Code Smells in Infrastructure as Code

Infrastructure as code (IaC) simplify the provision and configuration of the IT infrastructure at scale. Enterprises are increasingly adopting IaC. For example, Intercontinental Exchange, who owns exchanges for financial and commodity markets, and operates 12 regulated exchanges and marketplaces, maintains 75% of its 20,000 servers using IaC scripts [1]. As the size and complexity of the IaC projects increase, it is critical to maintain the code and design quality of IaC Scripts.

Infrastructure as Code Landscape

There are many tools that support IaC, for example, Puppet [2], Chef [3], Ansible [4], Pulumi [5], SaltStack [6], CFEngine [7], and Terraform [8]. The key programming models of IaC supported by these tools are declarative and imperative. For example, Puppet and Ansible are declarative, while Chef uses a mix of both approaches. The declarative approach enables specifying what the desired infrastructure design and configuration is, while the imperative focuses on how the infrastructure should be created and changed to meet the desired design and configuration.

What are Software Smells?

A software smell is any characteristic in the artifacts of the software that possibly indicates a deeper problem or quality issue. We can categorize software smells into different types such as architecture smells, design smells, implementation/code smells, and configuration smells [9]. The software smells can negatively impact software quality attributes such as maintainability, effort/cost, reliability, change proneness, testability, and performance.

Code Smells in Infrastructure as Code

Several studies [1, 10-11] have identified different types of smells in IaC scripts, particularly for Chef and Puppet. Sharma et, al. [10] identified 21 implementation and design smells for Puppet.

- Missing default case - No default case in the switch statement
- Incomplete Tasks - Tasks have TODO or FIXME statements
- Complex Expression - Conditional expressions are difficult to be read and understood.
- Duplicate Entity - Duplicate hash keys or duplicate parameters
- Misplaced Attribute - Ordering of attributes does not follow the recommendations.
- Improper Alignment - Use of TAB characters and inconsistent indentation
- Invalid Property Value - Values of properties or attributes are invalid with respect to the corresponding type definition
- Deprecated Statement Usage - Use one or more the deprecated statements in the code
- Improper Quote Usage - Single and double quotes are not used properly.
- Long Statement - The length of a statement exceeds the recommended maximum length.
- Incomplete Conditional - Conditional branches are complete
- Unguarded Variable - A variable is not enclosed in braces when being interpolated in a string.

Rahman et, al. [1] identified seven common security smells in Puppet scripts.

- Admin by default - The default user will be "admin" with full access privileges.
- Empty password - Use a string of length zero for a password
- Hard-coded secret - Sensitive information such as username and passwords in IaC scripts
- Invalid IP address binding - Use of the address 0.0.0.0 for a database server or a cloud service/instance
- Suspicious comment - A comment states about the presence of defects
- Use of HTTP without TLS - Using HTTP without the Transport Layer Security (TLS)
- Use of weak cryptography algorithms - Use of known weak cryptography algorithms MD4 and SHA-1 for encryption purposes

IaC Smell Detection in SODALITE

SODALITE simplifies the use of heterogeneous IaC approaches in designing and runtime reconfiguration of the deployment models of cloud and HPC applications. The SODALITE approach is built on top of the TOSCA (Topology and Orchestration Specification for Cloud Applications) standard. A key task in SODALITE is to detect and correct defects (smells or anti-patterns) in IaC code scripts, TOSCA configuration files, deployment workflows, and infrastructure designs. The design support in SODALITE is grounded on ontology engineering and model-driven architecture. Thus, defect detection and correction also follows a semantic model-driven

approach. In addition, we investigate the use of machine learning and NLP (natural language processing) techniques for detecting specific types of defects such as linguistic anti-patterns.

References

1. Rahman, Akond, Chris Parnin, and Laurie Williams. "The seven sins: security smells in infrastructure as code scripts." Proceedings of the 41st International Conference on Software Engineering. IEEE Press, 2019.
2. Puppet Documentation.https://puppet.com/docs
3. Chef Documentation. https://docs.chef.io/
4. Ansible Documentation: https://docs.ansible.com/
5. Pulumi Documentation: https://www.pulumi.com/
6. Saltstack Documentation: https://docs.saltstack.com/en/latest/
7. CFEngine: https://cfengine.com/
8. Terraform Documentation: https://www.terraform.io/docs/index.html
9. Sharma, Tushar, and Diomidis Spinellis. "A survey on software smells." Journal of Systems and Software 138 (2018): 158-173.
10. 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.
11. Schwarz, Julian, Andreas Steffens, and Horst Lichter. "Code Smells in Infrastructure as Code." 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC). IEEE, 2018.