The PIPr Dataset of Public Infrastructure as Code Programs

Daniel Sokolowski
daniel.sokolowski@unisg.ch
University of St. Gallen
Switzerland

David Spielmann
david.spielmann@unisg.ch
University of St. Gallen
Switzerland

Guido Salvaneschi
guido.salvaneschi@unisg.ch
University of St. Gallen
Switzerland

ABSTRACT

With Programming Languages Infrastructure as Code (PL-IaC), developers implement IaC programs in popular imperative programming languages like Python and Typescript. Such programs generate the declarative target state of the deployment, i.e., they describe what to set up, not how to set it up. Despite the popularity of PL-IaC, which has grown more than ten times from 2020 to 2023, we know little about how developers apply it and how IaC programs differ from other software. Such knowledge is essential to effectively use existing software engineering techniques and develop new ones for PL-IaC. To shed light on PL-IaC in practice, we present PIPr, the first systematic PL-IaC dataset. PIPr is based on 37 712 public IaC programs on GitHub from August 2022 and includes initial analyses, assessing the programming languages, testing techniques, and licenses of the IaC programs. Beyond the metadata and analysis results of all IaC programs, PIPr contains the code of all 15 504 IaC programs whose licenses permit redistribution. PIPr sets the ground for future in-depth investigations on PL-IaC in practice.

CCS CONCEPTS

• **Software and its engineering → Architecture description languages**; **Software configuration management and version control systems**; Cloud computing.

KEYWORDS

Infrastructure as Code, Pulumi, AWS CDK, CDKTF, Testing

ACM Reference Format:
Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2024. The PIPr Dataset of Public Infrastructure as Code Programs. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643991.3644888

1 INTRODUCTION

Infrastructure as Code (IaC) [22] solutions use machine-readable code for software configuration and deployment. Configuration-management-focused solutions, e.g., Ansible [38], Chef [28], and Puppet [33], and cloud-provisioning-focused solutions, e.g., AWS CloudFormation [3] and Terraform [16], have been developed. Typically, developers write IaC scripts in JSON and YAML configuration languages, which is tedious for big and complex deployments and has led to DSLs like HCL [15] and Bicep [21].

Listing 1: Example Pulumi IaC program in TypeScript and Python deploying a static website on AWS S3.[2]

```typescript
import * as aws from "@pulumi/aws";

const bucket =
  new aws.s3.Bucket("bucket", {
    website: {
      indexDocument: "index.html",
    }
  });
new aws.s3.BucketObject("index", {
  bucket: bucket,
  content:
    "<!DOCTYPE html>Hello world!",
  key: "index.html",
  contentType: "text/html",
});

export const url =
    bucket.websiteEndpoint;
```

```python
import pulumi
import pulumi_aws as aws

bucket = aws.s3.Bucket("bucket",
  website=aws.s3.BucketWebsiteArgs(
    index_document="index.html"
  )
)

aws.s3.BucketObject("index",
  bucket=bucket,
  content=
    "<!DOCTYPE html>Hello world!",
  key="index.html",
  content_type="text/html"
)

pulumi.export("url",
      bucket.website_endpoint)
```

Instead of new DSLs, Programming Languages IaC (PL-IaC) leverage general-purpose programming languages like Python and TypeScript to tackle the complexity of deployments. With PL-IaC, instead of IaC *scripts*, developers write IaC *programs* using any available (imperative) feature of the programming languages they already know, and the PL-IaC solution ensures declerativity, i.e., developers only express the intended target state, not how to achieve it. For example, Listing 1 shows the deployment of a simple static website on AWS S3 in Pulumi TypeScript and Python. Both versions are simple imperative programs that define two resources in the declarative target state: the bucket and the bucket object hosted in it. PL-IaC is production-ready with Pulumi [31], AWS CDK [2], and CDKTF [14], and its popularity is increasing. Pulumi reported a ~10× growth to 150 000 end users from 2020 to 2023 [8, 9], and the total annual downloads of all solutions' core packages on NPM grew by ~15× in this time.[1]

As IaC programs are very close to traditional software and use the same surface languages, there is great potential to apply existing software engineering methods—testing, verification, and static analysis, to mention a few. Yet, we know little about IaC programs and their differences from other software. Such insights are imperative to apply existing techniques effectively and to develop new approaches that leverage the peculiarities of this domain.

