Characteristics of Defective Infrastructure as Code Scripts in DevOps

Akond Rahman
North Carolina State University, North Carolina, USA
Adviser: Laurie Williams
aarahman@ncsu.edu

ABSTRACT

Defects in infrastructure as code (IaC) scripts can have serious consequences for organizations who adopt DevOps. By identifying which characteristics of IaC scripts correlate with defects, we can identify anti-patterns, and help software practitioners make informed decisions on better development and maintenance of IaC scripts, and increase quality of IaC scripts. *The goal of this paper is to help practitioners increase the quality of IaC scripts by identifying characteristics of IaC scripts and IaC development process that correlate with defects, and violate security and privacy objectives.* We focus on characteristics of IaC scripts and IaC development that (i) correlate with IaC defects, and (ii) violate security and privacy-related objectives namely, confidentiality, availability, and integrity. For our initial studies, we mined open source version control systems from three organizations: Mozilla, Openstack, and Wikimedia, to identify the defect-related characteristics and conduct our case studies. From our empirical analysis, we identify (i) 14 IaC code and four churn characteristics that correlate with defects; and (ii) 12 process characteristics such as, frequency of changes, and ownership of IaC scripts that correlate with defects. We propose the following studies: (i) identify structural characteristics that correlate with defects; (ii) with respect to prediction performance, compare which characteristics of IaC scripts are more correlated with defects; and (iii) identify characteristics that violate security and privacy objectives.

CCS CONCEPTS

• **Software and its engineering** → **Software reliability**; **Software defect analysis**;

KEYWORDS

defects; devops; infrastructure as code; metrics

ACM Reference Format:
Akond Rahman. 2018. Characteristics of Defective Infrastructure as Code Scripts in DevOps. In *Proceedings of International Conference on Software Engineering (ICSE'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

IT organizations that have adopted DevOps have strong collaboration between software development and operations teams to deliver software rapidly [8]. Automation of development and deployment steps is key to DevOps adoption, and DevOps organizations use technologies to automate repetitive work [8]. One technology that these organizations consider essential to implement DevOps is the use of infrastructure as code (IaC) scripts [8]. IaC scripts help to provision and manage cloud-based infrastructure [8], such as Amazon Web Services [1]. Treating IaC scripts as software source code and maintaining them in version control systems (VCSs) is one of the prerequisites to implementing the practice of IaC [13]. According to practitioners, maintenance of IaC scripts in VCS systems provides benefits, for example, increase in transparency and visibility amongst the development and operation teams, as scripts are shared and accessible [8]. However, similar to software source code, IaC scripts change frequently [9], and frequent changes in IaC scripts can introduce defects [13]. Defects in IaC scripts can have dire consequences, for example on January 2017, execution of a defective IaC script erased home directories of 270 users in cloud instances maintained by Wikimedia [2].

Characterization of defects can be helpful to identify actionable anti-patterns that correlate with defective IaC scripts. As a hypothetical example, if the practice of making large changes is a defect-related practice, then we can identify the practice of making large changes as an anti-pattern, and recommend practitioners to make small changes in IaC scripts. To characterize the defects that occur in software source code written in general purpose programming languages (GPLs) such C++ and Java, researchers [1] [23] [5] [21] have proposed and evaluated a plethora of metrics and techniques. Unlike GPLs, IaC scripts use domain-specific languages (DSLs) [17]. DSLs are fundamentally different from GPLs with respect to syntax and semantics [7], and therefore, metrics and techniques proposed by prior research may not be applicable for IaC scripts. For characterization of defects in IaC scripts, we observe the necessity of systematic investigation. We hypothesize:

Through systematic investigation and validation, we can identify characteristics of defective infrastructure as code scripts, and identify anti-patterns that (i) correlate with defects; and (ii) violate security and privacy objectives.

We evaluate our hypothesis by answering the following six research questions:

[1]https://aws.amazon.com/
[2]https://wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs

- **RQ-1**: ***What categories of defects occur in infrastructure as code scripts?*** **(Study-1)**
- **RQ-2**: ***What code-related characteristics correlate with defects in infrastructure as code scripts?*** **(Study-2)**
- **RQ-3**: ***What process characteristics correlate with defects in infrastructure as code scripts?*** **(Study-3)**
- **RQ-4**: ***What structural characteristics correlate with defects infrastructure as code scripts?*** **(Study-4)**
- **RQ-5**: ***Which characteristics are most effective in predicting defective infrastructure as code scripts?*** **(Study-5)**
- **RQ-6**: ***Which characteristics violate security and privacy objectives in infrastructure as code scripts?*** **(Study-6)**

We focus on characteristics of IaC scripts and IaC development that (i) correlate with IaC defects; and (ii) violate security and privacy (S&P)-related objectives namely, confidentiality, integrity, and availability.

