Analysis of Ansible and Puppet Softwares

Prashanth Mallyampatti, Pratik Abhyankar, Ramya Ananth, Sushmitha Natanasabapathy
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{pmallya,pabhyan,rananth2,snatana}@ncsu.edu

***Abstract*—In this project we compare Ansible and Puppet, two similar tools used primarily in the field of Development & Operations(DevOps) in Software Engineering. These tools help in provisioning, configuration management, app deployment, continuous delivery (CD) and orchestration. In order to formulate the comparison, we rely on our observations based on several different software metrics such as usability, stability & robustness, performance, platform compatibility and maintainability. We also take into account the data that we have gathered from various sources on the internet such as Github, StackOverflow, Technology Blogs such as HackerNews, Reddit threads and User Forums to understand the popularity, user community and their contributions to these tools. Along with all other metrics, a detailed analysis of the source codes of these tools is also performed to draw a rational conclusion from this comparison.**

I. Overview

Development Operations(DevOps)[2] is an emerging field in Software Development which focuses on provisioning nodes on different cloud environments, configuring & orchestrating these nodes, performing continuous integration(CI) and continuous deployment(CD), thereby automating the entire application deployment process. While the two open source engines, Ansible and Puppet, help in performing all or most of the DevOps tasks, they do so using different approaches and implementations. It can be difficult for one to choose between these two softwares and decide which one best suites the requirements of their use-case[2].

In this project, we compare Ansible and Puppet across a wide variety of metrics, ranging from source code analysis, robustness, space & time complexity, performance etc. to platform compatibility, maintainability as well as popularity, pricing and usability which are a few key points in choosing a software from an end-user's perspective. Our study contains both qualitative and quantitative analysis of these two softwares using tools such as SonarQube, Pylint, Rubocop, Molecule and Docker in conjunction with our own home-grown scripts.

Fig. 1. DevOps pipeline

II. Approach

We perform our study in three parts. The first part involves static code analysis of the source codes of Ansible and Puppet that are available on GitHub. Here, we analyze the lines of code, technical debts, vulnerabilities, code smells, duplicate lines of code, cyclomatic & cognitive complexity[1] and bugs using a tool called SonarQube[4]. Since Ansible is written in Python, we also used Pylint which finds possible bugs and checks for the quality of Python code which could determine the reliability and performance of Ansible. Similarly, Rubocop is used for Puppet as it is primarily written in Ruby.

The second part involves the following metrics: Usability that is, the ease of learning and using these softwares regardless of the user's experience level with the help of documentation, and other interactive resources. Platform compatibility and maintainability of both the scripts specific to these softwares and the softwares itself. For assessing the robustness that is, the ability of a software to cope up with erroneous input/execution, we use Molecule & Docker for Ansible, and Rspec for Puppet, along with custom scripts.

In the last part, from an end user's perspective we gather data from GitHub, StackOverflow, and various user forums such as Reddit to assess code-issues, mean time to close an issue, truck-factor, contributor analysis, tags and answered/unanswered questions.

III. Results

A. Static Code Analysis and Quantitative Metrics

1) Program Size
   a) Table I describes the size of Ansible and Puppet over different parameters as defined by SonarQube. These include, size & lines of code, total number of files and the percentage of commented code.
   b) It is observed that size of Ansible code is almost 4 times larger than Puppet. Thus, it looks much more code intensive.

TABLE I

Parameter	Ansible	Puppet
Size of code	99 MB	23 MB
Lines of Code	935,962	260,156
Number of files	4,369	2,255
Comment coverage	8.6%	12.5%

2) Bugs and Reliability
   a) We observed that SonarQube ranked Ansible and Puppet with a rating of 'C', in terms of reliability

based on the number of bugs detected, their severity and the time required to remediate them.

b) There were 49 major bugs reported for Ansible and 18 for Puppet, most of which were due to the usage of same operands on either side of a binary operator as shown in Fig. 2. These bugs tend to consume more computing power and software resources. Also, both programs contain a lot of magic numbers and hard coded strings in the code which decrease their readability.

c) Ansible has a ratio of 0.00005 bugs/line whereas Puppet has 0.00007 bugs/line, both of which are negligible.

```
elif self.state == 'present':
    if self.name is not None and not self.name == \
            self.name:
        changed = True
        rename_qtree = True
```

Fig. 2. Ansible code snippet using hard coded string *present* and comparing the two same operands in *if* predicate

3) Security and Vulnerabilities

a) SonarQube did not report any true positive vulnerabilities in either Ansible or Puppet source code. However, logical bugs that might lead to security vulnerabilities during run time were not detected by performing a static code analysis.

4) Code Smells and Maintainability

a) We observed that both of the software source codes have negligible amount of code smells. However, they have a lot of unimplemented empty methods and quite a few of them are commented with *TODO* and *FIXME* tags. Very less hard coded strings or magic numbers are used in either of the programs.

b) Since there are so few code smells and the efforts required to rectify them are minimal, SonarQube has rated maintainability of source codes for both Ansible and Puppet as an 'A'.

B. Qualitative Software Metrics

1) Usability

a) Here, we measure usability in terms of ease of learning and using these softwares using documentation and other resources. We also check the quality of official documentation, interactive video tutorials and active involvement of the user community in developing these softwares.

b) For this, we reached out to other developers and peers over Piazza, Google Forms, Quora etc. Out of the 44 responses that we received, 81.2% were in favor of Ansible and remaining 18.8% for Puppet. Also, we observed that Ansible has extensive official video tutorials as compared to Puppet which has nil. However, Ansible has no GUI supported except Ansible-Tower (paid version) whereas Puppet comes with full featured GUI for community as well as enterprise editions which improves its usability drastically.

c) Overall, Ansible still gets an upper hand over Puppet in this metric considering all the above factors.