To shed light on PL-IaC in practice, we present PIPr, the first dataset of IaC programs. PIPr comprises metadata of 37 712 IaC programs from 21 445 public GitHub repositories and shallow copies (i.e., without history) of the ones permitting redistribution. As the case studies of the dataset, we inspect IaC programs for their

MSR '24, April 15–16, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0587-8/24/04.
https://doi.org/10.1145/3643991.3644888

[1]https://npm-stat.com/ for aws-cdk-lib, @aws-cdk/core, cdktf, and @pulumi/pulumi.
[2]Ownership, access, and policy resources for internet access are excluded for simplicity.

(1) programming languages (Section 4.1), (2) testing techniques (Section 4.2), and (3) licenses (Section 4.3).

PIPr, including all scripts, is long-time archived under the Open Data Commons Attribution License (ODC-By) v1.0 on Zenodo [42]. The dataset builds the ground for future studies on PL-IaC in practice and the differences and similarities between such programs and traditional software. Investigating these issues is crucial to transferring existing software engineering techniques and developing new ones that are optimized for PL-IaC.

2 RELATED WORK AND DATASETS

All PL-IaC solutions provide example programs.[3][4][5] They are public on GitHub with explicit open-source licenses. PIPr contains their metadata and code from August 2022. Beyond these, the only datasets of IaC programs we know were created by us to evaluate µs [43]. They contain 64 Pulumi TypeScript programs using stack references [44] and simple benchmarking programs [45] in Pulumi TypeScript, AWS CDK TypeScript, and µs. We are not aware of other research analyzing IaC programs for PL-IaC solutions.

Various studies examined IaC scripts for Ansible, Chef, and Puppet. Most—despite some claiming the opposite—published at most their analysis scripts and results, but not all IaC scripts they analyzed [7, 18, 19, 24–27, 34–37, 39, 41, 47]. In contrast, Sotiropoulos et al. [46] provided the 33 studied Puppet scripts, and Saavedra and Ferreira [40] published the most comprehensive dataset of 108 509 Ansible, 70 939 Chef, and 17 037 Puppet scripts, containing unpublished IaC scripts of earlier studies [35, 37]. Further, Opdebeeck et al. [24] built a dataset of Ansible Galaxy ecosystem metadata, including abstract structural representations of over 125 000 Ansible roles and 800 000 changes.

PIPr is the first systematic PL-IaC dataset. It provides the code of 15 504 redistributable IaC Programs and metadata of all 37 712 IaC programs we found on GitHub in August 2022, including the programs' programming languages, testing techniques, and licenses.

3 DATASET CONSTRUCTION

Figure 1 provides an overview of PIPr's construction, analysis, and distribution, which we now describe in detail.

3.1 Repository Identification

We searched for platforms hosting AWS CDK, Pulumi, and CDKTF programs—the only three established PL-IaC solutions. We chose GitHub as the data source of PIPr because we did not find another platform publicly hosting many IaC programs—even AWS CDK, Pulumi, and CDKTF themselves are publicly maintained on GitHub.

We identified GitHub repositories with IaC programs by searching for IaC program configuration files. By design, in PL-IaC solutions, each IaC program has one configuration file named `cdk.json`, `cdktf.json`, `Pulumi.yml`, or `Pulumi.yaml`. Munaiah et al. [23] summarized techniques to query GitHub, including Boa [10] and the discontinued GHTorrent [13]. We used the GitHub REST Code Search API [11]—despite having to cope with its severe limitations

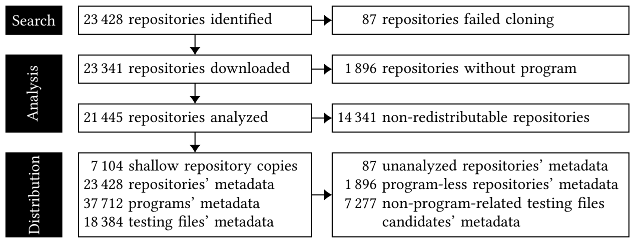

Search
23 428 repositories identified
87 repositories failed cloning
Analysis
23 341 repositories downloaded
1 896 repositories without program
21 445 repositories analyzed
14 341 non-redistributable repositories
Distribution
7 104 shallow repository copies
23 428 repositories' metadata
37 712 programs' metadata
18 384 testing files' metadata
87 unanalyzed repositories' metadata
1 896 program-less repositories' metadata
7 277 non-program-related testing files candidates' metadata

Figure 1: Flow diagram quantifying the dataset creation.

that we will discuss next—because it is the only solution that allows up-to-date searches for file names across GitHub.

