledge graph depicted in Fig. 8. The graph compiles three main levels of information: effects on cost, actions related to an effect, and the properties of the deployment that are considered for the action. The edges reflect the most significant connections we found and subsequently confirmed in the dataset.

The most recurrent relationships identified during the open coding (for $RQ_1$ and $RQ_2$) are also represented in the graph. Moreover, new links between preexisting labels were found based on topics (e.g., between `increase` and `alert`) and new, more specific, terms appeared (e.g., 'CPU'). We highlight that, although seeing specific terms in one context is what led us to add the associated general label (e.g., `instance` for 'CPU'), the fact that the specific term appears in a topic is a strong indicative that it is prominent. Such particular cases prompted us to include the specific term to graph and connect it to the related terms and the more general one.

After identifying their relationships, some of the topics identified for the previous question can be interpreted as specific pain points that developers are clearly concerned about: choices concerning networking, instance and provider selection, and the applicable billing mode (*property* level in Fig. 8). Other topics signify specific actions taken to address these concerns, or awareness that these actions could/should be taken to avoid unnecessary costs: setting or removing alerts, testing for one or more of these concerns (e.g. the use of VPN), or changing something in the Terraform file towards directly dealing with these concerns (*action* level in Fig. 8). All these actions, both affected and intended to be affected can be characterized based on their desired outcome: increasing awareness, dealing with increasing costs, or produce savings (*effect* level in Fig. 8).

In conclusion, the information aggregated in the knowledge graph serves as a summary of relevant cost-related concerns and actions when deploying cloud-based applications using Terraform. It also introduces the subjects one may expect to find in our dataset in terms of both content and context. Altogether, it opens the door to extract deeper insights and inform research and deployment decisions, as discussed below.

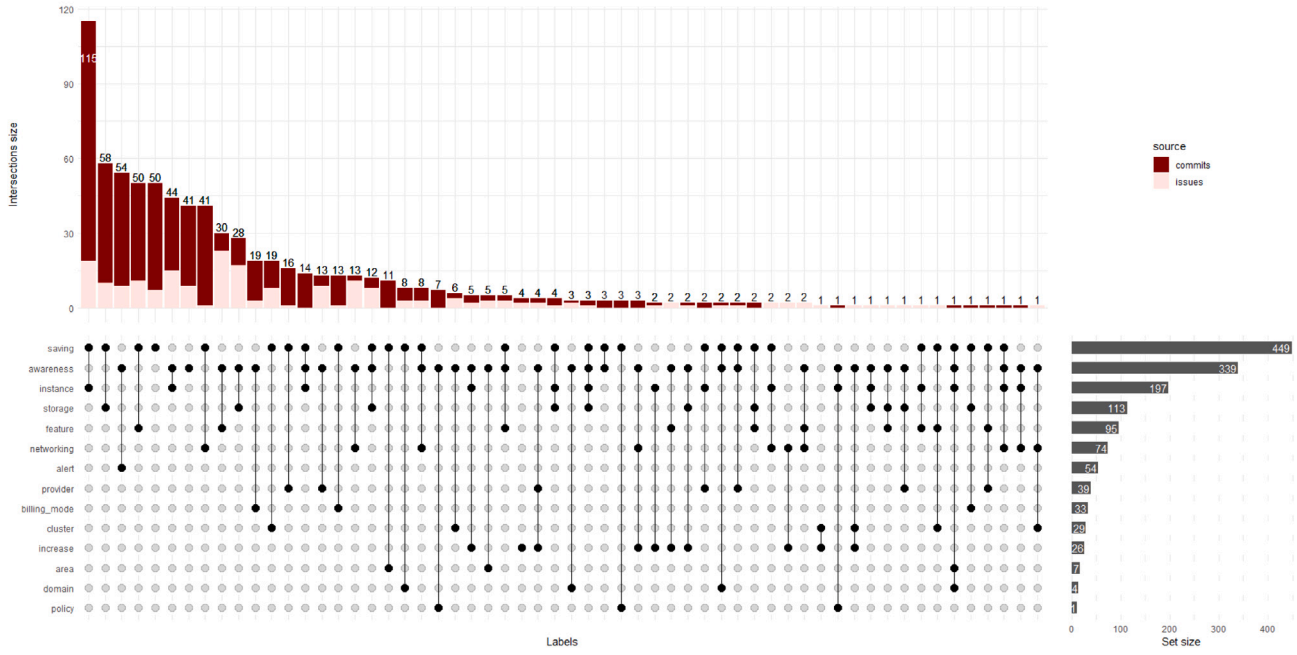


**Fig. 7.** UpSet plot showing the occurrences and co-occurrences of topic labels in commit messages and issues discussions as label intersections resulting from the open coding in $RQ_1$ and $RQ_2$.

