History of the IAC

We live in an era where everything is being automatized. Machines are useful because they are faster than humans and they do not get exhausted. We live in a word where things develop quickly, and people have to work fast. But things were not always like this.

Years ago, you could purchase a computer and for the time it was delivered and installed, weeks passed. For this reason, configuring it was not necessary to be done in a short period of time. But then a revolution came: virtualization. VMs could be spined out in minutes, and they required a configuration. The solution? Infrastructure automation again, more specifically using cloning and server image templates. But there is a new problem: now we have a huge number of servers whose initial configuration can be automatized but we have to maintain them (having them up to date and avoiding Configuration Drift).

So, we have to manage our configuration of the infrastructures better; due to this problem a new category of infrastructure automation emerged. CFengine, Puppet, and Chef established a new category of infrastructure automation tool: IAC.

Infrastructure as code or as software was mainly used at first by organizations that took and take full advantage of IaaS (infrastructure as a service). Their servers are not in their offices, so they were interested in researching new ways for configuring infrastructure. They needed to manage their infrastructure as better as possible.

What is IAC?

First of all, we should understand some concepts:

- Turing Complete: a system capable of perform logic (if, else, else if...) and capable to iterate and expand memory (loops, recursion, linking...).
- Config: files which contain values for different parameters. They are not Turing Complete. Static text.
- Code: every Turing Complete human readable group of instructions. It is static.
- Software: it is the same as code but instead of static is dynamic and also everything that code does, can be done by software and even more.

To be dynamic means that its state can change along time (even at runtime). Software contains code but is even more than that: tests, features, syntax, documentation...

Nowadays, plenty of organizations are using config files to manage infrastructures. But you have to ask yourself a question: if you have to configure a system (for example a server) and you are in a Turing Complete use case (you need to use logic, for instance) what would you use? A config file, code or software?

Well, the thing is that config files are not Turing Complete, so it is not possible. Code is a good solution, but software goes further. Remember that software does everything that code does and more. So that is the ideal solution.

That is exactly IAC or IAS, define the configurations where our applications will run on (infrastructure) with software.

Common uses of IAC

The use of IAC is greatly extended among many fields in the industry. The scale of its use also varies a lot, from big server farms to medium size companies. Some of the common uses that IAC are:

- Unify the configuration of decentralized companies. Many companies nowadays have many headquarters in different countries. This helps them bringing different points of views in the development of their projects and simplifies logistics. However, it does bring compatibility problems in their networks. Let's say a EEUU based company decides to open an office in Spain. If the pcs of the offices are configured manually, there is a big chance that relevant parameters, such as the text encoding or default date formats are set up differently. This can lead to many issues when interchanging relevant information. But if the company configures their infrastructure with IAC, all these configurations can be unified, reducing the risk of problems appearing.
- Automatise the deployment at server. Servers need to provide service consistently. These means that the downtime of machines should be reduced as much as possible. However, these machines sometimes break down and need to be swapped or redeployed. If this deployment (with its consequent configuration) is done manually every time, the procedure becomes time consuming and error prone. However, this configuration can be automatised so that, whenever a machine is swapped, the network detects it is a new system and runs the configuration automatically.
- Test applications in early development. When developing software, deploying the project to production environment is a critical step. If the project has grown big, doing this step at the end of development can cause several problems related to dependencies and compatibilities. However, doing this deployment many times during development (continuous integration) takes a lot of time and effort. Then IaC solves both problems: we can deploy the project automatically (with containers, for example) to a production-like environment, which will unveil the problems in a much earlier stage of development without taking away the time that manually deployment requires. IaC is not the same as continuous integration, but a very helpful way of doing it.
- Security in relation to IAC. IaC is a very powerful tool for many scenarios. However, the fact that the configuration you specify is applied to many systems means that a vulnerability in the template can cause serious security issues in many machines. In 2020, a study published by Unit 42 identified around 200,000 vulnerabilities in IaC templates. Here's when a new term appears: Compliance as Code. After a "pure "IaC template is run, we can execute many other configurations (essentially, anything that can be executed in a command line). This means that we can use protocols such as SCAP to harden the systems deployed with IaC, solving the problem and even improving the results we would have with a traditional approach. In fact, many consider Compliance as Code a necessary part in Infrastructure as Code and not a different technique.

IaC as students

Since the scale of the common uses of IaC is considerably large, it may seem out of range for most of us to have any kind of practices with the most popular technologies. However, there´s many options, some of them we already have worked with.

Vagrant: Vagrant is a tool used to automate building and management of virtual machines. In this degree, Vagrant is used in SSI to configure the virtual machine with which the laboratories are done.

Docker: Docker is a tool that transforms IaC into a core component in the development process. With Docker, developers can write code that specifies environments and configurations, in which they can deploy the application at any moment in development to check if there's any problem related to dependencies or incompatibilities. In this subject (ASW), docker is recommended to apply continuous integration for the laboratory assignment. There´s a seminar in the subject ASR in which docker is explained in depth. Also, in SSI, the VM created with Vagrant has several docker images that configure environments prepared to work with the lessons.

Emulate servers with VM: the use of virtual machines is not uncommon in many subjects of this degree. Using technologies such as the ones commented above, we can automate the configuration of VM with Vagrant and use docker to configure the environment, and if we put it all together with several small virtual machines, we can see how IaC saves a lot of time and effort both for the developers and for the server configurations.

Why should we use IAC?

As everything in computer science, Infrastructure as Code brings us both advantages and disadvantages of its usage. Some of the best benefits are:

- **Simplicity**. You can re-deploy your infrastructure with just one click. This in comparison with the work you would have to do without the automatization, which would take a long time to meticulously redeploy and configure the entire environment, is a very easy task.
- **Shareability**. You can share your whole infrastructure with other teams or people by just sending them the code. This code could also be customizable (for example, using configuration files) allowing you to create templates.
- **Stability**. Every time we run the code, we get the same exact environment. This implies the same number of hosts, networks… and even their names are identical.
- **Scalability**. Your code can be done in a way that allows you to modify the configuration file in a way that the environment grows (As the code may contain loops).

This process as I said before, may also bring us some drawbacks, some of them are:

- **Learnability**. As any other code-based system, learning to write a script might be very difficult to learn for unexperienced users.
- **Maintainability**. As the infrastructure grows on size our code grows on lines. As the software version develops, we will need to update more and more features, which may result in technical debt.
- **Error Handling**. As is common in programming, bugs are an everyday thing, and our code is not free of it. We might find the program failing in the middle of the execution and it may not be easy to restart from the exact same point, and re-executing from scratch may take a long time.
- **Understandability**. If you are using someone else's code, it might take a lot of time for you to understand it, and it could be difficult.

When should we use IAC?

We can use IaC whenever we have to manage any type of infrastructure.

Bibliography:

- https://www.redhat.com/sysadmin/pros-and-cons-infrastructure-code