Expected contributions:

- A set of churn, code, process, and structural characteristics that correlate with defects;
- A set of characteristics that violate S&P-related objectives;
- Defect prediction models built using the identified code, process, and structural characteristics;
- Tool suites that extract code, process, and violated S&P-related characteristics from IaC scripts; and
- Datasets where scripts are labeled as defective and violated S&P characteristics are identified.

2 BACKGROUND AND RELATED WORK

2.1 Background

IaC is the technology of automatically defining and managing computing and network configurations, and infrastructure through source code [8]. IaC scripts use DSLs [17]. DevOps organizations widely use commercial tools, such as Ansible [3], and Puppet [4] to implement IaC [8] [9] [17]. IaC scripts are also known as configuration as code scripts [18] [8].

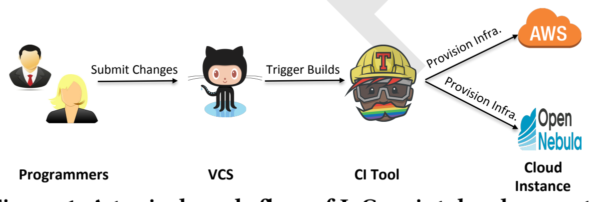

Figure 1: A typical work-flow of IaC script development.

We use Figure 1 to describe a typical work-flow of the IaC development process. Programmers make changes to the required IaC scripts and submit them to a VCS, such as Git. Once changes are submitted, a build in the continuous integration (CI) tool (e.g. Travis CI) is triggered. The CI tool runs the lint checks and test cases. If all the lint checks and tests pass, the CI tool integrates all the changes and deploys the changes to cloud instances.

[3]https://www.ansible.com/
[4]https://puppet.com/

2.2 Related Work

Our publication is closely related to prior academic studies that have (i) investigated the technology of IaC, and (ii) identified characteristics that correlate with defects that occur in software source code.

IaC Technology: Sharma et al. [18] investigated smells in IaC scripts and proposed 13 implementation and 11 design configuration smells. Hanappi et al. [6] investigated how convergence of Puppet scripts can be automatically tested, and proposed an automated model-based test framework. Jiang and Adams [9], and Parnin et al. [13] in separate studies reported that IT organizations change their IaC scripts frequently. Rahman et al. [15] identified which factors influence practitioners' usage of IaC tools. The above-mentioned studies highlight the lack of defect-related studies for IaC scripts.

Characteristics that Correlate With Defects: Prior studies have proposed and evaluated a set of characteristics in forms of metrics that correlate with software source code defects. We briefly describe these studies as following:

- *Code Characteristics*: Zimmermann et al. [23] proposed a set of 14 code characteristics in forms of metrics for predicting defects in Eclipse. Nagappan and Ball [10] investigated seven absolute and eight relative code churn characteristics, and reported relative churn characteristics that are better predictors of defect density. Nunuez-Varela et al. [12] performed a systematic literature review of 226 research papers, and reported that researchers have extensively studied code metrics as characteristics, and used them for defect and fault prediction.
- *Process Characteristics*: Bird et al. [1] reported that proportion of ownership is correlated with software failures. Nagappan et al. [11] investigated the structure of organization, and reported that structure of organization is correlated with failure-prone binaries. Rahman and Devanbu [16] compared process and code characteristics for defect prediction, and observed that process metrics are significantly better than that of code metrics.
- *Characteristics that Violate Security and Privacy Objectives*: Viega et al. [21] listed a set of S&P-related characteristics for structure programming languages such as C, and object-oriented programming (OOP) languages such as Java. The Common Weakness Enumeration (CWE) community [4] also lists a set of characteristics for SQL scripts and software source code that can violate security and privacy objectives.

The above-mentioned studies highlight the vast amount of research in the area of code, process and S&P characteristics for software source code, but not for IaC scripts. We propose to investigate the code, process, and S&P characteristics for IaC scripts, by mining open source VCS systems.

3 RESEARCH

3.1 Definitions

- **Defect-related commit**: A commit whose message indicates that an action was taken related to a defect.
- **Defective script**: An IaC script which is listed in a defect-related commit.

3.2 Dataset Construction & Modeling Techniques

Dataset Construction: Researchers [20] [5], in prior work on defect prediction, used datasets from public software data archives, such as Tera-PROMISE and NASA. But these datasets are derived from OOP-based systems [5], and not from IaC scripts, which motivates us to construct IaC-specific datasets.

