Unleashing the Giants: Enabling Advanced Testing for Infrastructure as Code

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

Infrastructure as Code (IaC) programs are written in imperative programming languages like Python or TypeScript while declaratively defining the target state of software deployments, which the IaC solution then sets up, e.g., Pulumi and AWS CDK. Through a repository mining study and analysis, we noticed that testing IaC programs poses a dilemma: current techniques are either slow and expensive or require prohibitively high development effort. To solve this issue, we introduce Automated Configuration Testing (ACT), enabling efficient testing with low development effort. ACT automates the tedious aspects of unit testing IaC programs and is extensible through a plugin system for test generators and oracles. ACT is already effective with simple type-based plugins, and leveraging existing giants, i.e., advanced test generation and oracle techniques, in new plugins will further boost its effectiveness.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; **Software functional properties**; *Orchestration languages.*

KEYWORDS

Property-based Testing, Fuzzing, Infrastructure as Code, DevOps

ACM Reference Format:
Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2024. Unleashing the Giants: Enabling Advanced Testing for Infrastructure as Code. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643078

1 INTRODUCTION

Infrastructure provisioning and application deployment are increasingly complex tasks that require automation to ensure organizations can adopt their applications quickly and frequently to changing requirements. Infrastructure as Code (IaC) [15] is the DevOps technology to automate deployments. With state-of-the-art *declarative* IaC, developers only describe the target state of their deployment, and the IaC solution achieves it by comparing the current infrastructure with the target state, deriving the required actions.

With the recent IaC solutions Pulumi [17], AWS CDK [1], and CDKTF [9], developers describe the target state of deployments in IaC programs written in languages like Python, TypeScript, or Java, as opposed to IaC scripts in JSON, YAML, or similar DSLs. Using these advanced languages provides them with powerful abstractions to tackle the increasing complexity of their deployments—besides familiarity and existing tooling. Despite only being available since 2018, the popularity of writing IaC programs is steadily increasing, a trend which we expect to continue and further accelerate.

2 THE DILEMMA OF IAC PROGRAM TESTING

The reliability of IaC programs is vital because faulty deployments can cause entire systems to malfunction and severe security vulnerabilities. As IaC programs use widely adopted programming languages with mature ecosystems, a whole array of existing quality assurance techniques is directly applicable, especially for testing. However, in a previous study on public IaC programs on GitHub, we found that only 25 % implement tests, dropping to 1 % for Pulumi [21]. We focus on Pulumi because it is the most expressive solution for IaC programs. It is the only one allowing computing on output configuration of just deployed resources in the IaC program—a powerful feature that, unfortunately, impedes testing.

To find out why developers do not write tests, we analyzed IaC program testing techniques and identified a dilemma: integration testing is very slow and resource-intensive for IaC programs. The only reliable alternative is unit testing, which is quick. However, due to the declarative nature of IaC programs, developing effective unit tests requires tremendous effort. Developers must (1) mock all resource definitions, often most of the IaC program code. The mocks must (2) validate the *input configuration* for each defined resource, making the mocks test oracles. Further, they have to (3) provide post-deployment *output configuration* for each defined resource, which is accessible in the remaining IaC program execution and, thus, test input, making the mocks also test generators. Implementing both good oracles and generators is tedious and replicates the logic of the IaC program and the infrastructure, causing a lot of tightly coupled testing code and slowing down future changes.

3 AUTOMATED CONFIGURATION TESTING

To solve the dilemma of IaC program testing, we envisioned Automated Configuration Testing (ACT) [19] inspired by property-based testing [3] and fuzzing [23] techniques. ACT automates the tedious work of implementing unit tests by automatically mocking all resource definitions. ACT leverages a plugin interface for the involved aspects of the mocks, i.e., test generation and oracles, allowing the reuse and exchange of various techniques (Figure 1). At this high degree of automation, ACT efficiently tests the IaC program under test

