






A Short History of Infrastructure Management

In this lesson, we will peek into the past and go through a short history of infrastructure management.

WE'LL COVER THE FOLLOWING

- The Beginning
-  Configuration Management
 -  Configuration Management Tools
 -  Pros
 -  Cons
- Virtual Machines
 - Mutability vs. Immutability
-  The Cloud Hosting
- Modern Infrastructure

The Beginning

A long time ago in a galaxy far, far away...

We would order servers and wait for months until they arrive. To make our misery worse, even after they come, we'd wait for weeks, sometimes even months, until they are placed in racks and provisioned.

At that time, only a select few people could access these servers. If someone does something that should not be done, we could face an extended downtime. On top of that, nobody knew what was running on those servers.

Manual provisioning and installations were a nightmare because even after putting a lot of effort into documentation, given enough time, the state of the servers would always diverge from the documentation. **Sysadmins** were the key people without whom no one can handle these servers.



Configuration Management

Configuration Management

To manage the configuration means to track and control changes in the software. Configuration management tools enable us to determine what was changed, who changed it and much more.

Configuration Management Tools

Then came configuration management tools. We got **CFEngine**.

Pros

It was based on promise theory and was capable of putting a server into the desired state no matter what its actual state was.

It allowed us to specify the state of static infrastructure and have a reasonable guarantee that it will be achieved.

Another big advantage it provided is the ability to have, more or less, the same setup for different environments. Servers dedicated to testing could be (almost) the same as those assigned to production.

Cons

Unfortunately, usage of CFEngine and similar tools were not yet widespread. We had to wait for virtual machines before automated configuration management became a norm. However, CFEngine was not designed for virtual machines. They were meant to work with static, bare metal servers. Still, CFEngine was a massive contribution to the industry even though it failed to get widespread adoption.

After CFEngine came Chef, Puppet, Ansible, Salt, and other similar tools. We'll go back to these tools soon. For now, let's turn to the next evolutionary improvement.

Besides forcing us to be patient, physical servers were a massive waste in resource utilization. They came in predefined sizes and, since waiting time was considerable, we often opted for big ones. The bigger, the better. That meant that an application or a service usually required less CPU and memory than the server offered. Unless you do not care about costs, that meant that we'd deploy multiple applications to a single server. The result was a dependencies nightmare. We had to choose between freedom and

dependencies nightmare. We had to choose between freedom and standardization.

Freedom meant that different applications could use different runtime dependencies while standardization involves systems architects deciding the only right way to develop and deploy something.

Virtual Machines

Then came Virtual machines and broke everyone's happiness.

Virtual machines (VMs) were a massive improvement over bare metal infrastructure.

- They allowed us to be more precise with hardware requirements.
- They could be created and destroyed quickly.
- They could differ i.e. a single physical server could have multiple VMs running in isolation. One VM could host a Java application, and the other could be dedicated to Ruby on Rails.
- We could get them in a matter of minutes, instead of waiting for months. Still, it took quite a while until “could” became “can”.

Even though the advantages brought by VMs were numerous, years passed until they were widely adopted. Even then, the adoption was usually wrong. Companies often moved the same practices used with bare metal servers into virtual machines. We could have identical servers in different environments. Companies started copying VMs. While that was much better than before, it did not solve the problem of missing documentation and the ability to create VMs from scratch. Still, multiple identical environments are better than one, even if that meant that we don't know what's inside.

Mutability vs. Immutability

The configuration management tools helped spread the adoption of “infrastructure as code” principles. But the problem was they were designed with static infrastructure in mind. On the other hand, VMs opened the doors to dynamic infrastructure where VMs are continuously created and destroyed. Mutability and constant creation and destruction were clashing. Mutable infrastructure is well suited for static infrastructure. It does not

respond well to challenges brought with dynamic nature of modern data

centers. Mutability (changeable at runtime) had to give way to immutability (nothing can be tweaked at runtime).

When ideas behind immutable infrastructure started getting traction, people began combining them with the concepts behind configuration management. However, tools available at that time were not fit for the job. They (Chef, Puppet, Ansible, and the like) were designed with the idea that servers are brought into the desired state at runtime. Immutable processes, on the other hand, assume that (almost) nothing is changeable at runtime. Artifacts were supposed to be created as immutable images. In case of infrastructure, that meant that VMs are created from images, and not changed at runtime. If an upgrade is needed, a new image should be created followed with a replacement of old VMs with new ones based on the new image. Such processes brought speed and reliability. With proper tests in place, immutable is always more reliable than mutable.

