Creed for Speed: Comprehensive Infrastructure as Code Testing

Accepted Talk Abstract at CONFLANG 2023, October 24, 2023, Cascais, Portugal

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

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

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

ABSTRACT

With Programming Languages Infrastructure as Code (PL-IaC), developers implement imperative IaC programs in one of many general-purpose programming languages, e.g., TypeScript, Python, or Go, to declaratively describe deployments. Using these languages provides access to quality assurance techniques and tools developed for traditional software; however, programmers routinely rely on prohibitively slow integration testing—if they test at all. As a result, even simple bugs are found late, tremendously slowing down the development process.

To improve the velocity of PL-IaC development, we propose ProTI, an automated unit testing approach that quickly tests PL-IaC programs in many different configurations. ProTI mocks all cloud resources, replacing them with pluggable oracles that validate all resources' configurations and a generator for realistic test inputs. We implemented ProTI for Pulumi TypeScript with simple generator and oracle plugins. Our experience of testing with ProTI encourages the exploration of more sophisticated oracles and generators, leading to the early detection of more bugs. ProTI enables programmers to rapidly prototype, explore, and plug in new oracles and generators for efficient PL-IaC program testing.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Software functional properties**; *Orchestration languages*; • **Computer systems organization** → Cloud computing.

KEYWORDS

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

PL-IAC PROGRAM TESTING

With PL-IaC solutions, e.g., Pulumi [7] and AWS CDK [1], developers implement programs in a general-purpose programming language and define resources by instantiating objects of the resources' classes. For instance, Lines 1 to 5 in Figure 1 define an AWS S3 bucket object. All features of the programming language can be used, and PL-IaC programs can have plenty of issues. Yet, the reliability of PL-IaC programs is imperative. Faulty PL-IaC programs can prevent the deployment, cause a non-functional setup, or yield a functional setup with security issues. However, our survey of all PL-IaC projects public on GitHub shows that less than 1 % of the Pulumi projects use unit testing. For AWS CDK, it is 38 %, but AWS CDK only implements a limited form of PL-IaC, simplifying testing.

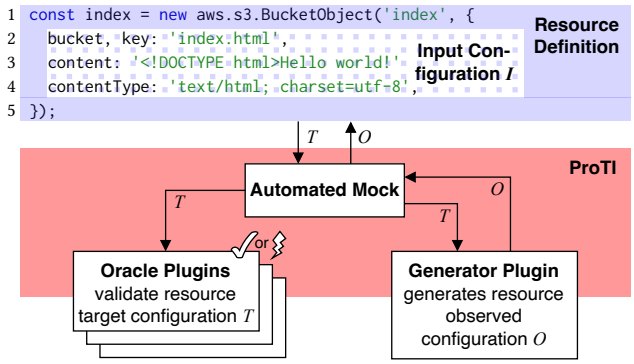

Figure 1: Resource definition in a Pulumi TypeScript PL-IaC program. ProTI mocks the resource, validates the resource's target configuration T using pluggable oracles, and returns suitable test inputs, i.e., a simulated observed resource configuration O that is provided by the generator plugin.

The absence of unit tests limits to integration testing, where a single test takes at least tens of seconds and often minutes or longer. Even simple bugs are spotted at high latency, hampering the developers' velocity. Still, developers seem to perceive unit testing as impracticably effortful. Related work confirms that, generally, testing IaC is high-effort and needs better techniques [8, 6].

AUTOMATING UNIT TESTING WITH PROTI

We propose ProTI for PL-IaC program unit testing [9], an approach bringing low-effort property-based testing [5, 2] and fuzzing [10] to PL-IaC programs. ProTI provides fast test execution with high-quality assessment of the PL-IaC program under test.

ProTI automatically mocks all resource definitions, eliminating the slow integration with cloud providers. Figure 1 shows ProTI's interaction with the PL-IaC program under test. The automated mock receives the resources' target configuration T, derived from the developer-specified input configuration I. However, a naïve mock would not provide insight. It would permit any resource configuration, including invalid and faulty ones. Thus, ProTI uses oracle plugins that implement an efficient model of the cloud providers to validate each T. Further, a naïve mock does not generate an observed post-deployment configuration O, which otherwise would be returned by the cloud provider. O is test input because it is accessible on the resource object and can be used in the remainder of the PL-IaC program. ProTI uses a generator plugin that efficiently transforms each T into an O. At this level of automation, ProTI can

quickly run the PL-IaC program hundreds of times, steered by a generator that provides valid configurations that are ideally in an order that tests bug-triggering configurations as early as possible.

IMPLEMENTATION

We implemented ProTI for Pulumi TypeScript as an extension of Jest [4] using fast-check [3]. Initial oracle and generator plugins are based on Pulumi package schemas, checking the type-level validity of resource configurations and generating type-compliant observed configurations. Prototyping and using new oracle and generator plugins is simple and facilitates the exploration of new strategies.

REFERENCES

[1] Amazon Web Services. 2023. AWS cloud development kit. https://aws.amazon.com/cdk/ (Accessed: 2022-07-12). (2023).

[2] Koen Claessen and John Hughes. 2000. QuickCheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming* (ICFP '00). Association for Computing Machinery, New York, NY, USA, 268–279. ISBN: 1581132026. DOI: 10.1145/351240.351266.

[3] Nicolas Dubien. 2022. Fast-check: property based testing framework for JavaScript/TypeScript. https://github.com/dubzzz/fast-check (Accessed: 2022-07-12). (2022).

[4] Facebook. 2023. Jest: delightful javascript testing. https://jestjs.io/ (Accessed: 2023-01-29). (2023).

[5] George Fink and Matt Bishop. 1997. Property-based testing: a new approach to testing for assurance. *ACM SIGSOFT Softw. Eng. Notes*, 22, 4, 74–80. DOI: 10.1145/263244.263267.

[6] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. 2019. Adoption, support, and challenges of infrastructure-as-code: insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 580–589. DOI: 10.1109/ICSME.2019.00092.

[7] Pulumi. 2022. Pulumi: universal infrastructure as code. https://github.com/pulumi/pulumi (Accessed: 2022-07-12). (2022).

[8] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2019. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108, 65–77. DOI: https://doi.org/10.1016/j.infsof.2018.12.004.

[9] Daniel Sokolowski and Guido Salvaneschi. 2023. Towards reliable infrastructure as code. In *20th International Conference on Software Architecture, ICSA 2023 - Companion, L'Aquila, Italy, March 13-17, 2023*. IEEE, 318–321. DOI: 10.1109/ICSA-C57050.2023.00072.

[10] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2021. *The Fuzzing Book*. Retrieved 2021-10-26 21:30:20+08:00. CISPA Helmholtz Center for Information Security. https://www.fuzzingbook.org/.