What are the benefits and issues of using Infrastructure as Code?

André Brogärd
Eva Despinoy

April 2021

Introduction

Infrastructure as Code is a process which is used to manage infrastructures such as servers, networks, data storage, and more, with code instead of with physical hardware [1]. Infrastructure as Code can take different forms, but usual ones are configuration files or scripts. The goal of using Infrastructure as Code is that tasks that touch the configuration of infrastructure, such as provisioning and deployment, are automated. This subject is highly relevant for DevOps, as the goal of DevOps is to automate the software development process by modifying the systems and tools used by organisations to integrate the development and IT operations together [2]. Examples of tools that are commonly used to implement Infrastructure as code are Terraform, AWS CloudFormation or Google Cloud Deployment Manager. Different tools often automate different parts of the process. Three different ones are:

- The infrastructure provisioning, which means for example spinning up new servers or configure networks
- The configuration of the infrastructure, which is for example installing applications on the servers
- The deployment of the applications

There are also different phases in the infrastructure life-cycle, which need different types of management. The first phase is the initial setup phase, the second one is the phase of managing the infrastructure when it is already existing, and the third one is the deletion of infrastructures.

In this essay, we want to analyze the effects of using Infrastructure as Code, and study which positive consequences it can have, and which issues are related to it. We will therefore answer the following question: What are the benefits and issues of using Infrastructure as Code?

Background

Traditional IT organizations that have not adopted DevOps practices struggle with dependencies across teams. The application being dependent on many separate teams limits productivity. When developers lack insights in how the application is used, there may be unnecessary iterations. When the IT department finds problems, only the developers know how to fix it. When the QA teams find issues, other teams may have misunderstood some requirements earlier in the process.

One common approach is to let the app team that builds the application also run it. This means that they are on call whenever something goes wrong, and they provision and configure the infrastructure around the application. They can quickly react to errors that happen during testing stages or even in production. They do not depend on having errors forwarded to them, along with a report, from another team. The work becomes more automatable, more efficient, and allows the organization as a whole to be more productive. When the problem actually belongs to another team, such as IT or QA, then the issue is forwarded to them from the app team. This reversed dependency allows quicker response times, and instead of making the workflow defined to be dependent on other teams, it only involves other teams when necessary.

Another common approach, especially for large organizations, is to design cross-functional teams. One team consists of all necessary competencies for a certain application or service. (product testers, developers, IT, project managers a.s.o). Naturally, the teams have little dependencies on other teams and can work efficiently. Creating these workflows and integrating this into the work of the application team may be difficult and complex. Infrastructure as Code may be the tool that allows the organization to give more control to the application team and enable these workflows in a simpler and more integrated manner.

Infrastructure as Code can be employed to make organizations more productive. Infrastructure as Code is a tool that can allow the app team to control the entire application life-cycle and reduces overhead while working across teams. In the coming sections, we will discuss the benefits and issues of working with Infrastructure as Code.

Benefits

Firstly, using Infrastructure as Code can reduce costs. This is achieved by several effects. One of them is that configuration files or scripts that are created for designing the infrastructures can be reused [1]. If the developer has already created an environment, they can recreate the same one by just duplicating the code. This is useful for example if the developer wants to deploy a development environment and a test environment which are exactly the same, or if they want to deploy the same type of environment in several projects. Also, if they want to change some details in the infrastructure, they do not have to redo the whole installation manually, but can just modify parts of the code which

controls it, and restart another infrastructure based on the new code [1]. This adds to the flexibility and ability to tailor the infrastructure, and make the systems consistent with each other. Another effect that helps reduce costs is that less people need to work on building and managing infrastructure, and therefore less time and money is spent on this task. This is because of the ease of writing and reusability of the code. In addition to this, infrastructure as code is easier for software developers to understand as it is nearer their field [3]. It can also increase visibility to the state and configuration of the infrastructure for them and other users in the enterprise, which can be positive as they become more knowledgeable of it and can help to manage it or tailor the product to the infrastructure, and vice-versa. This goes hand in hand with the DevOps principle of making the development and operations parts of the organisations closer [4]. Also, the increased visibility and possibility to run or take down the server easily makes it simple to reduce the cloud infrastructure to only the necessary, which also reduces costs.

In addition to this, Infrastructure as Code helps with risk management [3]. An effect of any automation is removing human error. There is still a possibility of human error as a human will write the code which creates the infrastructure. However, for instance, in cases where configuration consistency is important, which could be in the scenario where two identical infrastructures need to be deployed, or where an old infrastructure needs to be replaced with a new version of it, there is a big probability of error if a human needs to follow all the same steps for creating the infrastructure manually. This risk is greatly reduced as code can be copy and pasted, and is easier to read. In addition to this, automated tests can be used to check if a mistake was made. Also, if an error is actually made, it is easy and fast to fix the problem and redeploy the new infrastructure. It is also fast to find an error as the infrastructure can be monitored very easily, as no changes are made during deployment. Infrastructure as Code increases the speed of the setup, modification and destruction of infrastructures [1]. Another aspect of using Infrastructure as Code, which is relevant for risk management, is that often, the infrastructure is made to be immutable. This means that the running infrastructure cannot be modified, but needs to be completely replaced by a new one instead, in the next version [5]. This prevents modifications and quick fixes being made by different people without being documented, which is a very risky way of working. Versioning is also made easier by Infrastructure as Code, which adds to the ease of management of the infrastructures. [3]