Our study addresses the API's limitations [11]. First, the API only returns files in repositories that (i) are not a fork or a fork with more stars than their parent, (ii) have fewer than 500 000 files, and (iii) saw activity or were returned in search results in the last year. Further, the searched files must be (iv) on the default branch (v) and smaller than 384 KB. Inheriting these criteria ensures we do not analyze irrelevant (i, iii, iv) and technically not tractable (ii, v) repositories. Second, the API's results are incomplete and unstable: it (a) only returns up to 1 000 results per query, (b) returns varying query result counts across responses, and (c) returns varying results for the same result page that are often fewer than expected and repeating results from previous pages. To address (a), we recursively divided the queries by file size until each sub-query has at most 990 results; for (b), we only accepted a changed result count if we received it five consecutive times; for (c), we requested the pages with default size (max. 30 results) up to 200 times until the combined responses contained the expected number of *new* results.

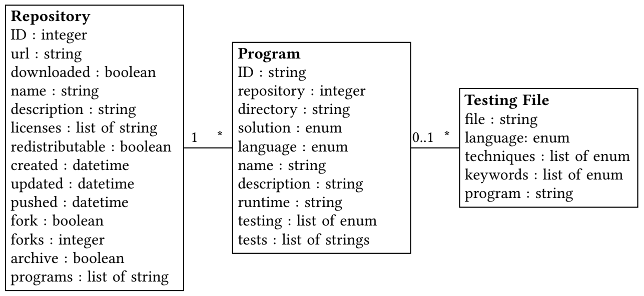
The search started on August 16, 2022, and was distributed over two GitHub accounts. It required two weeks due to retrying upon incomplete results and API rate limiting. We recorded the metadata for all 23 428 identified repositories. We downloaded a shallow recursive copy of each repository's default branch using `git clone` on August 31 and September 1, 2022. 87 repositories could not be downloaded (one retry), most of them due to missing permissions.

3.2 Repository Analysis

In the analysis, we identified all IaC programs in the downloaded repositories. We searched all `cdk.json`, `cdktf.json`, `Pulumi.yml`, and `Pulumi.yaml` files that are not in a `node_modules` folder, yielding 39 255 files. 105 are not parsable, and 892 do not contain a certain set of fields, i.e., they do not define the IaC program's *runtime*, which is the only information PL-IaC solutions require to run an IaC program. For AWS CDK, this means there is no `app` field; for Pulumi, there is no `runtime` field; and for CDKTF, there is neither `app` nor `language`. Further, we removed 469 files whose path contains a string from a denylist of 28 entries. The list was manually created to identify dependency and PL-IaC solution implementation paths and 77 files due to implausible runtime values. For the remaining 37 712 IaC programs, we extracted the runtime, the PL-IaC solution, and, if present, the program name and description.

As the first applications of this dataset, we analyzed all IaC programs for their programming language, testing techniques, and licenses (Section 4). The results are included in PIPr.

[3]https://github.com/aws-samples/aws-cdk-examples
[4]https://github.com/pulumi/examples
[5]https://developer.hashicorp.com/terraform/cdktf/examples-and-guides/examples



Solutions: AWS CDK | CDKTF | Pulumi
Languages: csharp | go | haskell | java | javascript | python | typescript | yaml
Testing & Techniques: awscdk | awscdk_assert | awscdk_snapshot | cdktf | cdktf_snapshot | cdktf_tf | pulumi_crossguard | pulumi_integration | pulumi_unit | pulumi_unit_mocking
Keywords: /go/auto | /testing/integration | @AfterAll | @BeforeAll | @Test | @aws-cdk | @aws-cdk/assert | pulumi.runtime.test | @pulumi/ | @pulumi/policy | @pulumi/pulumi/automation | Amazon.CDK | Amazon.CDK.Assertions | Assertions_ | HashiCorp.Cdktf | IMocks | Moq | NUnit | PolicyPack(| ProgramTest | Pulumi | Pulumi.Automation | PulumiTest | ResourceValidationArgs | ResourceValidationPolicy | SnapshotTest() | StackValidationPolicy | Testing | Testing_ToBeValidTerraform(| ToBeValidTerraform(| Verifier.Verify(| WithMocks(| [Fact] | [TestClass] | [TestFixture] | [TestMethod] | [Test] | afterAll(| assertions | automation | aws-cdk-lib | aws-cdk-lib/assert | aws_cdk | aws_cdk.assertions | awscdk | beforeAll(| cdktf | com.pulumi | def test_ | describe(| github.com/aws/aws-cdk-go/awscdk | github.com/hashicorp/terraform-cdk-go/cdktf | github.com/pulumi/pulumi | integration | junit | pulumi | pulumi.runtime.setMocks(| pulumi.runtime.set_mocks(| pulumi_policy | pytest | setMocks(| set_mocks(| snapshot | software.amazon.awscdk.assertions | stretchr | test(| testing | toBeValidTerraform(| toMatchInlineSnapshot(| toMatchSnapshot(| to_be_valid_terraform(| unittest | withMocks(

Figure 2: Schema of the PIPr metadata and results.