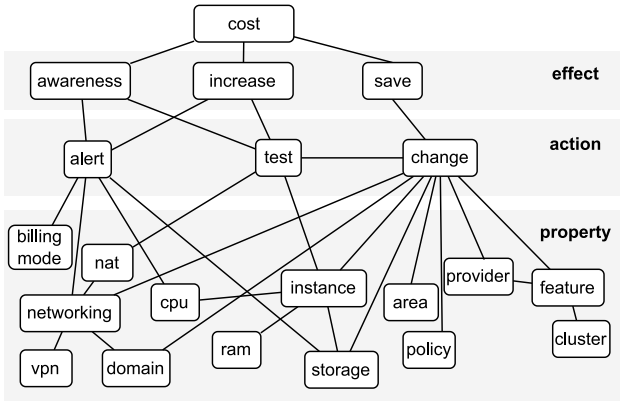


**Fig. 8.** Knowledge graph resulting from the axial and selecting coding; label `save` replaces the label `saving` used in open coding.

## 5. Results triangulation

To verify and strengthen our findings, we sought to investigate the extent to which the identified related topics are also discussed among developers outside GitHub. In particular, in this section, we analyze the discussions of developers in Stack Overflow[12] (SO) and aim at the triangulating the observations from this empirical work with those presented in Section 4.3. We start by presenting the data collection procedure, followed by the performed analysis, and concluding with the results and comparison against our previous observations.

### 5.1. Data collection

The data collection entailed the extraction of relevant discussions from SO. For that, we used key datasets from the September 12, 2023 data dump, namely, posts,[13] comments[14] and change histories,[15] totaling 57.3 GiB. As the goal of this process is to triangulate the results of the study on GitHub repositories, we specifically targeted discussions about Terraform, rooted in questions containing tags with the string 'terraform' in the post's metadata. Thus, we started by filtering questions fitting the mentioned criteria, which led to a dataset of 19 139 questions stored as JSON files.

We gathered the comments and post histories associated with each question and updated the respective JSON file to include them. We then extracted the answers linked to each question, identified through their parent post IDs, and added the comments and post histories similarly to the questions. The answers (and associated data) were then added to the question files, creating the final version of the dataset of 19 139 questions and their associated answers, complete with their respective comments and post histories.

### 5.2. Data analysis

In the analysis, we aimed at investigating whether or not the concepts depicted in Fig. 8 are also present and prominent in SO discussions. For that, we started with filtering cost-related questions by searching the body, title, comments and history of all questions and associated answer(s) for the same keyword stems used for filtering commit messages (see Section 3.3): `bill`, `cheap`, `cost`, `efficient`, `expens`, and `pay`.

Next, we searched the same fields mentioned above of each filtered question for the properties documented in our knowledge graph (see Fig. 8). In this process, we augmented each question JSON file with a list of filtered sentences that contain one or more of these properties. Finally, two of the authors manually inspected a representative sample

[12] https://stackoverflow.com/

[13] https://archive.org/download/stack-exchange-data-dump-2023-09-12/stackoverflow.com-Posts.7z

[14] https://archive.org/download/stack-exchange-data-dump-2023-09-12/stackoverflow.com-Comments.7z

[15] https://archive.org/download/stack-exchange-data-dump-2023-09-12/stackoverflow.com-PostHistory.7z

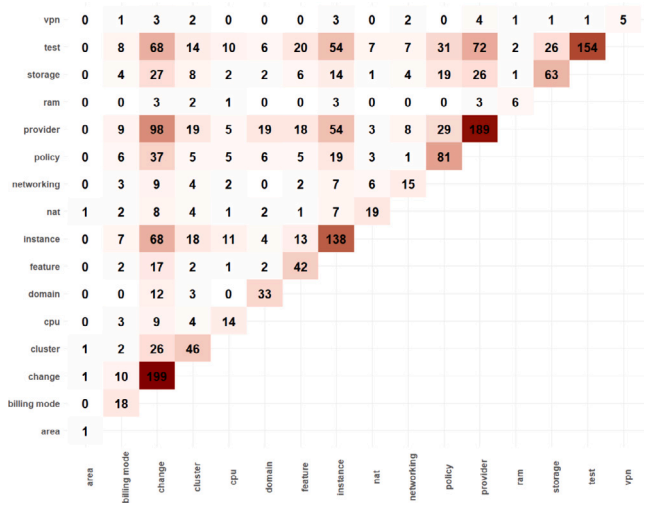


**Fig. 9.** Co-occurrences of action and property concepts from the knowledge graph of Fig. 8 in Stack Overflow questions and their associated answers.

of the questions where properties were found with the goal of establishing the accuracy of this process. We note that the generated dataset and used scripts are publicly available (Feitosa et al., 2024).

