# Evaluating the Quality of Open Source Ansible Playbooks: An Executability Perspective

Pemsith Mendis
Auburn University
Auburn, Alabama, USA
jpm0011@auburn.edu

Wilson Reeves
Auburn University
Auburn, Alabama, USA
wgr0009@auburn.edu

Muhammad Ali Babar
University of Adelaide
Adelaide, Australia
ali.babar@adelaide.edu.au

Yue Zhang
Auburn University
Auburn, Alabama, USA
yzz0229@auburn.edu

Akond Rahman
Auburn University
Auburn, Alabama, USA
akond@auburn.edu

## ABSTRACT

Infrastructure as code (IaC) is the practice of automatically managing computing platforms, such as Internet of Things (IoT) platforms. IaC has gained popularity in recent years, yielding a plethora of software artifacts, such as Ansible playbooks that are available on social coding platforms. Despite the availability of open source software (OSS) Ansible playbooks, there is a lack of empirical research on the quality of these playbooks, which can hinder the progress of IaC-related research. To that end, we conduct an empirical study with 2,952 OSS Ansible playbooks where we evaluate the quality of OSS playbooks from the perspective of executability, i.e., if publicly available OSS Ansible playbooks can be executed without failures. From our empirical study, we observe 71.5% of the mined 2,952 Ansible playbooks cannot be executed as is because of four categories of failures.

## CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**.

## KEYWORDS

Ansible, data quality, devops, executability, infrastructure as code

**ACM Reference Format:**

Pemsith Mendis, Wilson Reeves, Muhammad Ali Babar, Yue Zhang, and Akond Rahman. 2024. Evaluating the Quality of Open Source Ansible Playbooks: An Executability Perspective. In *Proceedings of the 4th International Workshop on Software Engineering and AI for Data Quality in Cyber-Physical Systems/Internet of Things (SEA4DQ '24), July 15, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3663530.3665019


*SEA4DQ '24, July 15, 2024, Porto de Galinhas, Brazil*

ACM ISBN 979-8-4007-0672-1/24/07
https://doi.org/10.1145/3663530.3665019

## 1 INTRODUCTION

Infrastructure as code (IaC) is the practice of automatically managing computing platforms [7, 10], such as Internet of Things (IoT) platforms [18]. IaC has gained popularity in recent years, yielding multiple benefits for organizations. For example, use of Ansible scripts was one of the contributing factors for IxN Engineering to support over 7,000 IoT devices [2].

The popularity and usefulness of IaC languages have triggered interest amongst researchers and industry practitioners alike [13]. Such interest has played a pivotal role in generating datasets of IaC scripts, such as open source software (OSS) Ansible playbooks. Generation of IaC-related datasets could be potentially of use to practitioners. However, there is a lack of empirical research on the quality of OSS IaC scripts, which may prohibit practitioners in gaining the full advantage of using IaC scripts. One aspect of evaluating quality of publicly available OSS Ansible playbooks is investigating and quantifying executability, i.e. if playbooks can be executed without generating failures. A systematic empirical investigation can provide insights on the quality of publicly-available Ansible playbooks from the perspective of executability, which till date remains under-explored. Accordingly, we answer the following research questions:

- **RQ1**: *How frequently can we execute publicly-available Ansible playbooks without failures?*
- **RQ2**: *What categories of failures occur when executing publicly-available Ansible playbooks?*

We conduct an empirical study with 2,952 OSS Ansible playbooks to answer our research questions. From our execution process, we separate out failures, and their corresponding crash reports that occurred during execution. With the collected crash reports, we apply open coding [20] to derive the categories of failures that occur while executing Ansible playbooks.

```yaml
---
- name: Update Application servers
  hosts: appservers
  remote_user: root

  tasks:
  - name: Install package on ABC
    yum:
      name: ABC
      state: latest

  - name: Write to a file
    template:
      src: /srv/abc.j2
      dest: /etc/abc.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Install package on XYZ
    yum:
      name: XYZ
      state: latest
```

**Listing 1: Example Ansible Playbook**

**Contributions**: We list our contributions as follows:

- A categorization of failures that occur when executing publicly-available Ansible playbooks; and
- An evaluation of the executability of publicly-available Ansible playbooks.