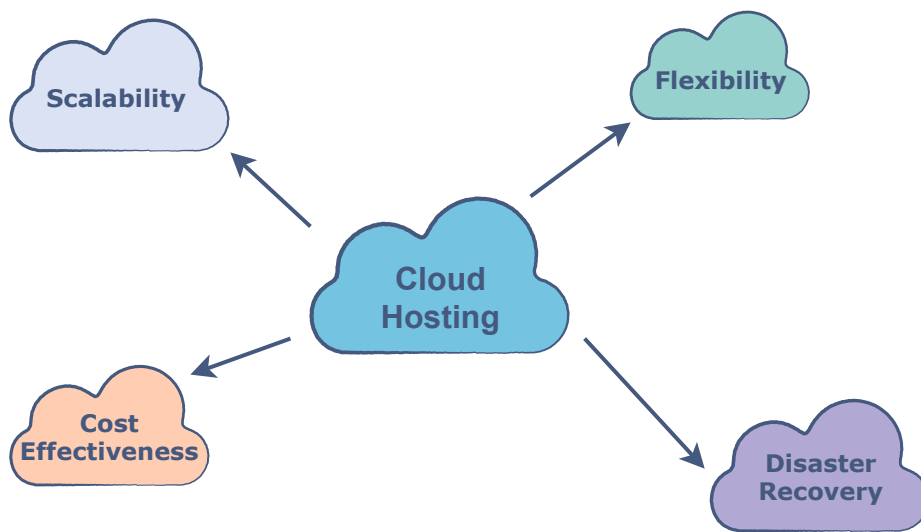
Subsequently, we got tools capable of building VM images. Today, they are ruled by Packer. Configuration management tools quickly jumped on board, and their vendors told us that they work equally well for configuring images as servers at runtime. However, that was not the case due to the logic behind those tools. They are designed to put a server that is in an unknown state into the desired state. They assume that we are not sure what the current state is. VM images, on the other hand, are always based on an image with a known state. If for example, we choose Ubuntu as a base image, we know what's inside it.

Adding additional packages and configurations is easy. There is no need for things like “if this then that, otherwise something else.” A simple shell script is as good as any configuration management tool when the current state is known. Creating a VM image is reasonably straightforward with Packer alone. Still, not all was lost for configuration management tools. We could still use them to orchestrate the creation of VMs based on images and, potentially, do some runtime configuration that couldn't be baked in. Right?

The Cloud Hosting

The way we orchestrate infrastructure had to change as well. A higher level of

dynamism and elasticity was required. That became especially evident with the emergence of cloud hosting providers like Amazon Web Services (AWS) and, later on, Azure and GCE.



They showed us what can be done. While some companies embraced the cloud, others went into defensive positions. “We can build an internal cloud”, “AWS is too expensive”, “I would, but I can’t because of legislation”, and “our market is different”, are only a few ill-conceived excuses often given by people who are desperately trying to maintain status quo. That is not to say that there is no truth in those statements but that, more often than not, they are used as an excuse, not for real reasons.

Still, the cloud did manage to become the way to do things, and companies moved their infrastructure to one of the providers. Or, at least, started thinking about it. The number of companies that are abandoning on-premise infrastructure is continuously increasing, and we can safely predict that the trend will continue.

Still, the question remains. How do we manage infrastructure in the cloud with all the benefits it gives us? How do we handle its highly dynamic nature? The answer came in the form of vendor-specific tools like CloudFormation or agnostic solutions like Terraform. When combined with tools that allow us to create images, they represent a new generation of configuration management. We are talking about full automation backed by immutability.

Modern Infrastructure

We're living in an era without the need to SSH into servers.

Today, modern infrastructure is created from immutable images. Any upgrade is performed by building new images and performing rolling updates that will replace VMs one by one. Infrastructure dependencies are never changed at runtime. Tools like Packer, Terraform, CloudFormation, and the like are the answer to today's problems.

One of the inherent benefits behind immutability is a clear division between infrastructure and deployments. Until not long ago, the two meshed together into an inseparable process. With infrastructure becoming a service, deployment processes can be clearly separated, thus allowing different teams, individuals, and expertise to take control.



We'll need to go back in time one more time and discuss the history of deployments. Did they change as much as infrastructure?