2) Platform Compatibility

a) Ansible and Puppet, both, can be easily installed on MacOS platform and Linux distributions such as Ubuntu, CentOS etc. using the native package managers.

b) Puppet has x32 and x64 bits binary packages for Windows platform, however, Ansible does not. In order to run Ansible on Windows, we require a simulated Linux-like environment and the entire process to our observation is very time consuming, tedious and error-prone.

3) Stability and Robustness

a) The code breaks whenever there occurs a problem in the code even when the "ignore_errors:yes" attribute is included.The Puppet code breaks unless a separate catch block is written to handle the error.

b) Robustness of Puppet is evaluated by running multiple threads of the same script. Since Puppet stores both current and desired end state of a node, running multiple threads of same script does not disrupt the functionality. The functionality test is done using Rspec to check the desired end state. Robustness of Ansible is tested by running a Molecule test in Docker. This is done by building multiple roles and testing the software on multiple platforms using Docker containers. This process is time consuming as it involves modifying the docker images specific to Ansible and changing the permission of Docker with respect to the OS. We can still verify that Ansible runs over multiple OS environments and remains stable throughout.

C. User Community Metrics

1) GitHub was the first user community that we analyzed upon. Open/Close Issue, Truck factor, Contributor and Commit analysis was done over the period of 6 years of data. The following table presents GitHub insights for Ansible and Puppet.

The average bug life time was calculated using random samples of 30 recently closed issues. Some of them had a turn around time as less as 1 day, while some had their turn around time as high as 320 days. The actual bug life time is subject to vary considering the entire Issue-list 16000+ for Ansible and 7000+ for Puppet. The truck factor, here, is calculated by analyzing the contributions made by the most major contributors for the development of a repository.

TABLE II

Parameter	Ansible	Puppet
Total Contributors	4000	508
Average Bug Lifetime (days)	59	59.6
Past 1 year Commits (per week)	120	30
Truck Factor	5	9
Commit Activity (Oct 2018)	232 authors -711 to master, 976 to all branches	14 authors- 53 to master, 70 to all branches

2) Stack Overflow measures included number of Tags, Questions and Unanswered Questions. Table III presents these figures.

TABLE III

Parameter	Ansible	Puppet
Tags	8,378	3,504
Questions	10,104	3,504
Unanswered Questions	2,143	1,122

Tags and questions are from the core modules of Ansible and Puppet. These do not include tags for other secondary related modules such as Ansible-Galaxy or Puppeteer. However, the secondary module questions can have the main core module tags included.

By considering the above GitHub and Stack OverFlow metrics, we can conclude that Ansible has more user community support when compared to Puppet.

IV. Discussion

In this section, we discuss the main findings of this study:

1) With respect to source code analysis, both, Ansible and Puppet are very well implemented and maintained by the open source community. Most of the parameters of these two softwares are in fair comparison with one another.
2) Considering the usability metric, Ansible clearly gains an upper hand over Puppet with its extensive learning material and simple usage. Puppet tries to cover this gap by providing excellent GUIs for all OS platforms, whereas Ansible lacks a proper GUI support.
3) The unique feature of Ansible is that it focuses on task-based configuration management, while, Puppet is more centric to node-based management.
4) Puppet supports the entire DevOps pipeline using its different modules such as Enterprise, Discovery and Pipeline. Ansible on the other hand does provide many enterprise grade solutions such as Tower and Galaxy to support orchestration and automation.
5) Further, we observe that Ansible has a greater community support, making it more popular amongst the audience, as compared to Puppet.
6) Finally, Ansible is a better priced enterprise solution than Puppet.

V. Conclusion

As per our study, we conclude that Ansible would be a preferred choice of the end-user if they plan on getting up-and-running quickly with minimal learning curve, using its extensive resources and community support. Ansible has an edge over Puppet when it comes to pricing and popularity due to a strong backing by Redhat, which might be an important deciding factor in enterprises. Puppet on the other hand has unique features that focuses on complete automation of DevOps pipeline, unlike Ansible which is primarily used for configuration management and orchestration. Convenient GUI support and better cross platform compatibility of Puppet gives a strong competition to Ansible. We can safely infer that a choice between these two softwares really depends on the actual use-case and the primary focus of the user.

VI. Appendix

SonarQube congurations for metric rating:

1) Maintainability:
   Remediation cost of the detected code smell, that is, technical debt ratio is divided as:
   a. Less than or equal 5% of the time that has already gone into the application, the rating is A
   b. between 6 to 10% the rating is a B
   c. between 11 to 20% the rating is a C
   d. between 21 to 50% the rating is a D
   e. anything over 50% is an E
2) Reliability:
   A = 0 Bugs
   B = at least 1 Minor Bug
   C = at least 1 Major Bug
   D = at least 1 Critical Bug
   E = at least 1 Blocker Bug
3) Vulnerability and security:
   A = 0 Vulnerabilities
   B = at least 1 Minor Vulnerability
   C = at least 1 Major Vulnerability
   D = at least 1 Critical Vulnerability
   E = at least 1 Blocker Vulnerability

VII. References

[1]K. Torberntsson and Y. Rydin, A Study of Configuration Management Systems: Solutions for Deployment and Configuration of Software in a Cloud Environment, Dissertation, 2014.
[2]Ebert, Christof, et al. "DevOps." IEEE Software 33.3 (2016): 94-100.
[3]Venezia, Paul. "Review: Puppet vs. chef vs. ansible vs. salt." InfoWorld 21 (2013).
[4]Garca-Munoz, Javier, Marisol Garca-Valls, and Julio Escribano-Barreno. "Improved metrics handling in SonarQube for software quality monitoring." Distributed Computing and Artificial Intelligence, 13th International Conference. Springer, Cham, 2016.