### 5.3. Results and discussion

The initial filtering (using keywords stems) returned a total of 491 questions, i.e., approx. 2.6% of all questions on SO with 'terraform' in any of the tags. Although the sample size may seem small compared to the entire population of Terraform-related questions, we note that this is double the sample size compared to that of GitHub repositories containing cost-related commits (1.3%, 2 010 out of 152 735 containing Terraform artifacts; see Section 3.3). This proportional increase may be further indicative of developers' involvement with and attention to cost-related decisions.

Upon inspecting these questions and their answers for actions and properties from our knowledge graph, we learned that 451 (approx. 92%) of them mention one or more of these concepts. In the diagonal of Fig. 9, we show the exact number of questions and associated answers where each concept was present as a term. While properties such as `area` appear only once, other properties such as `provider` and `instance` (189 and 138 occurrences, respectively) but also actions such as `change` (199) and `test` (154) are very popular. The same question and its associated answers might contain a reference to more than one such concept. From these questions' JSON file, we then extracted 3 934 sentences containing one or more of these concepts.

Extracted sentences showcase developers sharing their experiences in dealing with insidious cost-inducing technicalities in using existing services or asking for advice in this direction:

> `Azure by itself discusses the challenge you are addressing and recommends using ""data lifecycle"" for reducing the storage costs. [...] After you enable blob versioning for a storage account, every write operation to a blob in that account results in the creation of a new version.`
> *post id: #72881767* **label**: `storage`

> `I am trying to reduce the cost of my AWS infrastructure deployed using Terraform for a Django app. I have 2 public subnets and 2 private subnets and in the subnets I deploy NAT gateways and elastic ips. all works but is expensive.`
> *post id: #74422443* **label**: `nat`

Developers even identify cost-related features requested to be added to Terraform as an answer to existing problems:

> `Also savings plans have largely replaced reserved instances for now so I had be tracking this issue https://github.com/hashicorp/terraform-provider-aws/issues/10785 if you are interesting in support for using Terraform to manage savings plans.`
> *post id: #48751593* **label**: `provider`

Adding sentences where the initial keywords (i.e. cost, bill, etc.) are mentioned provides additional insights:

> `Since licensing of SQL Server is expensive, I want to switch it off at least for the night.`
> *post id: #76474230* **label**: `expense`

> `I have multiple databases running in each environment which are charging me a lot of cost each month, so i wanted to downscale the DTUs to some lower count during non-working hours, again during working hours DTUs to be upscale back to actual DTUs count, it should happen automatically as per time settings every single day.`
> *post id: #74538381* **label**: `cost`

Some of these discussions point towards the need for architectural refactoring as a way of addressing cost-related concerns:

> `If you need zero cost during idle time, should you go with serverless with new design?`
> *post id: #56530721* **label**: `cost`

Aided by these highlighted sentences, we proceeded to manually inspect a representative sample of 37 questions,[16] considering that we selected 2.6% of a population of 19 139 questions with Terraform tags. During the inspection, we aimed to verify whether: (1) the question indeed contained a cost-related discussion (either as the main problem or as the element of a description or argument), and (2) one of the properties is an essential element of the identified cost-related text. As a result, we found that all inspected questions were true positives. These combined observations show that the knowledge graph cannot only be used to identify actual pain points faced by cloud application developers, but it also points to the most recurrent ones.

Based on the co-occurrences identified in Fig. 9, we proceeded to investigate if and how the knowledge graph could be augmented. We note that the frequency of the observed co-occurrences varies greatly while also being positively skewed, i.e., many low co-occurrences (min.: 1, $Q1$: 2, $Q2$: 5, $Q3$: 12, max.: 98). Following our goal of representing the

[16] https://www.calculator.net/sample-size-calculator.html?type=1&cl=95&ci=5&pp=2.6&ps=491&x=Calculate