## 2 BACKGROUND AND RELATED WORK

In this section, we provide necessary background and discuss related work in this domain.

### 2.1 Background

Infrastructure as code (IaC) is the practice of automatically managing computing platforms [7, 10]. IaC uses code to automatically provision infrastructure including physical and virtual resources and environments at scale [7]. IaC scripts need to be well designed, tested and version controlled the same as any other software code produced [9]. IaC languages, such as Ansible use a state reconciliation approach using which it identifies the differences between the desired state and the actual state of the infrastructure, and apply changes only if differences exist [15].

We provide background information on Ansible as it is one of the most popular languages to implement IaC [6, 12, 14]. An Ansible script is developed using Yet Another Markup Language (YAML) [1, 6] where Ansible users provide the tasks that need to be executed to setup necessary computing infrastructure. A playbook is an Ansible script with one or multiple 'tasks' that are executed against a specified host or host groups within the inventory. One of more tasks can be grouped into a 'play'. There can be one or more plays within a playbook, and each play can be mapped to a specific host or a host group [1] as shown in Listing 1. When executing the playbook the tasks in play 'Update Application servers' will be run across all application servers and the tasks in play 'Update db servers' will run against all database servers.

**Table 1: Dataset Attributes**

| Attribute | Value |
|---|---|
| Total Repositories | 56 |
| Total Ansible Playbooks | 2,952 |
| Total Commits | 240,775 |
| Total Number of Lines of Code | 163,769 |

### 2.2 Related Work

Our paper is closely related with prior research that have used program analysis in the context of IaC. We observe majority of the research publications to rely on coding pattern-based static analysis to identify defects in IaC scripts. For example, Hassan and Rahman [6] used static analysis to identify anti-patterns in Ansible test code. As other examples, researchers in separate publications developed static analysis tools to detect security weaknesses in Ansible [11, 14, 17], Chef [17], and Puppet [16, 19] scripts. Researchers [21] have also used static analysis to investigate code properties that cause maintainability problems. In short, we observe a lack of research related to measuring quality of IaC scripts with respect to executability, which we address in our paper.

## 3 METHODOLOGY

We describe the methodology for RQ1 and RQ2 in this section.

### 3.1 Methodology for RQ1

*3.1.1 Playbook Collection.* We use a publicly available dataset of Ansible playbooks curated by Hassan and Rahman [6]. The dataset contains 2,952 playbooks mined from 56 OSS repositories. Attributes of the downloaded repositories are available in Table 1.

*3.1.2 Executing Ansible Playbooks.* We use an Ansible controller and one managed node [3]. An Ansible controller is a control node where the machine in which Ansible command line interface tools are executed. A managed node is target node that Ansible control node will manage [5]. The control node has a memory of 8GB and 2 CPUs, and the managed node is assigned a memory of 2GB and 2 CPUs. Both nodes use the CentOS Stream 9 operating system.

*3.1.3 Methodology to Answer RQ1.* We answer RQ1 by executing the collected set of 2,952 playbooks using our execution harness described in Section 3.1.2 once. We use 'Ansible Runner' [4] a tool that can execute Ansible playbooks. 'Ansible Runner' is available as a package as part of Ansible Tower/AWX [4]. Upon completion, we separate out playbooks for which we observe no failures. A failure is termination of the ability of a playbook to perform a required function [8], which can generate a crash. We report the proportion of playbooks with no failures.

### 3.2 Methodology for RQ2

We answer RQ2 using the following steps:

**Multi-round Execution of Ansible Playbooks**: We answer RQ2 by performing multiple rounds of playbook execution. For each round of execution, we record the count of failures. For each failure, we obtain the corresponding crash report. We stop execution if there are no failure categories for which >= 100 crashes appear.

