A study of the challenges within IaC, and the best practices to potentially overcome them.

May 2, 2022

DD2482 - Automated Software Testing and DevOps

Philip Salqvist

Contents

1 Introduction

The use of Infrastructure as Code (IaC) has become largely popular in the field of DevOps during recent years. This becomes quite obvious when examining Figure 1, where data from Google Trends describing the frequency of searches containing the phrase "Infrastructure as code" is presented. In parallel with the adoption of IaC in industry, scientific research in the field has been growing substantially as well, which can be illustrated by typing in "Infrastructure as code" in the Google Scholar search engine, giving over 3 million hits currently.

The benefits of implementing IaC are quite obvious when properly executed. It essentially means to take the principles of software development, and apply those to infrastructure and configuration. By doing so, one can apply tools from software development such as Version Control Systems (VCS), automated testing libraries and deployment orchestration. Furthermore, it introduces proper testing alternatives to the world of infrastructure by using techniques such as Test Driven Development (TDD), Continuous Integration (CI) and Continuous Deployment (CD). [4] By implementing IaC, organizations can possibly achieve performance increases in cost, speed and risk-minimization. [3]

While IaC and the benefits that it brings clearly is a fairly well research area, some subareas within the field are not as thoroughly examined. A systemic mapping study of infrastructure as code research from 2019 [5] presents a number of potential avenues for future research. The author of the study recommends researchers to focus on both challenges within the industry of IaC, as well as outlining the best practices that are used in the industry. This essay will combine these areas by presenting the challenges that practitioners of IaC face, as well as outlining potential best practices to overcome them.

The problem that this essay attempts to address, is the lack of research within these two areas. Insufficient knowledge regarding the problems and challenges that the IaC practitioners are facing, will make it difficult to create a clear direction for future research and development in creating tools and services that aid these practitioners. By summarizing recent findings from these topics, and by connecting the challenges with best practices that have enabled practitioners to overcome them, one can more clearly see what needs and potential solutions there are to be addressed in future attempts.

The work in this essay resulted in outlining four different main challenges that practitioners experience from the IaC industry, namely struggles concerning testing, automation, integrating tools when they rely on different languages and keeping compatibility across different versions. In the essay these challenges and their potential solutions are respectively referred to as Testing, Polyglot, Version Control and finally Automation. It was also concluded that a lot of

the potential solutions did not only apply to a single challenge, but often seem to have positive effects across multiple challenges.

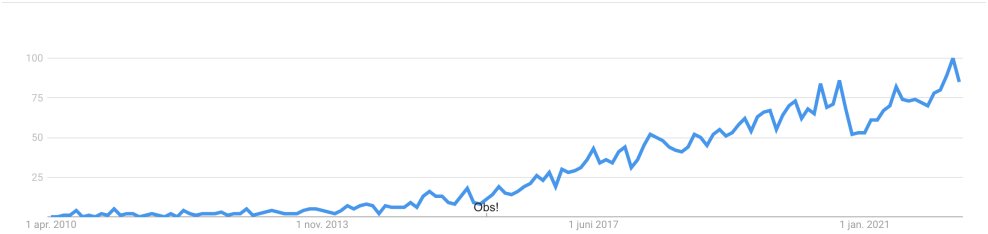

Figure 1: Presenting the search data from Google Trends on the phrase "Infrastructure as code" between the years 2010-2022

2 Results

The results are divided into challenges and best practices concerning IaC. First, challenges are presented, followed by proposed best practices that could enable practitioners to overcome these challenges.

2.1 Challenges with Infrastructure as Code

2.1.1 Testing

As previously mentioned in the introduction, there is a lack in research outlining the challenges of IaC practitioners. However, one academic paper that attempts to address this is written by Guerriero et al. [1]. In their research, 44 interviews are conducted with practitioners in the field, where qualitative data is gathered by the use of a questionnaire. Issues and challenges that are most frequently mentioned by practitioners are then ranked and described in the paper.

The most frequently mentioned challenge regarded testing of IaC, and how a lack of proper testing environments as well as poor testing practices created a negative situation regarding testing of IaC. One practitioner expressed this in the following way [1]:

> "Issues are mostly related to setting up a testing environment, since this is usually quite a complex problem. Combine that with no standard practices when it comes to testing and you have one big mess".

IaC as any other code, is naturally susceptible to defects which could lead to terrible consequences, that could have been found early with proper testing. A good example of this is the Amazon Web Services (AWS) outage in 2017. By eliminating barriers to proper testing, one could potentially avoid similar results in the future. [2]

2.1.2 Polyglot

Furthermore, it was frequently argued that the vast number of tools that is needed for different purposes often created a polyglot system, which decreased the readability of code. While this clearly is a concern in the IaC community, the authors present findings that show how about 20% of practitioners use tools that use the OASIS TOSCA, which is the key standard in the field. This could potentially be evidence that points to flaws in the standard, if tools that use the standard are not adopted by practitioners. [1]

2.1.3 Version Control

Moreover, it is evident from the study that the number of technologies that often are involved and their dependencies create problems in version management and backward compatibility.[1] Morris argues that a lot of these issues stems from the fact that virtualization makes it very easy for developers to introduce new servers from a pool of resources. However, one may not have the resources to maintain and manage these properly. This lack of resources often leads to configuration fixes not being rolled out across all servers, and thereby creating differences between versions and configurations. This could result in some versions running properly on certain machines while not doing so on others. This kind of inconsistency across servers Morris calls configuration drift. Some sources also present the notion that security concerns becomes more difficult to handle when the number of technologies involved together with IaC increases [6], which is another reason to aim at keeping the number of technologies and tools minimized if it can be achieved.