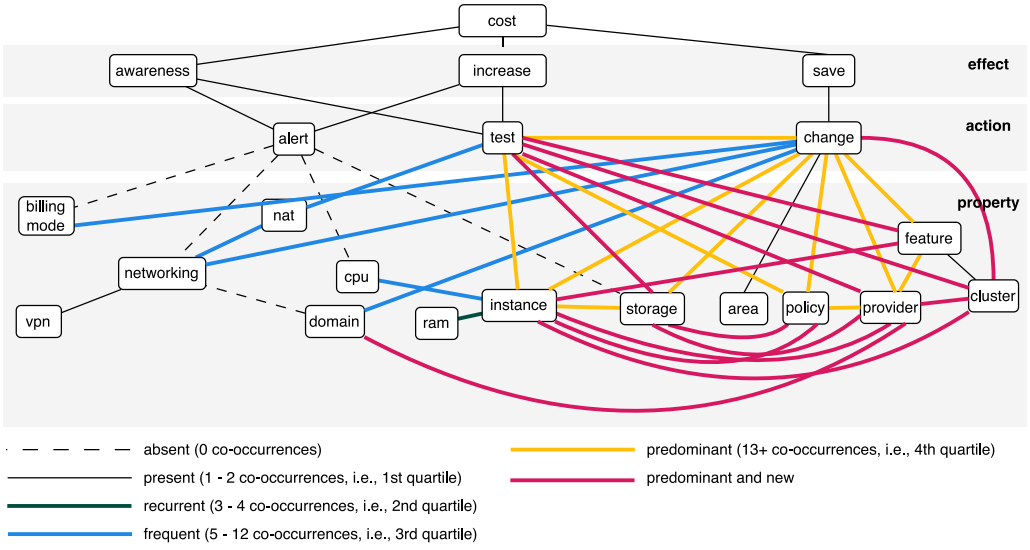


**Fig. 10.** Knowledge graph updated based on the co-occurrences of concepts in the Stack Overflow posts.

most meaningful information in the knowledge graph, we processed the co-occurrence frequencies as follows:

1. mark preexisting graph edges that were not observed as `absent`;
2. classify the observed co-occurrences into `present` ($f \leq 2$), `recurrent` ($2 < f \leq 4$), `frequent` ($4 < f \leq 12$) and `predominant` ($12 < f$) according to the four quartiles;
3. mark preexisting graph edges observed in the data according to their frequency class (established in the previous step); and
4. add new graph edges for predominant co-occurrences that were not present in the previous version.

The resulting knowledge graph is depicted in Fig. 10. We note that the markings explained in the figure legend are aimed at helping the reader visualize the process. The augmented graph comprise all pre-existing edges with the addition of the new edges for predominant co-occurrences. Looking at the differences and co-occurrences frequencies, while many relationships between concepts are also observed in this dataset, we learn two main lessons: (a) the cost-related topics revolve mainly around the ability to try out different configurations (i.e., `test`) and updating configuration (i.e., `change`), also meaning that creating system alerts is not frequently discussed; and (b) discussions are more concrete in the sense that they get more specific about deployment matters such as features and benefits of providers, e.g., appropriate instances and policies. The latter point, in turn, led to a number of new relations being identified between "low level" concepts in the graph, e.g., between `instance` and `policy` or `cluster`.

## 6. Implications & future work

The presented findings have implications for both practitioners and academic researchers. With respect to the former, it becomes clear that cost awareness should actually be present, if it is not already, throughout the development of cloud-based applications. Our dataset contains multiple examples of developers rushing to adapt their deployment configurations to deal with prohibitive costs, or intentionally designing their deployment with the clear intention of avoiding them, particularly when specific cloud services are involved. More importantly, there are specific pain points and actions that can be taken for reducing costs in these cases and lessons learned by other developers to be extracted by investigating our curated dataset. Looking at the questions posted on Stack Overflow on the topic, and the corresponding answers, it becomes clear that having such knowledge in advance at their disposal would help practitioners avoid common mistakes and pitfalls in managing their operational expenses in the cloud.

With respect to the researcher community, our findings demonstrate that there is indeed empirical evidence of software developers being aware of the cost of their choices with respect to deploying their software in the cloud. This evidence is corroborated by the number of posts on Stack Overflow discussing the topics we identified as important for cost awareness. While this conclusion is the product of processing only a specific type of artifacts in open source repositories and user forums, there is no reason to make us believe that there is no further evidence to be uncovered when other types of artifacts, or even the software source code in the identified repositories in the dataset is examined. The findings of this study are therefore a call for further studies on cost awareness during software development.

More specifically, the following research items can be pursued by starting with our existing dataset:

- Provide a finer-grain analysis of the collected evidence looking at e.g. the commit contributors and type of projects involved. Combined, for example, with a practitioner's survey or interviews, it can shed light on how different development teams and organizations deal with cost-related issues.
- Collect and correlate cost awareness evidence from the pull requests (PRs) of the identified repositories. PRs can provide the missing link between commits and issues and offer further insights into the cost management practices.
- Extend the evidence search to other cloud orchestrator solutions, both provider-agnostic (e.g. TOSCA[17]) and -specific ones (AWS CloudFormation[18]), and compare the findings. Especially for the latter, GitHub might not necessarily be the best data source for this purpose, with the repositories of (large) organizations and enterprises with mature cloud presence over the years being much more attractive sources of data.
- Apply natural language processing and other machine learning techniques such as sentiment analysis to gain further insights into the reasoning of the developers. Sentiment analysis in particular has been shown to be a mixed bag when software is concerned (Lin et al., 2022), but with the training of the analyzer focused on cloud orchestrator artifacts and their associated commit messages and issue discussions, it might be possible to get better results through the tighter scope.