Infrastructure has traditionally been provisioned declaratively, in configuration files for example, which has shown to have issues with portability and reusability. Modern tools such as pulumi or AWS CDK are imperative, which means that they are written as code that can be run, in JavaScript for example. The biggest advantage of declarative programming is that the developer can describe more exactly how the system should look like, and it is easier to tailor the needs of the company [6]. The advantage of imperative programming is that it is less time-consuming and the human error is decreased even more. This is because it improves the readability and portability of the infrastructures. It en-

ables developers to understand the provisioning process better and thus reduces the dependency between IT and developers teams.

Issues

As explained above, Infrastructure as Code facilitates the processes that are necessary for working seamlessly between different teams, and helps decrease costs and risks. However, there are many issues linked with the use of Infrastructure as Code. There are several ways to implement and use Infrastructure as Code, which increases the difficulty of using it in an optimal way. There is a lack of standard practices among the many tools that are available, and they can be very complex [7]. Practitioners in the industry are faced with a plethora of equally popular tools that handle provisioning and configuration of infrastructure. Some common tools are Chef, Puppet, Ansible, Terraform, Cloudformation and Vagrant. They are not mutually exclusive, and cover in some cases very different aspects of the provisioning and configuration process. In most organizations, it is common to use many of these. The implementation is always on a best-effort basis.

The complexity and the lack of best practices for using these tools makes autonomous processes difficult to implement and manage. Several studies have identified code smells, flaws in code that may introduce problems, particular to Infrastructure as Code, both affecting security [8, 9] and overall code quality [8] that may have difficult long-term effects. Practitioners who use Puppet and Chef share many code smells, some are more common in the other and vice versa [8], it is likely that this applies to more tools as well. This highlights how important it is to have good practices and ensuring code quality while working with Infrastructure as Code. There is more research on tools themselves and on the adoption of infrastructure as Code rather than on the defects and security flaws bad code quality causes [10].

Other issues concern the infrastructure's dependencies with hardware, which makes the environment very complex [7]. These dependencies make the process error prone, and these errors need to be monitored and managed. For example, when only one task fails or one resource fails to be deployed, the entire deployment needs to be rolled back.

Because the infrastructure is dependent on hardware, and a lot of existing resources, it leads to issues with testability and verifiability. Even when using Infrastructure as Code, it is difficult to test that infrastructure behaves correctly, or that applications work in a certain infrastructure setting. Modern tools aim to improve on these weaknesses [1]. Especially, they allow for more testing on many levels, such as on the resources they provision and the applications they configure. There are still limitations to what is possible; it is difficult to test that a certain resource is connected to only one other resource, or only allows certain access, usually, one makes assertions on the configuration that is being generated and on the state after deployment. In a perfect world, we would want to test every aspect of our deployment, but that may not be possible nor feasible

to do. Furthermore, there are also long feedback loops for developers because provisioning infrastructure is not an instant action, even when automated.

Infrastructure as Code is a powerful tool in any organization. Just reimplementing the old infrastructure, if that infrastructure was flawed, could just be a waste of time. When implemented poorly, it holds little to none improvement over traditional IT organizations where the DevOps way of thinking was non-existent.

Conclusion

In conclusion, Infrastructure as Code can greatly reduce costs by helping teams be more efficient, and reduces risks which can be created by human error. It allows teams to have more control of the entire infrastructure and the workflow that it supports. However, the technology is hard to get right. Successfully implementing Infrastructure as Code is more related to people and processes than to code. Issues related to the implementation of Infrastructure as Code are linked to the complexity and number of existing tools and the dependencies of the technologies on hardware, which makes the systems error-prone and hard to test.

Infrastructure as Code is a powerful tool which allows organizations to shift their processes to be more automated. The infrastructure architecture reflects and defines a large part of the organization. This tool allows for better change management and can make a large impact on the organization. The organization can change its processes in a way to reduce dependencies, empower their teams and improve productivity with less effort. A perfect solution is most likely unattainable, but is worth striving for. Perhaps, as more standards are developed, and the tools mature, the path to perfect automation may be more clear.

References

[1] K. Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016.

[2] IBM Cloud Education. Devops as code, 2019.

[3] Mohamed Basher. Devops: An explorative case study on the challenges and opportunities in implementing infrastructure as code, 2019.

[4] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Software*, 33(3):94–100, May 2016.

[5] Adrian Hornsby. Immutable Infrastructure, Medium, March 2020.

[6] IBM Cloud Education. Infrastructure as code, 2019.

[7] Michele Guerriero, Martin Garriga, Damian A Tamburri, and Fabio Palomba. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 580–589. IEEE, 2019.

[8] J. Schwarz, A. Steffens, and H. Lichter. Code smells in infrastructure as code. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 220–228, 2018.

[9] A. Rahman, C. Parnin, and L. Williams. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175, 2019.

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