2.1.4 Automation

Finally, it is presented that automation is still rather limited for IaC during runtime. [1] Some sources also indicate that the full capabilities of automation sometimes aren't used due to what the author calls "Automation fear" [4] This could potentially work as a barrier to the full adoption of IaC, which is critical to overcome since automation is a key aspect of the field [7].

2.2 Best practices for Infrastructure as Code

By examining the research done by [1] et al. one can conclude that there does not exist an overarching fit-all-purposes technology for IaC. This has led organizations to adopt different tools and technologies, each differing from one another. One can also note that there has not yet been a large adoption of a standard for IaC. Since the technologies used in IaC are not that mature yet, and a clear standard hasn't been fully adopted, best practices can differ a lot amongst different organizations. This section is an attempt to derive generalized principles that could be applied independent of an organizations particular adoption of certain technologies.

2.2.1 Testing

A number of best practices used in IaC testing has previously been identified by Rahman et al. The top three most frequently used practices for testing was Automation, Sandbox Testing and Continuous Integration (CI). The study shows that practitioners advocate for automation to be used in testing since it reduces a lot of manual work, which then frees up time for developers to focus on essentials like making the test suite more robust. [2]

However, developers of IaC can occasionally suffer from Automation Fear [4], which was discussed in the previous section. Which leads us to the next practice for IaC testing, namely Sandbox Testing. Using Sandbox Testing can ease up the fear of creating defects in the actual system, and can thus likely make developers more prone to use automation practices. [2]

Finally, it is recommended to use CI, which essentially means running all tests on every commit. The study shows that although this practice is time consuming, practitioners finds that the benefits of CI outweigh the costs. This is largely due to the fact that running all tests on every commit makes it more likely to find defects caused by new dependencies introduced to the system that will break compatibility between versions. [2]

2.2.2 Polyglot

One best practice that is mentioned by Guerriero et al. [1] is simply to avoid the combination of multiple languages and formats across IaC platforms. This is naturally dependent on the specifics of a particular project, and combining languages can sometimes not be avoided for one to reach a certain purpose, but in this case, it is at least good practice to aim at minimizing the number of different languages used. This would increase readability and make good maintenance of the project more likely. [1]

2.2.3 Version Control

To tackle the issues related to version control and more specifically to keep compatibility across different versions, as previously mentioned in the testing section, CI is critical. However, running all tests on every commit isn't enough in this case, since a dependency problem doesn't have to be introduced from a commit relating to source code, but might as well be introduced by an update of the OS. That's why it is beneficial to add a regularly scheduled run of the test suite, to make sure that compatibility is not broken between versions due to changes that are external in relation to the source code. [1]

2.2.4 Automation

Once again, a large reason why developers are hesitant to fully adopt automation practices with IaC are due to not trusting the automation and a fear that relying on automation to a too large extent could introduce defects into the system. As previously discussed during the testing section, one way to mitigate this fear is to use Sandbox Testing, to ensure that changes do not break the system in a safe setting, before integrating changes with the real system. [2]

Furthermore, Morris argues that CI is critical to solving the challenge of automation as well. Testing continuously can give developers the confidence to rely more on automation tools. [4] One could argue that using CI for IaC is particularly important since very small changes can have serious consequences, which can be exemplified by the incident that occurred at AWS in 2017 [2].

3 Conclusion

Four main challenges are defined in this essay, namely Testing, Polyglot, Version Control and Automation. The most prevalent issue according to IaC practitioners is the one of testing. Especially, the fact that there is a lack of a comprehensive overarching testing environment seems to be most concerning. This could be a result of the heterogeneity regarding languages and formats amongst different tools in the industry, since different testing environments are needed for different tools and purposes. It can be concluded by looking at previous research in the area that standards are poorly adopted, which could indicate that the standard needs to be adapted to the needs of the industry. The OASIS TOSCA standard was adopted by merely 20% of practitioners that did partake in the study by Guerriero et al. [1]. Finally, one can conclude that the challenges presented in this essay are not isolated from one another. The poor adoption of standards in the field leads to a Polyglot problem, which in turn leads to difficulties in developing proper testing environments that integrate well with different IaC platforms. A result of poor testing practices then leads

to an uncertain environment for developers, which can easily lead to them not fully trusting and thereby use automation practices available. It is not certain that the problems connect to each other in a casual manner as in the example that was just outlined, but there certainly seems to be some dependencies between these challenges. Thus, the topic of developing a standard that is more adapted to the needs of the practitioners in the IaC industry seem particularly relevant, since there seems to be a potential for positive effects across multiple different currently perceived problems. Another example of this is the potential effects that adopting CI in IaC could have. The adoption of CI would probably not only lead to positive effects regarding perceived issues in testing, but would probably also have positive effects regarding the problem of keeping compatibility across different versions of IaC software.

References

[1] Michele Guerriero et al. "Adoption, support, and challenges of infrastructure-as-code: Insights from industry". In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2019, pp. 580–589.

[2] Mohammed Mehedi Hasan, Farzana Ahamed Bhuiyan, and Akond Rahman. "Testing practices for infrastructure as code". In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing*. 2020, pp. 7–12.

[3] *Infrastructure as code*. URL: `https://en.wikipedia.org/wiki/Infrastructure_as_code/`.

[4] Kief Morris. *Infrastructure as code*. O'Reilly Media, 2020.

[5] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. "A systematic mapping study of infrastructure as code research". In: *Information and Software Technology* 108 (2019), pp. 65–77.

[6] *Security Challenges With IAC And How To Overcome Them*. URL: `https://www.ibexlabs.com/security-challenges-with-iac/`.

[7] Clauirton Siebra et al. "From theory to practice: the challenges of a DevOps infrastructure as code implementation". In: *ICSOFT 13th 2018, International Conference on Software Technologies. Porto: Portugal July*. 2018, pp. 26–28.