[17] https://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html
[18] https://aws.amazon.com/cloudformation/

- ➥ Look for similar evidence in other types of artifacts, for example other configuration files, and/or in closed source repositories, e.g. of large software-intensive organizations and enterprises. This search can go beyond cloud-based software, or at least public cloud deployments, and incorporate also the wealth of DevOps tools available for automating software deployment and management.
- ➥ Look also in the source code of repositories, starting with the ones already containing cost-aware orchestration artifacts as easier to reach targets. This is an obvious extension of the current work, but will need a wide experience with multiple programming languages and platforms on behalf of the researchers.
- ➥ Use the extracted posts from Stack Overflow to extend the list of topics related to cost awareness, and update the knowledge graph of Fig. 8 accordingly.
- ➥ Organize the collected information into reusable knowledge concerning the best practices of managing the operational expenses of cloud-based software. In a sense, this would be the outcome of this research agenda with the most impact to the wider community.

We strongly believe that this is only the first study of many to come on this particular topic.

## 7. Threats to validity

Like any other empirical study in software engineering, this work's validity is also threatened (and these threats mitigated) in several ways. The main threats to this work are discussed in the following.

**External Validity**: Regarding external validity, the population in our data sets (from GitHub and Stack Overflow) may not represent all possible cost-related discussions in cloud projects. The inclusion of repositories hosted in other (closed-source) platforms, or that use other cloud orchestrators, or that use no cloud orchestrators, could lead to the identification of new discussions. The inclusion of discussion forums other than Stack Overflow could also lead to new discussions. However, our decisions were carefully considered with the aim of ensuring data quality, diversity and quantity while keeping the execution feasible for the available human resources. Furthermore, our goal was not to collect all possible evidence, to begin with, but rather to see if there is any evidence available. That said, the results triangulation (between GitHub- and Stack Overflow-based discussions) helps mitigate threats to external validity.

**Construct Validity**: The selection criteria, and most noticeably the defined keywords may threat the construct validity of our work. We mitigated this threat by piloting and testing our selection criteria, and considering the knowledge of both academic and industrial domain experts. Furthermore, the coding activity is naturally open to subjectivity and inconsistencies. To mitigate this threat, we followed a well-established process, and introduced an extra final step to consolidate the knowledge and systematically discuss the labels. Also, the topics derived from applying LDA may not fully represent their content. We mitigate this threat by manually inspecting the units of analysis during the investigation of $RQ_3$, which was part of the axial coding and selective coding. This threat is also mitigated by the triangulation presented in Section 5. That said, we acknowledge that the Stack Overflow study itself suffers from its own threats to construct validity, mainly related to selection of appropriate data points, i.e., pertaining to cost-related discussion of Terraform-based deployments. The mitigation strategies in this case entail the use of Terraform-related tags that have been assigned by users, on top of using previously-validated keywords for searching cost-related entries within the filtered posts.

**Reliability**: Finally, to mitigate threats to the reliability of the study, we have described the data acquisition process in as much detail as possible. More importantly, the dataset curated through our efforts is publicly available (Feitosa et al., 2024) together with the scripts to aid the replication the data collection and topic modeling tasks. This package also includes the dataset and scripts used for the results triangulation with Stack Overflow questions.

## 8. Conclusions

Managing the operational expenses of deploying software in the cloud is a major challenge for organizations. However, how practitioners approach this topic has so far been treated by the literature in an anecdotal and therefore non systematic manner. Consequently, and as a first step, in this work we investigated whether there is (empirical) evidence of software developers being aware of the operational expenses of deploying and delivering software in the cloud, and if yes, then what kind of information can be extracted from this evidence and how this information can be organized for further study.

Given the wide scope and previously unexamined nature of these questions, at least from an MSR perspective, we started tackling them by focusing on cloud orchestrator descriptor files. We chose repository mining as our methodology, and designed and executed the first such study searching for evidence in open source code repositories on GitHub that contain Terraform orchestration artifacts, a very popular provider-agnostic IaC tool.

Our search was shown to be successful, insofar as it actually allowed us to retrieve and organize in a dataset not only evidence of cost awareness by software developers, but also of specific actions being taken as a result. More specifically, with respect to extracting information from commit messages in the selected repositories ($RQ_1$), our findings show that the most popular topics dominating the developer discourse is not limited to being aware of the potential or actual cost of deploying and operating the system in the cloud. Developers seem also to not only be taking concrete actions to minimize this cost, but also to avoid excessive charging to occur when e.g. using specific cloud services and/or offerings within them. Processing cost-related issues from the same repositories ($RQ_2$) did not reveal any additional information in terms of identified topics of discussion. It did however offer further insights into the decision-making process entailed in managing cloud costs that can be pursued further in future work.