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3643078

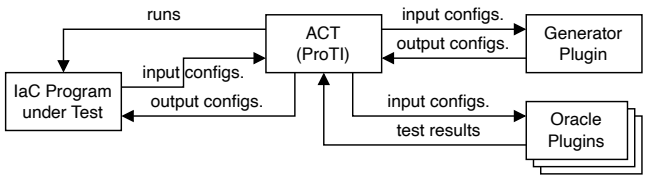

Figure 1: ACT components and their interaction.

by running it automatically in many configurations quickly. ACT sends every resource input configuration to the generator plugin, providing the resources' output configuration to use in the rest of the execution, and to all oracle plugins, verifying them. For generalized properties, developers can reuse plugins centrally maintained by the community, enabling IaC program testing without writing additional testing code. To additionally test for application-specific properties, ACT implementations can provide efficient ways to specialize oracles and generators, e.g., inline specification syntax.

As the first implementation of ACT targeting Pulumi TypeScript, we present ProTI[1] [20] based on the testing framework Jest [14] and the property-based testing library fast-check [5]. As the first generators and oracles, we provide type-based ACT plugins leveraging resource types from Pulumi package schemas. Even though these plugins are simplistic and not very precise, our evaluation with thousands of Pulumi TypeScript programs from GitHub and artificial benchmarks shows that ACT is effective. A single ProTI test run typically takes hundreds of milliseconds, allowing testing IaC programs in hundreds to thousands of configurations quickly, effectively finding bugs—even in corner cases.

4 STANDING ON THE SHOULDERS OF GIANTS FOR ADVANCED IAC PROGRAM TESTING

ACT's effectiveness depends on the generators and oracles. While our simple type-based plugins are already useful, advanced techniques promise a significant boost for the whole approach. At this point, existing and novel advanced automated testing techniques can be integrated into ProTI (ACT), effectively preventing malfunctioning and insecure deployments. In initial experiments, we already demonstrated that such integration is suitable and simple, where we implemented slim wrapper plugins integrating the Daikon invariant detector [6] and the Radamsa fuzzer [10]. We now outline ideas and approaches for future explorations.

So far, the test generation we have used is purely random and uninformed of the program and previous test runs. For better test input generation strategies, the automated testing literature proposed various approaches, including techniques leveraging test coverage and feedback information [8, 12, 16] as well as search- and grammar-based techniques [13, 22].

The problem of finding good test oracles is not limited to IaC program testing, either. Our current type-based oracles are imprecise and cannot cover validation across properties or take other contexts into account. However, there are oracle strategies that should be explored in this domain. Promising directions are finding IaC properties for differential [7], metamorphic [2], intramorphic [18], and learning-based testing approaches [4, 11].