**Crash Report Collection**: We separate out playbooks for which we observe failures using the methodology in Section 3.1.2. We also collect the crash reports for each playbook that generated a failure.

**Qualitative analysis**: We apply a qualitative analysis technique called open coding [20] on the obtained text from the collected crash reports. Open coding is a qualitative analysis technique that identifies themes or categories from unstructured text sources, such as crash reports [20]. Upon application of open coding, we derive failure categories where each category is mapped to an exception for which the crash occurred.

**Deriving Fix Strategies**: For each identified failure type, we *first* examine the corresponding crash report from where we identify five strings. This examination is performed by the first author. *Second*, we use the Google search engine in incognito mode to analyze Internet artifacts, such as blog posts. We read the first 10 search results to derive the fix strategy for the exception in the crash. *Third*, we apply the fix strategy obtained from the previous step for a set of 10 playbooks that correspond to the failure category. If the fix strategy repairs the exception for 10 playbooks for which the failure occurred, then we apply the strategy for all playbooks. If crash count is >= 100 we identify and apply fix strategies to resolve the failure, and apply another round of execution. We stop execution if there are no failure categories with >= 100 crashes.

## 4 RESULTS

We provide answers to our research questions as follows:

***Answer to RQ1***. : We are able to execute 841 of the 2,952 playbooks (28.5%) in our dataset without any failures. The remaining 2,110 playbooks (71.5%), the execution resulted in failures.

***Answer to RQ2***. : From our qualitative analysis with 2,110 crash reports, we derive four failure categories that are described below. A mapping of each category and their corresponding exception message is available in Table 2. The 'Total' row represents the count of exceptions obtained for 'Iteration-0', 'Iteration-1', and 'Iteration-2'. We stop at 'Iteration-2' because after 'Iteration-2' we do not observe the crash count to be >= 100 for any of the four failure categories.

I-**Environment Inference**: This category is defined as failures that occur due to incorrect assumptions, references, or inferences in the system's operating environment. We identify 10 types of exceptions for this category as shown in Table 2.

II-**Deprecation**: This category is defined as failures that occur due to components or features in the software that have become obsolete. We identify three types of exceptions, as shown in Table 2.

III-**Play Semantics**: This category is defined as failures that occur because of how Ansible interprets and/or executes commands. We found 13 types of exceptions for this category, as shown in Table 2.

IV-**File Operations**: This category is defined as failures that occur because of file operations such as read, write, and delete. We find three types of exceptions related to file operations.

*Fix Strategies*: We identify the following strategies to fix crashes:

**Table 2: Error Types and Error Counts by Iteration**

| Category | Exception | Iter.-0 | Iter.-1 | Iter.-2 |
|---|---|---|---|---|
| Environment Inference | Role was not found | 1185 | 180 | 180 |
| | Missing host pattern | 514 | 51 | 56 |
| | Module/action missing | 282 | 109 | 109 |
| | Field 'hosts' has an invalid value | 15 | 105 | 2 |
| | 'ANSIBLE_REPO_PATH' is undefined | 4 | 4 | 4 |
| | 'import_playbook_role_-name' is undefined | 4 | 7 | 7 |
| | Hosts list cannot be empty | 1 | 1 | 1 |
| | Could not resolve action | 1 | 1 | 1 |
| | Could not find specified file in role | 0 | 31 | 31 |
| | Error when evaluating variable in import path | 0 | 2 | 2 |
| Deprecation | Deprecated and has been removed | 36 | 34 | 34 |
| | Invalid old style role requirement | 1 | 1 | 1 |
| | The use of 'user' is deprecated | 1 | 1 | 1 |
| Play Semantics | Not a valid attribute for a Play | 23 | 23 | 23 |
| | Conflicting action statements | 13 | 29 | 29 |
| | Requested handler was not found | 5 | 5 | 5 |
| | Malformed | 4 | 4 | 4 |
| | Invalid vars_prompt data structure | 4 | 4 | 4 |
| | Not a valid attribute for a Task | 2 | 8 | 8 |
| | Unexpected parameter type in action | 2 | 2 | 2 |
| | Unexpected Exception | 1 | 9 | 9 |
| | Invalid options for import_tasks | 1 | 1 | 1 |
| | Included task files must contain a list of tasks | 1 | 1 | 1 |
| | Unable to read either as JSON nor YAML | 1 | 1 | 1 |
| | Expected a string but got object | 1 | 1 | 1 |
| | Task has extra params | 0 | 1 | 1 |
| File Operation | Attempting to decrypt but no vault secrets file found | 4 | 6 | 6 |
| | Unable to retrieve file contents | 3 | 4 | 4 |
| | Unable to retrieve documentation | 1 | 1 | 1 |
| Total | | 2,110 | 627 | 524 |