Prior research [22] leveraged open source repositories that use VCS for defect prediction studies. We use two artifacts from VCS of the selected repositories to construct our datasets: (i) commits that indicate modification of IaC scripts; and (ii) issue reports that are linked with the commits. We combine the commit message with any existing issue summary to construct the message for analysis. We refer to the combined message as 'extended commit message (XCM)' throughout the rest of the paper. We use the extracted XCMs to separate the defect-related commits.

We apply qualitative analysis to determine which commits were defect-related commits. Using qualitative analysis we can classify which commits and XCMs are defect-related. From the defect-related commits we determine which IaC scripts are defective, similar to prior work [22]. Defect-related commits list which IaC scripts were changed, and from this list we determine which IaC scripts are defective. We constructed our datasets from three organizations: Mozilla, Openstack, and Wikimedia, which respectively included 580, 1383, and 296 IaC scripts. We used these three datasets for our initial studies: Study-1, 2, and 3.

Modeling Techniques: For Study-2, 3, 4, and 5, we use the identified code, process and structure characteristics to build models to predict defective IaC scripts. We identify characteristics as anti-patterns which significantly correlate with defective scripts. The identified characteristics can correlate with each other, and we account for this collinearity using principal component analysis (PCA) [19]. Next, we use build prediction models using a statistical learner, random forest [19]. We evaluate the accuracy of the prediction models using area under the receiver operator characteristic curve (AUC).

3.3 Study-1: Defect Categories (Under Review at EMSE)

Motivation: Categorization of defects for a software system helps in formulating effective mitigation strategies, and prioritize testing efforts [3]. Researchers [14] have previously used classification schemes, such as the defect type attribute of orthogonal defect classification (ODC) [3], 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 this technique uses semantic information collected from the software system, and make informed decisions on the defect categories [3]. According to the ODC defect type attribute, a defect can belong to one of the eight categories: 'algorithm', 'assignment', 'build/package/merge', 'checking', 'documentation', 'function', 'interface', and 'timing/serialization'. 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.

Results: Respectively, for Mozilla, Openstack, and Wikimedia, we observe (i) 44.6%, 58.5%, and 54.4% of the studied IaC scripts contain at least one defect; (ii) 42.8%, 66.8%, and 50.3% of the defective IaC scripts to contain defects that belong to category assignment; and (iii) assignment-related defects are more prevalent amongst IaC systems compared to previously studied non-IaC systems.

3.4 Study-2: Code Characteristics (Under Review at ICSE)

Motivation: Prior studies used a wide range of code characteristics for defect prediction, such as depth of inheritance and coupling between objects [12]. However, these characteristics have been shown to be applicable for GPLs [12]. Therefore, to build defect prediction models for IaC scripts, we need to systematically identify a set of characteristics in forms of metrics applicable for IaC scripts.

Methodology: We investigate what metrics can be used for characterizing defective IaC scripts. To characterize defective IaC scripts we apply Constructivist Grounded Theory [2] on defect-related commits mined from VCSs, and extract the characteristics.

Results: We identify 18 characteristics: 14 of the identified characteristics are IaC-related, and four characteristics are churn-related. Examples of these characteristics are: count of value assignments, string density, and string count within a script. Our findings suggest an anti-pattern which is the practice of using IaC scripts to assign configuration values. For 10×10-Fold cross validation, the median accuracy is 0.74, 0.75, and 0.71 respectively for Mozilla, Openstack, and Wikimedia.

3.5 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.

Preliminary Results: We identify two process anti-patterns: (i) the practice of making large and scattered changes at a time, and script modification by minor contributors. We observe a median accuracy measure of 0.72, 0.77, and 0.68, respectively for Mozilla, Openstack, and Wikimedia, for 10×10-fold cross validation.

3.6 Study-4: Structural Characteristics (Proposed)

Motivation: Abstract syntax trees (ASTs) of IaC scripts contains information on how the IaC script is structured. We hypothesize that certain characteristics of theses ASTs can provide us clues on which structural characteristics can correlate with defects, and can be used for predicting defective IaC scripts.

Methodology: We propose a set of structural characteristics mined from ASTs of IaC scripts that are correlated with defects. These structural characteristics include: number of parent and leaf nodes, and number of changed nodes. We also plan to use these characteristics to build prediction models.

3.7 Study-5: Comparison of Characteristics (Proposed)

Motivation: In the previous subsections we have discussed which group of code, process, and structural characteristics metrics can be related to defects, and used for predicting defective IaC scripts. However, we have not discussed prediction performance-wise, which group of characteristics are better. Through systematic investigation we can identify which group of characteristics correlate more with defective IaC scripts, and can yield better defect prediction performance.

