INTRODUCTION TO IMMUTABLE INFRASTRUCTURE

Immutable infrastructure has gained immense popularity in the last few years, particularly in the cloud native realm—and deservedly so. Immutable infrastructure is dramatically changing the way software is shipped, built, and managed, speeding up development releases. Let's take a look at immutable infrastructure, including how it compares to mutable (traditional) infrastructure, and how its benefits make it less a trend and more a new standard for software deployment.

Mutable infrastructure: an overview

In order to understand immutable infrastructure, let's step back and explore mutable infrastructures. This is, until recently, how ops teams handled infrastructure.

"Mutable" is defined as anything that's capable of being changed. So, mutual infrastructure is capable of being changed when needed. Traditional, or mutable, servers are servers that are continually updated and modified in place. Systems engineers and admins spent time secure shelling (SSH) into servers, upgrading/downgrading packages manually, and tweaking configuration files as needed, server by server. These servers required dozens of logins and accounts, could be in any state of repair or disrepair, and managed software with updates that might succeed—or fail.

With my systems engineering background, I remember the days of installing a server: I'd order a Dell server, wait a couple weeks for arrival, provision the server over a few days, then take it to the data center to rack. This process took a lot of time—and for only a single server. Now, imagine doing this for multiple servers! Of course, the server work is not done: you must patch weekly to ensure that

servers are updated during production.

There are a lot of problems with this approach. These ongoing changes mean it's nearly impossible to have identical servers in a cluster-like environment, particularly after deployment. Further, there are no fast and effective ways to replace an existing server with something identical in case of issues.

To solve the problems of mutable infrastructure, infrastructure-as-code tools like Chef were created to get servers up-to-date quickly. Still, drifts in configuration could create a debugging nightmare when things go wrong in production.

What is immutable infrastructure?

Recently, the emergence of cloud services has given rise to immutable infrastructure. Immutable infrastructure refers to servers (or VMs) that are never modified after deployment. With an immutable infrastructure paradigm, servers work differently. We no longer want to update in-place servers. Instead, we want to ensure that a deployed server will remain intact, with no changes made.

When you do need to update your server, you'll replace it with a new version. For any updates, fixes, or modifications, you'll:

- Build a new server from a common image, with appropriate changes, packages, and services included
- Provision the new server to replace the old one
- Validate the server
- Decommission the old server

Every update (environment) is exact, versioned, timestamped, and redeployed. The previous servers are still available if you need to roll back your environment. This change almost entirely removes troubleshooting for broken instances, and these new servers are quick to deploy thanks to OS-level virtualization.

So, immutable infrastructure can speed up deployment, but is the actual performance enhanced? Generally, yes. Performance on an individual instance can occasionally be slower than a mutable server, but it's rare. Importantly, horizontal scaling is significantly easier, supporting quicker environment building with both more machines and speedier deployments.

Benefits of immutable infrastructure

Perhaps the biggest benefit of immutable infrastructure is how quickly engineers can replace a problematic server, keeping the application running with minimal effort. But that's just the beginning—immutable infrastructure offers several benefits:

- Infrastructure is consistent and reliable, which makes testing more straightforward.
- Deployment is simpler and more predictable.
- Each deployment is versioned and automated, so environment rollback is a breeze.
- Errors, configuration drifts, and snowflake servers are mitigated or eliminated entirely.
- Deployment remains consistent across all environments (dev, test, and prod).
- Auto-scaling is effortless thanks to cloud services.

The ability to improve reliability, efficiency, and consistency in a deployed environment, and to

recreate it within minutes, is invaluable. Gone are the days of mutable servers increasing cost and iteration time, severely delaying your time-to-market. Immutable infrastructure promotes agile development.

Of course, immutable infrastructure has drawbacks, but they are few: There are no in-place updates to servers, which may occasionally be problematic, depending on your environment. Secondly (perhaps lastly), a fairly steep learning curve for new tools can make the initial adoption tricky to maneuver—but that applies to any new technology.

Immutable infrastructure best practices

To take every advantage of immutable infrastructure, you'll want to use it efficiently with tools and processes like:

- Automating deployment comprehensively (improves predictability)
- Provisioning quickly in cloud computing environments
- Developing go-to solutions for handling ephemeral or stateful data

Popular immutable infrastructure tools

Below are some of the common tools and service that you can leverage to accomplish immutable infrastructure, but this list is certainly not exhaustive:

- Docker
- Kubernetes
- Spinnaker
- AWS, GCP

With these dramatic changes that affect the entire software development cycle, from devs to ops and, finally, end users, immutable infrastructure is the future. If you're not working this way yet, you probably should be.

Additional resources

Immutable Infrastructure & Rethinking Configuration - Interop 2019 from **RackN**