*Adding roles locations in roles_path parameter*: We add locations that contain roles as values for the 'roles_path' configuration parameter in 'ansible.cfg' to fix the failure related to 'role was not found'.

*Adding missing host groups*: We add the relevant host groups within the referenced inventory file. We identify the referenced host groups with a Python script that iterates through all playbooks within the dataset, and output a list of host groups in text format that can be easily appended to the existing inventory file or hosts file.

*Adding module locations in library parameter*: We add all possible locations of modules within the dataset by adding paths in the 'library' configuration parameter within the ansible.cfg file.

*Changing host group in playbook*: We resolve the failure due to 'Field *hosts* has an invalid value' by changing the *hosts* key value to an already existing host group within the inventory file. We use a Python script that iterates through all playbooks within the dataset and programmatically update the *hosts* key value if the original *hosts* key value has a variable referenced.

In the first iteration, i.e., 'Iteration-0' we do not apply any fix strategies. With the identified fix strategies, we execute the collected Ansible playbooks in two iterations: (i) in the first iteration, upon applying the four fix strategies 2,326 out of 2,952 playbooks (78.8%) are executed without failures; and (ii) in the second iteration, upon reapplying the four fix strategies 2,421 out of 2,952 playbooks (82.2%) are executed without failures.

# 5 DISCUSSION AND CONCLUSION

## 5.1 Discussion

*5.1.1 Implications.* The implications of our empirical study are:

**Implication-1**: Using our four fix strategies listed in Section 4 we increase the proportion of executable playbooks from 28.5% to 82.2%. The implication of this finding is that fix strategies obtained from crowdsourced knowledge can aid in fixing failures attributed to publicly available playbooks. Both researchers and practitioners can use our identified set of strategies as heuristics on how to resolve playbook-related failures.

**Implication-2**: Our fix strategies are not comprehensive as 17.8% of the playbooks still result in failures after applying the four strategies. Researchers can further investigate what other strategies can aid in comprehensive playbook executability.

*5.1.2 Threats to Validity.* The limitations of our paper are:

**Conclusion Validity**: Our paper relies on a specific execution harness to execute the collected set of Ansible playbooks. This can influence the frequency of executability of the collected playbooks, as well as the identified categories of failures.

**External Validity**: Our paper is susceptible to external validity as we only use one dataset provided Hassan and Rahman [6]. Our findings may not generalize to other OSS and proprietary datasets. Furthermore, the reported executability rate may be different for other types of data sources, such as GitHub Gists.

## 5.2 Conclusion

While there has been a plethora of publicly-available IaC artifacts, such as Ansible playbooks, there has been a lack of understanding about whether these artifacts can be executed to perform infrastructure-related operations. To address this, we have conducted an empirical study to assess the executability of publicly available Ansible playbooks. From our empirical study, we have found that 28.5% of 2,952 Ansible playbooks can be executed without generating failures. We have also observed that using four fix strategies derived from crowdsourced knowledge sources, 82.2% of the playbooks can be executed. Based on our findings, we recommend practitioners to use our identified set of fix strategies as heuristics to resolve playbook-related failures.