3.3 Distribution

PIPr [42] is long-term archived on Zenodo. Figure 2 describes the schema of the CSV files `repositories.csv`, `programs.csv`, and `testing-files.csv`, which contain the metadata of:

- 23 428 repositories (Section 3.1), of which 87 failed to download, and in 21 445 (1 896), we found (not) an IaC program (Section 3.2).
- 37 712 IaC programs (Section 3.2) with their programming language (Section 4.1), testing (Section 4.2), and licenses (Section 4.3).
- 18 384 testing file candidates (Section 4.2), of which we identified a testing technique in 13 631 files; 11 107 in an IaC program.

Further, PIPr contains copies of 7 104 repositories with IaC programs (58 GB of code). Lastly, we release all creation and analysis scripts, execution logs, and additional documentation.

4 CASE STUDIES

To showcase the use of PIPr, we answer these research questions:
RQ1: Which programming languages are used in IaC programs?
RQ2: Which testing techniques are employed in IaC programs?
RQ3: Which licenses are applied to public IaC programs?

4.1 Languages of IaC Programs

To answer RQ1, we map each IaC program's runtime configuration value to a programming language using regular expressions. For less popular languages where the IaC program reuses the runtime of another language, we assign the program to the main language of the runtime because identifying them based on the runtime configuration is not reliable and would lead to additional insignificant minorities with low confidence in the results (Table 1). We map all non-TypeScript NodeJS, e.g., CoffeeScript and Scala.js, programs to JavaScript, JVM programs to Java, and .NET programs to C#.

Table 1: IaC programs on GitHub by solution and language.

Language	Pulumi	AWS CDK	CDKTF	Total
TypeScript	6 081	14 639	525	21 245
Python	2 927	5 521	162	8 610
C#	1 835	563	28	2 426
Go	1 834	338	73	2 245
JavaScript	35	1 844	5	1 884
Java	75	1 035	34	1 144
YAML	157	0	0	157
Haskell	1	0	0	1
Total	12 945	23 940	827	37 712

Table 1 summarizes the results. TypeScript is, with 56 % (21 245) of the IaC programs, by far the most popular language across all PL-IaC solutions. Python is popular, too, with 23 % (8 610). Further, there is a significant amount of Pulumi C# and Go programs (each 14 % of the Pulumi programs) and AWS CDK JavaScript and Java programs (8 % and 4 % of the AWS CDK programs).

4.2 Testing Techniques of IaC Programs

To answer RQ2, we first researched the available testing techniques for IaC programs. All PL-IaC solutions support unit testing [4, 17, 32]. Additionally, Pulumi features policy testing with CrossGuard [30] and an integration testing library [29].

Our method to identify testing files in repositories is similar to other studies [5, 20, 23]. We created a list of 34 keywords that are specific to PL-IaC testing code (e.g., `@aws-cdk/assert` and `setMocks()`, 14 common keywords in PL-IaC code (e.g., `@aws-cdk` and `@pulumi/`), and 26 common keywords in testing code (e.g., `describe(` and `test()`. Potential PL-IaC testing files are those that contain at least one PL-IaC-testing-specific keyword or one out of 72 combinations of the remaining keywords in non-comment lines, as well as all files that are named `PulumiPolicy.yaml`. We found 65 156 files and extracted their path and keywords. We manually inspected this data and the full content of some files to (1) identify which file extensions to ignore, (2) validate and apply the file path filtering we applied in Section 3.2, and (3) create a function that maps to a testing technique the remaining 18 384 files based on their keywords and file extensions. We identified a testing technique in 13 631 files and related 11 107 files to an IaC program based on their file path by matching the nearest IaC program in a parent folder.

