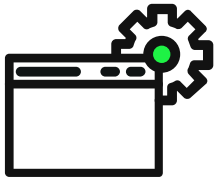
Whitepaper

Dude, just define your hard drive!



Why Infrastructure-as-Code builds better software

Published in July, 2022

1 Introduction

Building software solutions for sustainable futures is what Sobolt lives and breathes. However, without hardware to run our software on, code achieves nothing. The emergence of cloud computing has led to new possibilities to integrate managing hardware - infrastructure - into the developmental process. For Sobolt this means its developers build more reliable solutions, allowing us to make a bigger impact.

Many challenges in infrastructure management stem from hardware being a completely different beast than software. In software, you can just define a new function and quickly test whether it does what you expect it to do. With hardware, that's much harder, requiring much more manual experimentation or writing ad-hoc scripts. That's not even taking into account turning a successful system configuration into a repeatable, maintainable, testable, reliable, documented process that can also be iterated upon in the future. But what if we could just define our hard drive - or indeed our whole system configuration?

This is where Infrastructure-as-Code (IaC) comes in. Software packages such as Pulumi and Terraform have created programmable interfaces for Cloud providers' APIs. These allow developers to declaratively define their infrastructure resources and their relations as code. For example, a Pulumi script for a simple server with an extra hard drive looks like this:

```
import pulumi
import pulumi_gcp as gcp


# Support different DTAP environments
env: str = pulumi.get_stack()  # Different values enable different deployments

# Define a virtual machine with an extra 512GB drive
data_disk = default = gcp.compute.Disk(f"example-data-{env}", size=512)
server = gcp.compute.Instance(
        f"example-server-{env}",
        machine_type="e2-small",
        tags=["http-server", "https-server"],
        boot_disk=dict(
            initialize_params=dict(image="debian-cloud/debian-11", size=100),
        ),
        attached_disks=[dict(source=data_disk.self_link)],  # Attach disk
)
```

As this is just an illustration, it creates a simple setup of a single VM server with an extra attached 512GB. Yet it can be expanded with almost anything the cloud providers themselves support. For example Sobolt used IaC to define scalable instance groups steered by autoscalers, fronted by a proxy and load balancer, with multiple health checks and optimized for GPU inference. Setups can be as complicated as needed.

2 Benefits

Many of IaC's advantages are those of typical software. For example, IaC is repeatable. Because it is code, we reliably deploy the exact same setup as often as we want, from scratch. Creating Develop-Test-Acceptance-Production (DTAP) streets becomes trivial.

Furthermore, now that the infrastructure is just a text file, we can version control it and make it part of software releases. This allows the infrastructure to evolve with the solution. With little fuss, solution version 2 in testing can be different from version 1 in production. Iterating on infrastructure is now easy and does not affect your solution in production.

This in turn leads to near fearless updates of the production environment. This is due to the updated IaC-code being able to go through the DTA environments. We also know that this version of the solution belongs on this version of the infrastructure, because it is part of the same release.

Now that the infrastructure is 'just' software, it can be maintained and tested together via Continuous Integration (CI). As part of the CI cycle, the code can create a temporary CI infrastructure stack to which the solution is deployed. This way, integration tests are performed with real, live systems that are exactly like the environment we will deploy to in production instead of whatever system the CI pipeline runs on.

Expanding on this, the infrastructure can be automatically upgraded using our Continuous Deployment (CD) pipelines. This reduces the amount of error-prone manual maintenance required. Maybe more importantly, however, it also enforces that the upgrade process indeed goes through the DTAP street as described above. Together they mean our upgrade process is tested, more predictable and reliable.

Another important advantage is that the infrastructure is now documented. Even if there are no further comments or explanations, the structure is right there. The code is now also part of the code review process, allowing colleagues to give feedback. Six months in the future, the developer themselves can go back and look at how they designed and implemented a complicated piece again. Two years into the future when the original team has left, new developers only need to read this to understand what is currently running in production. Detailed knowledge of the setup has left the developers' brain and entered the repository.

Lastly, IaC arguably lowers the barrier to entry for inexperienced developers. They can now use the tools they know - programming - to learn about infrastructure and implement improvements under the guidance of more seasoned colleagues (now possible via code review!). Since development can happen in a personal environment, a developer can safely experiment. This way, IaC serves as a vehicle for knowledge transfer.

3 Challenges

Of course, IaC is not only all moonlight & roses, but also introduces challenges. For one, it requires designing cloud-native applications. This might not always be possible or even desirable. Relatedly, current solutions are not cloud provider neutral, so adoption furthers dependency on a single choice. The ease with which IaC allows rebuilding infrastructure in another provider's ecosystem does also introduce more flexibility, but the risk of vendor lock-in is still present.

Secondly, the initial hurdle to get started is not insignificant. Because it is a complex topic, the IaC solutions' easy tutorials and extensive API documentation are often not enough. To design good infrastructure, it is still necessary to truly understand the options the cloud provider offers. Similarly, in our experience implementation often still requires much cross-referencing cloud provider APIs and documentation with those of the IaC solutions.

4 Conclusion

Regardless of these challenges, Sobolt is convinced of Infrastructure-as-Code's benefits. The advantages of version control, automatic documentation, easy developmental iteration, predictable DTAP, reliable (re)deployment and maintenance all bring so much that whenever possible we set up new solutions with IaC. With it, our development process is more reliable, allowing us to build better software for a better future.

sobolt
sustainable futures

Published in July, 2022

www.sobolt.com