Anti-Patterns in Infrastructure as Code

Akond Rahman
Department of Computer Science
North Carolina State University
Raleigh, North Carolina
Email: aarahman@ncsu.edu

***Abstract*—In DevOps, infrastructure as code (IaC) scripts are used by practitioners to create and manage an automated deployment pipeline that enables IT organizations to release their software changes rapidly at scale. Low quality IaC scripts can have serious consequences, potentially leading to wide-spread system outages and service discrepancies. *The goal of this research is to help practitioners increase the quality of infrastructure as code (IaC) scripts by identifying anti-patterns in IaC scripts and development of IaC scripts.* Using open source repositories, we conduct three initial studies to (i) quantify the frequency and categorize the defects in IaC scripts; and (ii) identify operations that characterize defective IaC scripts. Based on our empirical analysis we observe (i) the dominant defect defect categories to be related to syntax and configuration assignments, and (ii) three operations that characterize defective IaC scripts. The above-mentioned findings motivate us to identify anti-patterns in IaC scripts and IaC development. To this end, we propose three studies that identify (i) process anti-patterns; and (ii) security-related anti-patterns in IaC.**

I. Introduction

Information technology (IT) organizations are increasingly adopting DevOps practices [1]. DevOps organizations i.e. IT organizations that adopt DevOps, have strong collaboration between software development and operations teams to deliver software rapidly [2]. One technology that these organizations consider essential to implement DevOps is the use of infrastructure as code (IaC) scripts [2] [3]. DevOps organizations use IaC scripts such as Puppet [1] scripts, to automatically manage their configurations and operations infrastructure [2].

Similar to software source code, IaC scripts can experience frequent churn, making these scripts susceptible to quality issues such as defects [4]. Defects in IaC scripts can have serious consequences, as these scripts are associated in setting up and managing cloud-based infrastructure, and ensuring availability of software services. For example on January 2017, execution of a defective IaC script erased home directories of around 270 users in cloud instances maintained by Wikimedia [2]. The above-mentioned evidence demonstrated in real-world, and research studies motivate us to systematically study quality issues of IaC scripts in forms of anti-patterns.

Anti-patterns in software engineering correspond to practices that may have negative consequences [5]. Practitioners might be inadvertently implementing practices with negative consequences due to lack of knowledge, lack of experience, or applying a perceived good practice in the wrong context [5]. By identifying anti-patterns we can provide actionable recommendations to practitioners, and pinpoint characteristics that correlate with defects and violates security and privacy objectives.

Thesis Statement: Through systematic investigation, we can identify anti-patterns in infrastructure as code that (i) correlate with defects; and (ii) violate security and privacy objectives.

We empirically evaluate our thesis statement by answering the following research questions:

- ***RQ-1: How frequently do defects occur in infrastructure as code (IaC) scripts? What categories of defects occur IaC scripts?***
- ***RQ-2: What text features characterize defective infrastructure as code scripts?***
- ***RQ-3: What are the process anti-patterns in developing infrastructure as code scripts?***
- ***RQ-4: What security anti-patterns are exhibited in infrastructure as code scripts?***

Upon completion of this thesis we expect to make the following contributions:

- A set of process, and security-related anti-patterns;
- Tool suites that extract process, and security anti-patterns from IaC scripts;
- Datasets where scripts are labeled as defective, and violating security-related anti-patterns are identified; and
- Defect prediction models built using process anti-patterns of IaC scripts

II. Research

A. Study-1: Defect Categories (Under Review at TSE'18)

Motivation: Categorization of defects for a software system helps in formulating effective mitigation strategies, and prioritize testing efforts [6]. Researchers [7] have previously used classification schemes, such as the defect type attribute of orthogonal defect classification (ODC) [6], to classify defects for non-IaC software systems written in GPLs. By characterizing defects in IaC we can understand how frequently defects occur, and what categories of defects occur in IaC scripts.

Methodology: We use the defect type attribute of ODC to categorize defects. We select the ODC defect type attribute as

[1] https://puppet.com/

[2] https://wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs

this technique uses semantic information to provide informed decisions on the defect categories [6]. According to the ODC defect type attribute, a defect can belong to eight categories. As an XCM might not correspond to a defect, we added a 'no defect' category. Furthermore, a XCM might not to belong to any of the eight categories that belong to the ODC defect type attribute. Hence, we introduced the 'other' category. Altogether we considered 10 categories, and classified the XCMs into one of these 10 categories. We constructed three datasets: 'Mozilla', 'Openstack', and 'Wikimedia'.