ACKNOWLEDGMENTS

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

REFERENCES

[1] Amazon Web Services. [n. d.]. Cloud Development Framework: AWS Cloud Development Kit. https://aws.amazon.com/cdk/. Accessed: 2024-01-15.
[2] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Comput. Surv.* 51, 1 (2018), 4:1–4:27. https://doi.org/10.1145/3143561
[3] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proc. of the 5th ACM SIGPLAN ICFP '00, Montreal, Canada, 2000*. ACM, 268–279. https://doi.org/10.1145/351240.351266
[4] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K. Lahiri. 2022. TOGA: A Neural Method for Test Oracle Generation. In *Proc. of the 44th IEEE/ACM ICSE 2022, Pittsburgh, PA, USA, 2022*. ACM, 2130–2141. https://doi.org/10.1145/3510003.3510141
[5] Nicolas Dubien. [n. d.]. fast-check: Official Documentation. https://fast-check.dev/. Accessed: 2024-01-15.
[6] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon System for Dynamic Detection of Likely Invariants. *Sci. Comput. Program.* 69, 1-3 (2007), 35–45. https://doi.org/10.1016/j.scico.2007.01.015
[7] Robert B. Evans and Alberto Savoia. 2007. Differential Testing: A New Approach to Change Detection. In *Proc. of the 6th ACM SIGSOFT ESEC/FSE, 2007, Dubrovnik, Croatia, 2007*. ACM, 549–552. https://doi.org/10.1145/1287624.1287707
[8] Harrison Goldstein, John Hughes, Leonidas Lampropoulos, and Benjamin C. Pierce. 2021. Do Judge a Test by Its Cover - Combining Combinatorial and Property-based Testing. In *Proc. of the 30th ESOP, part of ETAPS,Luxembourg, 2021*, Vol. 12648. Springer, 264–291. https://doi.org/10.1007/978-3-030-72019-3_10
[9] HashiCorp. [n. d.]. CDK for Terraform. https://developer.hashicorp.com/terraform/cdktf. Accessed: 2024-01-15.
[10] Aki Helin. [n. d.]. Radamsa: A General-purpose Fuzzer. https://gitlab.com/akihe/radamsa. Accessed: 2024-01-15.
[11] Ali Reza Ibrahimzada, Yigit Varli, Dilara Tekinoglu, and Reyhaneh Jabbarvand. 2022. Perfect Is the Enemy of Test Oracle. In *Proc. of the 30th ACM ESEC/FSE, Singapore, 2022*. ACM, 70–81. https://doi.org/10.1145/3540250.3549086
[12] Dongge Liu, Gidon Ernst, Toby Murray, and Benjamin I. P. Rubinstein. 2020. LEGION: Best-first Concolic Testing. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 54–65. https://doi.org/10.1145/3324884.3416629
[13] Andreas Löscher and Konstantinos Sagonas. 2017. Targeted Property-based Testing. In *Proc. of the 26th ACM SIGSOFT (ISSTA), Santa Barbara, CA, USA, 2017*. ACM, 46–56. https://doi.org/10.1145/3092703.3092711
[14] Meta Platforms. [n. d.]. Jest: Delightful JavaScript Testing. https://jestjs.io/. Accessed: 2023-11-29.
[15] Kief Morris. 2021. *Infrastructure as Code: Dynamic Systems for the Cloud Age* (second ed.). O'Reilly Media, Inc., Sebastopol, CA, USA.
[16] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. 2007. Feedback-directed Random Test Generation. In *Proc. of the 29th ICSE, Minneapolis, MN, USA, 2007*. IEEE, 75–84. https://doi.org/10.1109/ICSE.2007.37
[17] Pulumi. [n. d.]. Pulumi: Infrastructure as Code in Any Programming Language. https://github.com/pulumi/pulumi. Accessed: 2024-01-15.
[18] Manuel Rigger and Zhendong Su. 2022. Intramorphic Testing: A New Approach to the Test Oracle Problem. In *Proc. of the 2022 ACM SIGPLAN Onward!, Auckland, New Zealand, 2022*, Christophe Scholliers and Jeremy Singer (Eds.). ACM, 128–136. https://doi.org/10.1145/3563835.3567662
[19] Daniel Sokolowski and Guido Salvaneschi. 2023. Towards Reliable Infrastructure as Code. In *Companion Proc. of the 20th ICSA, L'Aquila, Italy, 2023*. IEEE, 318–321. https://doi.org/10.1109/ICSA-C57050.2023.00072
[20] Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2023. ProTI: Automated Unit Testing of Pulumi TypeScript Infrastructure as Code Programs. https://doi.org/10.5281/zenodo.10028479
[21] Daniel Sokolowski, David Spielmann, and Guido Salvaneschi. 2024. The PIPr Dataset of Public Infrastructure as Code Programs. In *Accepted at IEEE/ACM MSR*. Accepted.
[22] Dominic Steinhöfel and Andreas Zeller. 2022. Input Invariants. In *Proc. of the 30th ACM ESEC/FSE, Singapore, 2022*. ACM, 583–594. https://doi.org/10.1145/3540250.3549139
[23] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2023. Fuzzing: Breaking Things with Random Inputs. https://www.fuzzingbook.org/html/Fuzzer.html. Accessed: 2023-11-30.

[1] https://proti-iac.github.io