Methodology: We will perform our comparison using four performance measures: precision, recall, F-measure, and AUC. We will apply the Scott Knott Test to compare if the three types of characteristics namely, code, process, and structural characteristics perform significantly better than each other for the four performance measures. We will perform comparison using two evaluation schemes: (i) 10×10-fold cross validation, and (ii) cross dataset evaluation.

3.8 Study-6: Characteristics that Violate Security and Privacy Objectives (Proposed)

Motivation: As IaC scripts hold crucial information about the deployment environment, violation of security and privacy (S&P) objectives can be disastrous. We refer to characteristics of IaC scripts that violate S&P objectives as S&P-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. Systematic investigation can help in identifying which S&P-related anti-patterns occur in IaC scripts and at which frequency.

Methodology: As the first step to extract the S&P-related anti-patterns, we will apply grounded theory analysis [2]. Next, we will create an automated tool that will identify the S&P-related anti-patterns in IaC scripts using syntax-driven techniques. We plan to add custom heuristics derived from our CGT analysis, and extend existing commercial tools such as, puppet-lint [5], and ansible-lint [6].

4 TIMELINE

The author of the publication is a fourth year PhD student, who is expected to perform his Doctoral proposal in January, 2018. He has completed Study-1, and 2, and they are currently under review respectively, at the Empirical Software Engineering journal in 2017, and International Conference for Software Engineering (ICSE) 2018. He aims to submit Study-3, 4, 5, and 6, in prestigious conferences and journals related to software engineering.

[5]http://puppet-lint.com/
[6]https://github.com/willthames/ansible-lint

REFERENCES

[1] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don'T Touch My Code!: Examining the Effects of Ownership on Software Quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 4–14.
[2] Kathy Charmaz. 2014. *Constructing grounded theory*. Sage Publishing, London, UK.
[3] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M. Y. Wong. 1992. Orthogonal defect classification-a concept for in-process measurements. *IEEE Transactions on Software Engineering* 18, 11 (Nov 1992), 943–956.
[4] MITRE Corporation. 2017. Common Weakness Enumeration. https://cwe.mitre.org/. (2017). [Online; accessed 10-November-2017].
[5] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135 – 146.
[6] Oliver Hanappi, Waldemar Hummer, and Schahram Dustdar. 2016. Asserting Reliable Convergence for Configuration Management Scripts. *SIGPLAN Not.* 51, 10 (Oct. 2016), 328–343.
[7] P. Hudak. 1998. Modular domain specific languages and tools. In *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*. 134–142.
[8] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* (1st ed.). Addison-Wesley Professional.
[9] Yujuan Jiang and Bram Adams. 2015. Co-evolution of Infrastructure and Source Code: An Empirical Study. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE Press, Piscataway, NJ, USA, 45–55.
[10] Nachiappan Nagappan and Thomas Ball. 2005. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*. ACM, New York, NY, USA, 284–292.
[11] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 521–530.
[12] Alberto S. Nunez-Varela, Hector G. Perez-Gonzalez, Francisco E. Martinez-Perez, and Carlos Soubervielle-Montalvo. 2017. Source code metrics: A systematic mapping study. *Journal of Systems and Software* 128 (2017), 164 – 197.
[13] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor, M. Stumm, S. Whitaker, and L. Williams. 2017. The Top 10 Adages in Continuous Deployment. *IEEE Software* 34, 3 (May 2017), 86–95.
[14] A. Pecchia and S. Russo. 2012. Detection of Software Failures through Event Logs: An Experimental Study. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. 31–40.
[15] Akond Rahman, Asif Partho, David Meder, and Laurie Williams. 2017. Which Factors Influence Practitioners' Usage of Build Automation Tools?. In *Proceedings of the 3rd International Workshop on Rapid Continuous Software Engineering (RCoSE '17)*. IEEE Press, Piscataway, NJ, USA, 20–26.
[16] Foyzur Rahman and Premkumar Devanbu. 2013. How, and Why, Process Metrics Are Better. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 432–441.
[17] Rian Shambaugh, Aaron Weiss, and Arjun Guha. 2016. Rehearsal: A Configuration Verification Tool for Puppet. *SIGPLAN Not.* 51, 6 (June 2016), 416–430.
[18] 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 '16)*. ACM, New York, NY, USA, 189–200.
[19] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[20] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 321–332.
[21] John Viega and Gary McGraw. 2011. *Building Secure Software: How to Avoid Security Problems the Right Way (Paperback) (Addison-Wesley Professional Computing Series)* (1st ed.). Addison-Wesley Professional.
[22] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. 2016. Towards Building a Universal Defect Prediction Model with Rank Transformed Predictors. *Empirical Softw. Engg.* 21, 5 (Oct. 2016), 2107–2145.
[23] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2007. Predicting Defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE '07)*. IEEE Computer Society, Washington, DC, USA, 9–15.