Table 2 summarizes the results. For each technique, it shows the number of files in PIPr and IaC programs containing one. We report adoption in absolute numbers and relative to all programs of the respective PL-IaC solution and language, e.g., 1 % (118) of the Pulumi programs use unit testing, 51 of them are in TypeScript (1 % of the Pulumi TypeScript programs), of which 38 use runtime mocking. Only 25 % of the PL-IaC programs implement tests. Further, testing is far more common for CDK programs (38 % for AWS CDK and 15 % for CDKTF). Only 1 % of the Pulumi programs implement tests.

4.3 Licenses of IaC Programs

To answer RQ3, we applied Licensee [1] to all repositories. We chose Licensee because GitHub recommends and uses it [12]. Licensee analyses all files commonly containing license information, e.g., `LICENSE` and `README`, for license content or references to licenses, using the license database of https://choosealicense.com/.

Table 2: Number of IaC programs applying testing techniques in total and by language. # programs (% of programs in group).

	Testing Technique	Files	Total		TypeScript / Go		Python / JavaScript		C# / Java	
Pulumi	Unit Testing	259	118	(1 %)	51	(1 %)	27	(1 %)	22	(1 %)
					15	(1 %)	0	(0 %)	3	(4 %)
	with Runtime Mocking	149	100	(1 %)	38	(1 %)	26	(1 %)	20	(1 %)
					15	(1 %)	0	(0 %)	1	(1 %)
	CrossGuard	399	33	(0 %)	29	(0 %)	4	(0 %)	0	(0 %)
					0	(0 %)	0	(0 %)	0	(0 %)
	Integration Testing	677	22	(0 %)	12	(0 %)	8	(0 %)	0	(0 %)
					2	(0 %)	0	(0 %)	0	(0 %)
AWS CDK	Unit Testing	12 102	9 152	(38 %)	7 116	(49 %)	1 141	(21 %)	12	(2 %)
					151	(45 %)	328	(18 %)	404	(39 %)
	with AWS CDK Assertions	10 436	8 161	(34 %)	6 967	(48 %)	772	(14 %)	11	(2 %)
					40	(12 %)	320	(17 %)	51	(5 %)
	with Snapshot Testing	1 338	819	(3 %)	788	(5 %)	10	(0 %)	0	(0 %)
					0	(0 %)	13	(1 %)	8	(1 %)
CDKTF	Unit Testing	194	121	(15 %)	81	(15 %)	21	(13 %)	4	(14 %)
					10	(14 %)	0	(0 %)	5	(15 %)
	with Snapshot Testing	80	36	(4 %)	29	(6 %)	1	(1 %)	2	(7 %)
					2	(3 %)	0	(0 %)	2	(6 %)
	with Terraform Compatibility	23	23	(3 %)	14	(3 %)	1	(1 %)	1	(4 %)
					6	(8 %)	0	(0 %)	1	(3 %)
Total		13 631	9 435	(25 %)	7 284	(34 %)	1 196	(14 %)	38	(2 %)
					177	(8 %)	328	(17 %)	412	(36 %)

Table 3: Redistributable IaC programs by PL-IaC solution and language. # programs with license permitting redistribution (% of programs with any license) of which #T use testing.

Language	Pulumi		AWS CDK		CDKTF		Total	
TypeScript	3 084 (51 %)	53 T	5 131 (35 %)	2 230 T	401 (76 %)	60 T	8 616 (41 %)	2 343 T
Python	1 201 (41 %)	18 T	2 085 (38 %)	264 T	64 (40 %)	11 T	3 350 (39 %)	293 T
C#	633 (34 %)	19 T	299 (53 %)	10 T	13 (46 %)	3 T	945 (39 %)	32 T
Go	624 (34 %)	7 T	193 (57 %)	88 T	25 (34 %)	7 T	842 (38 %)	102 T
JavaScript	29 (83 %)	0 T	1 092 (59 %)	115 T	4 (80 %)	0 T	1 125 (60 %)	115 T
Java	49 (65 %)	3 T	442 (43 %)	166 T	16 (47 %)	4 T	507 (44 %)	173 T
YAML	119 (76 %)	0 T	0 (–)	0 T	0 (–)	0 T	119 (76 %)	0 T
Haskell	0 (0 %)	0 T	0 (–)	0 T	0 (–)	0 T	0 (0 %)	0 T
Total	5 739 (44 %)	100 T	9 242 (39 %)	2 873 T	523 (63 %)	85 T	15 504 (41 %)	3 058 T

We did not find a license in 67 % (14 330) of the repositories. The most popular are MIT and Apache 2.0, which are used by 3 545 (17 %) and 1 988 (9 %) repositories. Only 11 repositories prohibit redistribution explicitly. In PIPr, we redistribute all repositories with an IaC program that have at least one and only licenses permitting redistribution. Table 3 shows the redistributed IaC programs by PL-IaC solution, language, and how many use testing, e.g., we provide 51 % (3 084) of the Pulumi TypeScript programs; 53 of them test.