# ACKNOWLEDGMENTS

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141 and # 2312321. This work has benefited from Dagstuhl Seminar 23082 "Resilient Software Configuration and Infrastructure Code Analysis."

# REFERENCES

[1] Ansible. [n. d.]. Ansible playbooks. https://docs.ansible.com/ansible/latest/. [Accessed 29-09-2023].
[2] Ansible. 2019. AnsibleFest Atlanta - Scaling Ansible for IoT Deployments. https://www.ansible.com/scaling-ansible-for-iot-deployments
[3] Ansible. 2023. Ansible community documentation. https://docs.ansible.com/. [Online; accessed 19-December-2022].
[4] Ansible. 2023. Ansible Runner ansible-runner documentation. https://ansible.readthedocs.io/projects/runner/en/stable. [Accessed 29-09-2023].
[5] Ansible. 2024. Network Getting Started. https://docs.ansible.com/ansible/latest/network/getting_started/index.html. [Online; accessed 19-December-2023].
[6] Mohammad Mehedi Hassan and Akond Rahman. 2022. As code testing: Characterizing test quality in open source ansible development. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 208–219.
[7] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* (1st ed.). Addison-Wesley Professional.
[8] IEEE. 2010. IEEE Standard Classification for Software Anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)* (2010), 1–23. https://doi.org/10.1109/IEEESTD.2010.5399061
[9] John Klein. 2019. INFRASTRUCTURE AS CODE–FINAL REPORT. https://api.semanticscholar.org/CorpusID:225061723
[10] NIST. 2023. infrastructure as code. https://csrc.nist.gov/glossary/term/infrastructure_as_code. [Online; accessed 25-Sep-2023].
[11] 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 *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 534–545. https://doi.org/10.1109/MSR59073.2023.00079
[12] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian A. Tamburri. 2020. Towards a Catalogue of Software Quality Metrics for Infrastructure Code. *ArXiv* abs/2005.13474 (2020). https://doi.org/10.1016/j.jss.2020.110726
[13] Chris Parnin, Eric Helms, Chris Atlee, Harley Boughton, Mark Ghattas, Andy Glover, James Holman, John Micco, Brendan Murphy, Tony Savor, Michael Stumm, Shari Whitaker, and Laurie Williams. 2017. The Top 10 Adages in Continuous Deployment. *IEEE Software* 34, 3 (2017), 86–95. https://doi.org/10.1109/MS.2017.86
[14] Akond Rahman, Dibyendu Brinto Bose, Yue Zhang, and Rahul Pandita. 2024. An empirical study of task infections in Ansible scripts. *Empirical Software Engineering* 29, 1 (2024), 34.
[15] Akond Rahman and Chris Parnin. 2023. Detecting and Characterizing Propagation of Security Weaknesses in Puppet-based Infrastructure Management. *IEEE Transactions on Software Engineering* (2023).
[16] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 164–175.
[17] Akond Rahman, Md Rayhanur Rahman, Chris Parnin, and Laurie Williams. 2021. Security Smells in Ansible and Chef Scripts: A Replication Study. *ACM Trans. Softw. Eng. Methodol.* 30, 1, Article 3 (Jan. 2021), 31 pages. https://doi.org/10.1145/3408897
[18] RED HAT. 2021. Tips on managing IoT devices at the edge with Red Hat Ansible Automation. https://www.redhat.com/en/blog/tips-managing-iot-devices-edge-red-hat-ansible-automation
[19] Sofia Reis, Rui Abreu, Marcelo d'Amorim, and Daniel Fortunato. 2023. Leveraging Practitioners' Feedback to Improve a Security Linter. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 66, 12 pages. https://doi.org/10.1145/3551349.3560419
[20] J. Saldaña. 2009. *The Coding Manual for Qualitative Researchers*. Sage. https://books.google.co.in/books?id=OE7LngEACAAJ
[21] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 189–200. https://doi.org/10.1145/2901739.2901761

Received 2024-04-05; accepted 2024-05-04