Enriching and organizing the extracted information from the previous steps into a knowledge graph ($RQ_3$) helped us identify both higher-level, recurring concepts such as awareness, and specific pain points such CPU and RAM (sizes) in the deployment and operation of cloud-based software that dominate the developers' discussions. The follow-up triangulation with Stack Overflow Terraform discussions that address cost concerns corroborates these pain points and their prevalence in the spectrum of discussion topics.

Finally, based on that evidence and the limitations of this work we developed a list of future research items which both provides us with a clear roadmap for future work, and offers to the wider community an opportunity to develop a new research topic at the intersection of mining software repositories and cloud computing.

## CRediT authorship contribution statement

**Daniel Feitosa:** Writing – original draft, Validation, Supervision, Software, Methodology, Formal analysis, Data Curation, Conceptualization. **Matei-Tudor Penca:** Software, Formal analysis, Data curation. **Massimiliano Berardi:** Software, Formal analysis, Data curation. **Rares-Dorian Boza:** Software, Formal analysis, Data curation. **Vasilios Andrikopoulos:** Writing – original draft, Validation, Supervision, Methodology, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

We have shared the link to my data and code in the manuscript.

## Acknowledgments

The authors would like to thank Diomidis Spinellis for providing the inspiration for this work.

## References

Abdellatif, A., Costa, D., Badran, K., Abdalkareem, R., Shihab, E., 2020. Challenges in chatbot development: A study of stack overflow posts. In: Proceedings of the 17th International Conference on Mining Software Repositories. MSR, ACM, Seoul, Republic of Korea, pp. 174–185. http://dx.doi.org/10.1145/3379597.3387472.

Ahmed, S., Bagherzadeh, M., 2018. What do concurrency developers ask about? A large-scale study using stack overflow. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, Oulu, Finland, http://dx.doi.org/10.1145/3239235.3239524.

Al Alamin, M.A., Malakar, S., Uddin, G., Afroz, S., Haider, T.B., Iqbal, A., 2021. An empirical study of developer discussions on low-code software development challenges. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. MSR, pp. 46–57. http://dx.doi.org/10.1109/MSR52588.2021.00018.

Amato, A., Venticinque, S., 2016. Multiobjective optimization for brokering of multicloud service composition. ACM Trans. Internet Technol. (TOIT) 16 (2), 1–20.

Andrikopoulos, V., Binz, T., Leymann, F., Strauch, S., 2013. How to adapt applications for the cloud environment. Computing 95 (6), 493–535. http://dx.doi.org/10.1007/s00607-012-0248-2.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58.

Arunarani, A., Manjula, D., Sugumaran, V., 2019. Task scheduling techniques in cloud computing: A literature survey. Future Gener. Comput. Syst. 91, 407–415.

Bao, L., Lo, D., Xia, X., Wang, X., Tian, C., 2016. How android app developers manage power consumption? In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, http://dx.doi.org/10.1145/2901739.2901748.

Blei, D.M., 2010. Probabilistic topic models. IEEE Signal Process. Mag. 27, 55–65.

Campbell, J.C., Hindle, A., Stroulia, E., 2015. Latent Dirichlet allocation: Extracting topics from software engineering data. In: The Art and Science of Analyzing Software Data. Elsevier, pp. 139–159. http://dx.doi.org/10.1016/b978-0-12-411519-4.00006-9.

Chauhan, M.A., Babar, M.A., Benatallah, B., 2017. Architecting cloud-enabled systems: A systematic survey of challenges and solutions. Softw. - Pract. Exp. 47 (4), 599–644.

Chen, T.-H., Thomas, S.W., Nagappan, M., Hassan, A.E., 2012. Explaining software defects using topic models. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories. MSR, pp. 189–198. http://dx.doi.org/10.1109/MSR.2012.6224280.

Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A., 2015. Characterization and prediction of issue-related risks in software projects. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE, pp. 280–291.

Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A., Grundy, J., 2018. Predicting delivery capability in iterative software development. IEEE Trans. Softw. Eng. 44 (6), 551–573. http://dx.doi.org/10.1109/tse.2017.2693989.

Cong, P., Xu, G., Wei, T., Li, K., 2020. A survey of profit optimization techniques for cloud providers. ACM Comput. Surv. (CSUR) 53 (2), 1–35.