5 LIMITATIONS AND THREATS TO VALIDITY

PIPr is limited to the CDKs and Pulumi and the testing techniques provided by the PL-IaC solutions. We carefully researched other PL-IaC solutions and testing techniques but found none. Further, PIPr is based on a snapshot in August 2022 and does not contain historical data, limiting its direct use for longitudinal studies.

The internal validity of studies on PIPr may be impacted by using the GitHub REST Code Search API, inheriting its inclusion criteria that eliminate old projects and forks. The API also prevents reproduction, as new data is continuously added and old, unpopular results are removed, on top of its reliability issues (cf. Section 3.1). Another threat is caused by identifying PL-IaC programs based on project file names. To mitigate this issue, we ensured that the files are valid by parsing them and checking their content for plausibility. The identification of IaC programs may be impacted by our list of exclusion file path fragments to filter. This aspect also applies to the testing file identification for RQ2, which is further threatened by keyword searches and manually selecting the keywords. Also, mapping testing files to projects based on their file paths may cause mistakes because we systematically miss if testing files are managed separately; however, we could not find that this is common. Relatedly, the internal validity for RQ2 may be threatened because the project may still use the testing technique even if we did not find a testing file in an IaC program. The identification of the programming language in RQ1 relies on regular expressions. The results are limited by mapping some languages to the primary language of their runtime, e.g., Scala to Java. Lastly, for RQ3, we inherit the limitations and validity constraints of Licensee [1].

A threat to the external validity is that we only analyze public projects on GitHub, which may not be generally representative, e.g., for proprietary software. Further, PL-IaC solutions and their use evolve quickly, and we capture data up to August 2022 that was retrievable through the GitHub REST Code Search API; therefore, PIPr may not generalize to older and recent PL-IaC.

6 USE CASES

PIPr is the first systematic dataset for PL-IaC and a suitable basis for future studies on (1) PL-IaC itself, (2) its relation to IaC approaches, and (3) its relation to software in general. All three areas are important to enhance the development of IaC programs, while (2) and (3) are imperative to applying and specializing existing techniques and tools to PL-IaC. We now present concrete research ideas.

Assessing the code of the IaC programs in PIPr helps to understand the nature and issues of PL-IaC programs. Researchers can apply static and dynamic analyses and benchmark new linters, libraries, and language improvements to detect and mitigate common error sources.

Assessing the target state of IaC programs in PIPr sheds light on the configuration errors of (syntactically) valid IaC programs. For example, researchers can generate target states by running the programs and checking them for correctness against oracles, like CVEs and best practices. Further, one can trace errors back to their origin in the imperative program to detect the issues to address.

Replicating IaC studies (cf. Section 2) for PL-IaC on PIPr enables synergies with research on other IaC technologies. Researchers can explore, e.g., whether known IaC code smells exist in IaC programs, if current linters are effective [26, 35, 36, 39–41], and the applicability of IaC code quality metrics [6, 7].

Studying software engineering processes like testing and reviewing for IaC programs highlights inefficiencies in applying PL-IaC within software organizations. Researchers can combine PIPr with additional data, e.g., pull requests and issues from GitHub and Jira, for longitudinal studies.

Comparing PL-IaC with other software contributes to understanding the differences between PL-IaC and traditional applications, effectively enabling a vast array of existing software engineering knowledge for IaC programs. Researchers can compare insights from the longitudinal studies above and, e.g., code metrics, used language features (distributions), and evolution patterns.

7 CONCLUSION

PIPr [42] is an open-source dataset containing metadata of 37 712 public IaC programs on GitHub and the source code of the 15 504 IaC programs whose licenses permit redistribution. The metadata includes information about the used programming languages, applied testing techniques, and licenses. PIPr enables studies on PL-IaC and its differences from other software, which is vital to applying existing and developing new techniques for IaC programs effectively.

ACKNOWLEDGMENTS

This work has been co-funded by the Swiss National Science Foundation (SNSF, No. 200429).

REFERENCES

[1] n.d.. Licensee: A Ruby Gem to Detect Under What License a Project Is Distributed. https://licensee.github.io/licensee/. Accessed: 2023-11-30.