Results: Respectively, for Mozilla, Openstack, and Wikimedia, we observe (i) 42.8%, 66.8%, and 50.3% of the defective IaC scripts to contain defects that belong to category assignment; and (ii) assignment-related defects are more prevalent amongst IaC systems compared to previously studied non-IaC systems.

B. Study-2: Characteristics of Defective Scripts (Accepted at ICST'18)

Motivation: In prior work, researchers have used text features to characterize defective software source files written in GPLs, such as Java [8]. IaC scripts use domain specific languages (DSLs) [9]. The syntax and semantics of DSLs are fundamentally different from GPLs [10], and through systematic investigation we can determine if text-based features can be used effectively for characterizing and predicting defective IaC scripts.

Methodology: We characterize defective IaC scripts by extracting text features. We use two text mining techniques to extract text features: the 'bag-of-words (BOW)' technique [11] and the 'term frequency-inverse document frequency (TF-IDF)' technique [12]. We apply the Strauss-Corbin Grounded Theory (SGT) [**?**] on text features that correlate with defective IaC scripts to characterize properties of defective IaC scripts. We construct defect prediction models using the text features and Random Forest (RF) [13].

Results: We identify three properties that characterize defective IaC scripts: filesystem operations, infrastructure provisioning, and managing user accounts. Using the bag-of-word technique, we observe a median F-Measure of 0.74, 0.71, and 0.73, respectively, for Mozilla, Openstack, and Wikimedia Commons. Using the TF-IDF technique, we observe a median F-Measure of 0.72, 0.74, and 0.70, respectively, for Mozilla, Openstack, and Wikimedia Commons.

C. Study-3: Process Characteristics (In Progress)

Motivation: Prior research has shown that software source code written in GPLs is correlated with process metrics. We hypothesize that a certain set of characteristics related to the IaC development process are correlated with defects, and can be used to predict defective scripts.

Methodology: We hypothesize the following characteristics to be correlated with defective IaC scripts: commits, age, number of developers who modified the script, lines changed per commit, and number of developers who multitask. We also used these characteristics to build prediction models.

D. Study-4: Characteristics that Violate Security Objectives (Proposed)

Motivation: As IaC scripts hold crucial information about the deployment environment, violation of security objectives can be disastrous. We refer to characteristics of IaC scripts that violate security objectives as security-related anti-patterns. As an example anti-pattern, if administrator credentials are hard-coded in IaC scripts, attackers can use those credentials and hack into the deployment infrastructure.

Methodology: As the first step to extract the security-related anti-patterns, we will apply grounded theory analysis [14]. Next, we will create an automated tool that will identify the security-related anti-patterns in IaC scripts. We plan to add custom heuristics derived from our qualitative analysis, and extend existing commercial tools such as, puppet-lint [3].

REFERENCES

[1] N. F. A. Brown, J. Humble, N. Kersten, and G. Kim, "2016 State of DevOps Report," https://puppet.com/resources/whitepaper/2016-state-of-devops-report, 2017, [Online; accessed 15-August-2017].

[2] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[3] D. Spinellis, "Don't install software by hand," *IEEE Software*, vol. 29, no. 4, pp. 86–87, July 2012.

[4] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code: An empirical study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 45–55. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820527

[5] W. H. Brown, R. C. Malveau, H. W. S. McCormick, and T. J. Mowbray, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[6] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M. Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, Nov 1992.

[7] A. Pecchia and S. Russo, "Detection of software failures through event logs: An experimental study," in *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, Nov 2012, pp. 31–40.

[8] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, Oct 2014.

[9] R. Shambaugh, A. Weiss, and A. Guha, "Rehearsal: A configuration verification tool for puppet," *SIGPLAN Not.*, vol. 51, no. 6, pp. 416–430, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980983.2908083

[10] E. Van Wyk, L. Krishnan, D. Bodin, and A. Schwerdfeger, "Attribute grammar-based language extensions for java," in *Proceedings of the 21st European Conference on Object-Oriented Programming*, ser. ECOOP'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 575–599. [Online]. Available: http://dl.acm.org/citation.cfm?id=2394758.2394796

[11] Z. S. Harris, "Distributional structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[12] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[13] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[14] K. Charmaz, *Constructing grounded theory*. London, UK: Sage Publishing, 2014.

[3] http://puppet-lint.com/