Nightmare Infrastructures: keep the zombies away from your Cloud

28 October 2022 - 9 min. read

AWS Well-Architected | Cloud adoption | DevOps

"Life's no fun without a good scare." - The nightmare before Christmas.

In the Nightmare Before Christmas, Jack Skeleton also cited, "What's this, what's this?"

It sometimes happens to us when dealing with some infrastructure designs.

You may already have caught the spirit of this article: we'll describe some strange infrastructure designs and practices we have encountered, telling stories about Cloud anti-patterns that will become an absolute nightmare in the long period.

We don't want to point the finger at any particular design: requirements or contingencies lead and force the adoption of anti-patterns; we hope you can have a good laugh with this article; the real world out there is scary enough!

Disclaimer: If you are one of our customers reading and this story rings a bell: yes, we're talking about you! :D

Grab your favorite blanket and popcorn; we are about to start our horror journey!

Developmenttestproduction

The road to hell is paved with good intentions.

What can go wrong with a small fix in production because there's no time to test it in the development and test environment?

Then that fix remains only in production, making the development and test environment out of sync. Everyone forgets about that tiny little piece of code that waits for its revenge. When a new feature is released, a new bug could be found only in production because the patch interacts with the new feature (Murphy's law is a constant in these scenarios).

Since we are talking about the production environment, another minor fix is released, making it even more misaligned.

The story repeats until every new development is applied directly to production because there's no way to be sure that releases in test and development will work.

We finally have a new environment: **developmentestproduction**.

In other cases, there are test environments that aren't simply for testing; we always say: *"When the customer screams, you are in production."*

It started on a Tuesday morning when we had a drift in the test environment. After carefully checking how to solve it, we applied modifications on CloudFormation, and the scary "Update Rollback Failed" state was reached. Investigating the issue led us to the root cause of the problem: someone deleted a task definition of a Fargate ECS container that had been updated by CloudFormation. In this case, there's no other option: you have to delete the entire stack and deploy it from scratch.

No problem, we thought: it's only a test environment: we are using Infrastructure As Code, what can go wrong?

5 minutes after deleting the stack, Slack on our Macs (and PCs) started ringing altogether: the "Test" environment was used to host services for customer partners. Lucky for us, data was backed up, and by "the magic" of using CloudFormation, we were able to get the entire infrastructure running from scratch in 15 minutes.

What have we learned? Don't trust when something has "test" in its name.

By the way: if an environment is essential for the business, name it "production", "prod", or something that screams: "Please take care of me, handle with care".

The Cloud makes it easy to deploy infrastructures; use this quality to your advantage! Since everything can be automated using Infrastructure as Code and you pay as you go, costs are not an excuse: you can experiment and destroy everything once you're done! Pipelines for Continuous Integration and Continuous Deployment will also help you keep everything synchronized.

AWS CloudFormation, CodeBuild, CodePipeline, and CodeDeploy are key services to ensure that everything stays up to date between environments and ease the effort of

experimenting with new solutions.

Giant Microservices

It seemed to be a sort of monster, or symbol representing a monster, of a form which only a diseased fancy could conceive. If I say that my somewhat extravagant imagination yielded simultaneous pictures of an octopus, a dragon, and a human caricature, I shall not be unfaithful to the spirit of the thing. H.P. Lovecraft - The Call of Cthulhu

A microservice is, by definition, a small piece of software running in cooperation with others, usually to let different teams manage services by focusing on a single problem.

For example, a microservice can send emails to customers, while another can take ownership of updating the inventory when a successful purchase is made. Microservices are designed to be maintainable and have a small footprint (in terms of memory and CPU utilization).

We found a Docker container with a Liferay application that consumed 8 gigabytes of memory and 4 CPUs that acted as CRM, a payment gateway, and sent emails. This was not clearly a microservice but an elephant in a small car, trying its best to survive!

Enterprise-ish frameworks tend to be monolithic in nature. If you need such technology, simply embrace it and don't be tempted to containerize it only for the sake of a "better deployment strategy". Maybe EC2 instances and autoscaling groups aren't as fancy as Docker containers and lambdas, but they will work and help you.

As Lieutenant Columbo would say: "Just one more thing": no, spinning up an EKS cluster or, for worse, a Kubernetes custom deployment on EC2 doesn't mean you're using microservices. You are "simply" adding complexity and maintenance costs to your existing infrastructure.

If you develop something from scratch, making it "cloud native" will be easier. Simply porting an application into AWS with a lift and shift without adapting it for the Cloud paradigm will not make your software cloud-ready.

When you split every functionality and build small parts. AWS Cloud Map can help with service discovery.

Static web contents served by EC2 Instances

"*Do I look abominable to you? Why can't they call me the Adorable Snowman, or ... or the Agreeable Snowman for crying out loud?*" - Yeti, Monsters Inc

Let's face it: sometimes, we are tempted to use something we already master, even if there are managed services. One typical example is an Apache webserver to host a simple static website.

This approach increases maintenance efforts and even the AWS monthly bill, even if you don't want to achieve essential high availability using a load balancer and an AutoScaling Group.