[2] Amazon Web Services. n.d.. Cloud Development Framework: AWS Cloud Development Kit. https://aws.amazon.com/cdk/. Accessed: 2023-11-29.

[3] Amazon Web Services. n.d.. Infrastructure as Code Provisioning Tool: AWS CloudFormation. https://aws.amazon.com/cloudformation/. Accessed: 2023-11-29.

[4] Amazon Web Services. n.d.. Testing Constructs: AWS Cloud Development Kit (AWS CDK) v2. https://docs.aws.amazon.com/cdk/v2/guide/testing.html. Accessed: 2023-11-29.

[5] Luis Cruz, Rui Abreu, and David Lo. 2019. To the Attention of Mobile Software Developers: Guess What, Test Your App! *Empir. Softw. Eng.* 24, 4 (2019), 2438–2468. https://doi.org/10.1007/S10664-019-09701-0

[6] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a Catalog of Software Quality Metrics for Infrastructure Code. *J. Syst. Softw.* 170 (2020), 110726. https://doi.org/10.1016/J.JSS.2020.110726

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

[8] Joe Duffy. 2020. Pulumi Raises Series B to Build the Future of Cloud Engineering. https://www.pulumi.com/blog/series-b/. Accessed: 2023-11-30.

[9] Joe Duffy. 2023. Building the Best Infrastructure as Code with $41M Series C Funding. https://www.pulumi.com/blog/series-c/. Accessed: 2023-11-30.

[10] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: A Language and Infrastructure for Analyzing Ultra-large-scale Software Repositories. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 422–431. https://doi.org/10.1109/ICSE.2013.6606588

[11] GitHub. n.d.. Github Docs: Searching Code (Legacy). https://docs.github.com/en/search-github/searching-on-github/searching-code. Accessed: 2023-11-30.

[12] GitHub. n.d.. Licensing a Repository. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository. Accessed: 2023-11-30.

[13] Georgios Gousios. 2013. The GHTorrent Dataset and Tool Suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim (Eds.). IEEE Computer Society, 233–236. https://doi.org/10.1109/MSR.2013.6624034

[14] HashiCorp. n.d.. CDK for Terraform. https://developer.hashicorp.com/terraform/cdktf. Accessed: 2023-11-29.

[15] HashiCorp. n.d.. HCL. https://github.com/hashicorp/hcl. Accessed: 2023-11-30.

[16] HashiCorp. n.d.. Terraform. https://www.terraform.io/. Accessed: 2023-11-29.

[17] HashiCorp. n.d.. Unit Tests: CDK for Terraform. https://developer.hashicorp.com/terraform/cdktf/test/unit-tests. Accessed: 2023-11-29.

[18] Mohammad Mehedi Hassan and Akond Rahman. 2022. As Code Testing: Characterizing Test Quality in Open Source Ansible Development. In *15th IEEE Conference on Software Testing, Verification and Validation, ICST 2022, Valencia, Spain, April 4-14, 2022.* IEEE, 208–219. https://doi.org/10.1109/ICST53961.2022.00031

[19] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. 2013. Testing Idempotence for Infrastructure as Code. In *Middleware 2013 - ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, December 9-13, 2013, Proceedings (Lecture Notes in Computer Science, Vol. 8275)*, David M. Eyers and Karsten Schwan (Eds.). Springer, 368–388. https://doi.org/10.1007/978-3-642-45065-5_19

[20] Matej Madeja, Jaroslav Porubän, Michaela Bacíková, Matús Sulír, Ján Juhár, Sergej Chodarev, and Filip Gurbál. 2021. Automating Test Case Identification in Java Open Source Projects on GitHub. *Comput. Informatics* 40, 3 (2021). https://doi.org/10.31577/CAI_2021_3_575

[21] Microsoft Azure. n.d.. Bicep. https://github.com/Azure/bicep. Accessed: 2023-11-30.

[22] Kief Morris. 2021. *Infrastructure as Code: Dynamic Systems for the Cloud Age* (second ed.). O'Reilly Media, Inc., Sebastopol, CA, USA.