Corbin, J., Strauss, A., 2014. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. SAGE Publications.

da Costa, D.A., McIntosh, S., Treude, C., Kulesza, U., Hassan, A.E., 2017. The impact of rapid release cycles on the integration delay of fixed issues. Empir. Softw. Eng. 23 (2), 835–904. http://dx.doi.org/10.1007/s10664-017-9548-7.

Das, T., Di Penta, M., Malavolta, I., 2016. A quantitative and qualitative investigation of performance-related commits in android apps. In: 2016 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 443–447.

de Carvalho, L.R., de Araujo, A.P.F., 2020. Performance comparison of terraform and cloudify as multicloud orchestrators. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing. CCGRID, IEEE, http://dx.doi.org/10.1109/ccgrid49817.2020.00-55.

Dueñas, S., Cosentino, V., Robles, G., Gonzalez-Barahona, J.M., 2018. Perceval: Software project data at your will. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings. ICSE-C, ACM, Gothenburg, Sweden, pp. 1–4. http://dx.doi.org/10.1145/3183440.3183475, URL: https://github.com/chaoss/grimoirelab-perceval.

Feitosa, D., Penca, M.-T., Berardi, M., Boza, R.-D., Andrikopoulos, V., 2024. Supplementary material for mining cost awareness in the infrastructure as code artifacts of cloud-based applications. http://dx.doi.org/10.5281/zenodo.11312689.

Han, J., Shihab, E., Wan, Z., Deng, S., Xia, X., 2020. What do programmers discuss about deep learning frameworks. Empir. Softw. Eng. 25 (4), 2694–2747. http://dx.doi.org/10.1007/s10664-020-09819-6.

Harms, R., Yamartino, M., 2010. The economics of the cloud. Microsoft whitepaper, Microsoft Corporation.

Hindle, A., 2013. Green mining: A methodology of relating software change and configuration to power consumption. Empir. Softw. Eng. 20 (2), 374–409. http://dx.doi.org/10.1007/s10664-013-9276-6.

Hindle, A., Ernst, N.A., Godfrey, M.W., Mylopoulos, J., 2011. Automated topic naming to support cross-project analysis of software maintenance activities. In: Proceedings of the 8th Working Conference on Mining Software Repositories. MSR, ACM, Waikiki, Honolulu, HI, USA, pp. 163–172. http://dx.doi.org/10.1145/1985441.1985466.

Hoffman, M., Bach, F., Blei, D., 2010. Online learning for latent Dirichlet allocation. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (Eds.), In: Advances in Neural Information Processing Systems, vol. 23, Curran Associates, Inc..

Hosseinzadeh, M., Hama, H.K., Ghafour, M.Y., Masdari, M., Ahmed, O.H., Khezri, H., 2020. Service selection using multi-criteria decision making: A comprehensive overview. J. Netw. Syst. Manage. 28 (4), 1639–1693.

Howard, M.J., Gupta, S., Pollock, L., Vijay-Shanker, K., 2013. Automatically mining software-based, semantically-similar words from comment-code mappings. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE, pp. 377–386.

Jamshidi, P., Ahmad, A., Pahl, C., 2013. Cloud migration research: A systematic review. IEEE Trans. Cloud Comput. 1 (2), 142–157.

Kovács, J., Kacsuk, P., 2017. Occopus: A multi-cloud orchestrator to deploy and manage complex scientific infrastructures. J. Grid Comput. 16 (1), 19–37. http://dx.doi.org/10.1007/s10723-017-9421-3.

LeClair, A., Eberhart, Z., McMillan, C., 2018. Adapting neural text classification for improved software categorization. In: 2018 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, http://dx.doi.org/10.1109/icsme.2018.00056.

Lex, A., Gehlenborg, N., Strobelt, H., Vuillemot, R., Pfister, H., 2014. UpSet: Visualization of intersecting sets. IEEE Trans. Visual. Comput. Graph. 20 (12), 1983–1992. http://dx.doi.org/10.1109/tvcg.2014.2346248.

Lin, B., Cassee, N., Serebrenik, A., Bavota, G., Novielli, N., Lanza, M., 2022. Opinion mining for software development: A systematic literature review. ACM Trans. Softw. Eng. Methodol. 31 (3), http://dx.doi.org/10.1145/3490388.

Mell, P., Grance, T., et al., 2011. The NIST definition of cloud computing. NIST Special Publication 800-145.

Moura, I., Pinto, G., Ebert, F., Castor, F., 2015. Mining energy-aware commits. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, http://dx.doi.org/10.1109/msr.2015.13.

Pereira, R., Matalonga, H., Couto, M., Castor, F., Cabral, B., Carvalho, P., de Sousa, S.M., Fernandes, J.P., 2021. GreenHub: A large-scale collaborative dataset to battery consumption analysis of android devices. Empir. Softw. Eng. 26 (3), http://dx.doi.org/10.1007/s10664-020-09925-5.