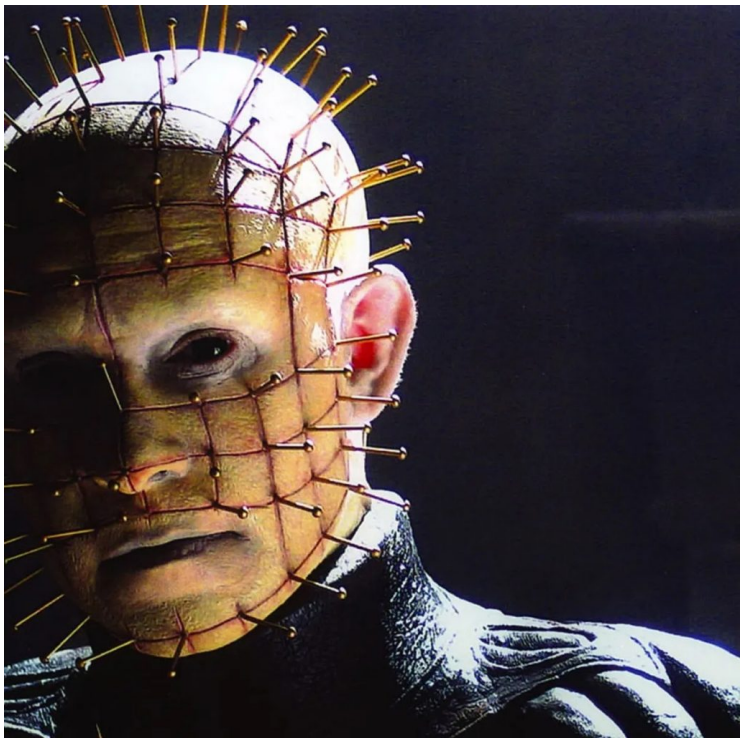
We saw two Apache webservers to serve static content that, for SEO and historical reasons, had 60.000 rewrite rules combined with 13.000 conditions. Most of the rewrite rules were redundant and interfered with others: depending on the URI path, a request could take 5 seconds to be served. When something needed investigation, enabling trace logging generated an average of 2.000 lines of logs.

EC2 instances where m5.2xlarge, a huge cost saving could be achieved using CloudFront and S3, with better performance and leveraging the power of a global Content Distribution Network.

In this scenario, a lambda@edge with a Dynamo table perfectly fits the use case!

Static content generated in a backend (with an unexpected plot twist)

No tears, please. It's a waste of good suffering - Pinhead, Hellraiser.



When a backend generates static content, we waste CPU, money, and energy. With a monolith application, you can try to at least use CloudFront to lower the load on the application server.

And please, please, don't put any script utility (or application routes) in subfolders that can accidentally be indexed by search engines (excluding them using robots.txt is not a solution).

A customer phoned us in panic mode because his database had been dropped during the night, and no employee logged in.

After some investigation, we found that there was a PHP script (utility.php) that was used by developers to quickly drop the database to start over with a clean environment. It was accessible by a simple GET operation from the browser to speed up everything, and the directory index was enabled for their convenience.

Since the application deployment was made by hand by copying files into the application server, someone forgot to exclude the utility directory; a search engine found it and started indexing, so it issued a GET to the utility.php, and... the production database was forever gone! Igor in Frankenstein Junior would say: "*Could be worse, could be raining.*"

Unmanaged Services - The stateful stateless

"*You pay for what you get, you own what you pay for... and sooner or later, whatever you own comes back home to you.*"

— Stephen King, It

A MySql deployment on Docker seems an excellent idea to manage updates and keep versions aligned between the local docker-compose on development and production environments.

So we'll spawn a container on an EC2 instance to ease the deployment process!

A container is stateless by definition, so, to keep data, you'll define volumes. With data, you need to worry about backups, so you'll deal with scripts and cron jobs to make it consistent (a simple ec2 snapshot doesn't guarantee consistency on transactions).

Then you'll learn the art of keeping up the service: container limits, version upgrades of tables and databases made by hand using "docker exec," disk resizing...

Ultimately, most of your time will be spent on keeping the service running without high availability or resilience.

If you calculate the Total Cost of Ownership (TCO), you'll see that a Multi-AZ Aurora RDS cluster is way more convenient than this solution!

For more information on stateful and stateless, you can read Alessio's excellent article: https://www.proud2becloud.com/stateful-vs-stateless-the-good-the-bad-and-the-ugly/

The stateful stateless - pt. 2

"Errare humanum est, perseverare autem diabolicum" - st Augustine *(to err is human, to persist in that same error is diabolical)*

We'll keep data on a shared filesystem! EFS is good, has native backup, it can scale, and you pay as you go!

In this case, we have seen a small PHP application hosted entirely on EFS, running on containers on ec2 instances in an autoscaling group.

We know that sometimes a shared filesystem is needed to store shared data.

Still, even a small web application on EFS will have poor performance because of how it interacts with the filesystem (and because IOPs will be lower than a small EBS volume).

You can follow this advice by AWS, but your experience may vary, depending on the application:

- Optimizing WordPress performance with Amazon EFS
- Why is my EFS file system performance slow?

Life is better when you keep application files inside the container, use ECS (in Fargate mode to avoid managing instances) and use S3 as your preferred storage solution.

The stateful stateless - the final chapter

through me the way into the suffering city,

through me the way to the eternal pain,

through me the way that runs among the lost.

Inferno, Canto III - Dante

Ok, we've listened to you, so we'll use ECS Fargate to deploy... a Cassandra cluster! Fargate is fully managed, so we don't have any problems; looks good!

A Fargate task has **ephemeral** storage. By default, its size is 20 Gb, expandable to 200, so you can lose more data if a container is restarted... Resorting to EFS is still not the right choice: performance may vary if you don't use the provisioned mode.

Additionally, there will be data transfer charges for replication because you'll want your instances to be spread across different Availability Zones. Consider that when you add a node, there will be an initial data replication. Inserting and removing data also generates traffic.

10 Gb/hour of replication traffic will lead to an additional bill of about 150$

Amazon KeySpaces will give you more performance, maintainability, and a lower bill!

... and thence we came forth to see again the stars - Inferno - Dante

This last episode concludes our night of terror through infrastructures ad implementations.

Have you felt a shiver in your spine while reading these stories? Do you have something more frightening to tell? Let us know in the comments!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!