[23] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for Engineered Software Projects. *Empir. Softw. Eng.* 22, 6 (2017), 3219–3253. https://doi.org/10.1007/S10664-017-9512-6

[24] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2021. Andromeda: A Dataset of Ansible Galaxy Roles and Their Evolution. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021.* IEEE, 580–584. https://doi.org/10.1109/MSR52588.2021.00078

[25] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2022. Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022.* ACM, 61–72. https://doi.org/10.1145/3524842.3527964

[26] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2023. Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort?. In *20th IEEE/ACM International Conference on Mining Software Repositories, MSR 2023, Melbourne, Australia, May 15-16, 2023.* IEEE, 534–545. https://doi.org/10.1109/MSR59073.2023.00079

[27] Ruben Opdebeeck, Ahmed Zerouali, Camilo Velázquez-Rodríguez, and Coen De Roover. 2021. On the Practice of Semantic Versioning for Ansible Galaxy Roles: An Empirical Study and a Change Classification Model. *J. Syst. Softw.* 182 (2021), 111059. https://doi.org/10.1016/j.jss.2021.111059

[28] Progress. n.d.. Chef Software DevOps Automation Solutions. https://chef.io. Accessed: 2023-11-29.

[29] Pulumi. n.d.. Integration Testing for Pulumi Programs. https://www.pulumi.com/docs/using-pulumi/testing/integration/. Accessed: 2023-11-29.

[30] Pulumi. n.d.. Policy as Code for Any Cloud with Pulumi: Pulumi CrossGuard. https://www.pulumi.com/crossguard/. Accessed: 2023-11-29.

[31] Pulumi. n.d.. Pulumi: Infrastructure as Code in Any Programming Language. https://github.com/pulumi/pulumi. Accessed: 2023-11-29.

[32] Pulumi. n.d.. Testing of Pulumi Programs. https://www.pulumi.com/docs/using-pulumi/testing/. Accessed: 2023-11-29.

[33] Puppet. n.d.. Puppet Infrastructure & IT Automation at Scale. https://puppet.com/. Accessed: 2023-11-29.

[34] Giovanni Quattrocchi and Damian Andrew Tamburri. 2022. Predictive Maintenance of Infrastructure Code Using "Fluid" Datasets: An Exploratory Study on Ansible Defect Proneness. *J. Softw. Evol. Process.* 34, 11 (2022). https://doi.org/10.1002/smr.2480

[35] Akond Rahman, Chris Parnin, and Laurie A. Williams. 2019. The Seven Sins: Security Smells in Infrastructure as Code Scripts. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 164–175. https://doi.org/10.1109/ICSE.2019.00033

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

[37] Akond Rahman and Laurie A. Williams. 2019. Source Code Properties of Defective Infrastructure as Code Scripts. *Inf. Softw. Technol.* 112 (2019), 148–163. https://doi.org/10.1016/j.infsof.2019.04.013

[38] Red Hat. n.d.. Ansible Is Simple IT Automation. https://www.ansible.com/. Accessed: 2023-11-29.

[39] Sofia Reis, Rui Abreu, Marcelo d'Amorim, and Daniel Fortunato. 2022. Leveraging Practitioners' Feedback to Improve a Security Linter. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022.* ACM, 66:1–66:12. https://doi.org/10.1145/3551349.3560419

[40] Nuno Saavedra and João F. Ferreira. 2022. GLITCH: Automated Polyglot Security Smell Detection in Infrastructure as Code. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022.* ACM, 47:1–47:12. https://doi.org/10.1145/3551349.3556945

[41] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 189–200. https://doi.org/10.1145/2901739.2901761

[42] Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2023. PIPr: A Dataset of Public Infrastructure as Code Programs. https://doi.org/10.5281/zenodo.10173400

[43] Daniel Sokolowski, Pascal Weisenburger, and Guido Salvaneschi. 2021. Automating Serverless Deployments for DevOps Organizations. In *ESEC/FSE '21: 29th*

ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 57–69. https://doi.org/10.1145/3468264.3468575

[44] Daniel Sokolowski, Pascal Weisenburger, and Guido Salvaneschi. 2021. Pulumi TypeScript Stack References to µs Converter. https://doi.org/10.5281/zenodo.4902171

[45] Daniel Sokolowski, Pascal Weisenburger, and Guido Salvaneschi. 2021. µs Performance Evaluation. https://doi.org/10.5281/zenodo.4902330

[46] Thodoris Sotiropoulos, Dimitris Mitropoulos, and Diomidis Spinellis. 2020. Practical Fault Detection in Puppet Programs. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 26–37. https://doi.org/10.1145/3377811.3380384

[47] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 2018. How Good Is Your Puppet? An Empirically Defined and Validated Quality Model for Puppet. In *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, Rocco Oliveto, Massimiliano Di Penta, and David C. Shepherd (Eds.). IEEE Computer Society, 164–174. https://doi.org/10.1109/SANER.2018.8330206