Pinto, G., Castor, F., Liu, Y.D., 2014. Mining questions about software energy consumption. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 22–31.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., Manning, C.D., 2020. Stanza: A Python natural language processing toolkit for many human languages. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations.

Reboucas, M., Pinto, G., Ebert, F., Torres, W., Serebrenik, A., Castor, F., 2016. An empirical study on the usage of the Swift programming language. In: Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. SANER, IEEE, http://dx.doi.org/10.1109/saner.2016.66.

Sas, C., Capiluppi, A., 2022. Antipatterns in software classification taxonomies. J. Syst. Softw. 190, 111343. http://dx.doi.org/10.1016/j.jss.2022.111343.

Shuaib, M., Samad, A., Alam, S., Siddiqui, S.T., 2019. Why adopting cloud is still a challenge?—A review on issues and challenges for cloud migration in organizations. In: Ambient Communications and Computer Systems. Springer, pp. 387–399.

Spadini, D., Aniche, M., Bacchelli, A., 2018. PyDriller: Python framework for mining software repositories. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2018, ACM, New York, New York, USA, pp. 908–911. http://dx.doi.org/10.1145/3236024.3264598, URL: https://github.com/ishepard/pydriller.

Tomarchio, O., Calcaterra, D., Modica, G.D., 2020. Cloud resource orchestration in the multi-cloud landscape: A systematic review of existing frameworks. J. Cloud Comput. 9 (1), 1–24.

Treude, C., Wagner, M., 2019. Predicting good configurations for GitHub and stack overflow topic models. In: Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, http://dx.doi.org/10.1109/msr.2019.00022.

Tricomi, G., Merlino, G., Panarello, A., Puliafito, A., 2020. Optimal selection techniques for cloud service providers. IEEE Access 8, 203591–203618.

Vakili, A., Navimipour, N.J., 2017. Comprehensive and systematic review of the service composition mechanisms in the cloud environments. J. Netw. Comput. Appl. 81, 24–36.

Řehůřek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA, Valletta, Malta, pp. 45–50.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. In: Computer Science, Springer Berlin, Heidelberg.

Zimmerle, C., Gama, K., Castor, F., Filho, J.M.M., 2022. Mining the usage of reactive programming APIs: A study on GitHub and stack overflow. In: Proceedings of the 19th International Conference on Mining Software Repositories. MSR, ACM, Pittsburgh, Pennsylvania, pp. 203–214. http://dx.doi.org/10.1145/3524842.3527966.

**Daniel Feitosa** is an assistant professor at the University of Groningen, where he contributes to the Software Engineering and Architecture group within the Bernoulli Institute for Mathematics, Computer Science, and Artificial Intelligence. Earning his Ph.D. from the same university, Daniel's research interests are rooted in software quality, mining software repositories, and developer experience, especially applied to technical debt management, and energy efficiency. He has participated in EU-funded project, such as SDK4ED, and is an active member of various software engineering communities, having served as a program committee member in several conferences and referee for multiple journals.

**Matei Tudor Penca** is a Data Scientist at Elsevier, a scientific publisher and data analytics company, where he contributes to the Data Science Life Sciences department of the Amsterdam team. He holds a Master's degree in Information Studies: Data Science from the University of Amsterdam. His expertise lies in natural language processing, specifically in LLM agents, information retrieval and data mining within the life sciences domain. Matei is committed to integrating the latest research technologies for optimized data-driven applications.

**Massimiliano Berardi** is a recent graduate of Computing Science from the University of Groningen, where he also contributed as a teaching assistant for various courses. He is currently working as a Software Engineer at Syntho, a startup specializing in synthetic data generation. His expertise in database management and software architecture is employed in improving their product and growing as a company.

**Rareş-Dorian Boza** is pursuing an M.Sc. in Computer Science at Delft University of Technology, having previously earned his B.Sc. from the University of Groningen, alongside the completion of the Honours College programme. Currently, he is delving into natural language processing solutions, software architecture, and information retrieval, specifically in the use case of recommender systems. Alongside academia, Rareş is engaged as a software tooling intern for Intel's department in Eindhoven.

**Vasilios Andrikopoulos** is associate professor at the University of Groningen, and a member of the Software Engineering and Architecture group in the Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence. He received his Ph.D. from Tilburg University, the Netherlands, and he has worked as a post-doc for both Tilburg University and the University of Stuttgart, Germany. His research interests are in software architectures for cloud-based systems, with an emphasis on their sustainability across all its dimensions. He has participated in a number of EU-funded projects, including the Network of Excellence S-Cube, and is reviewing for a multitude of journals.