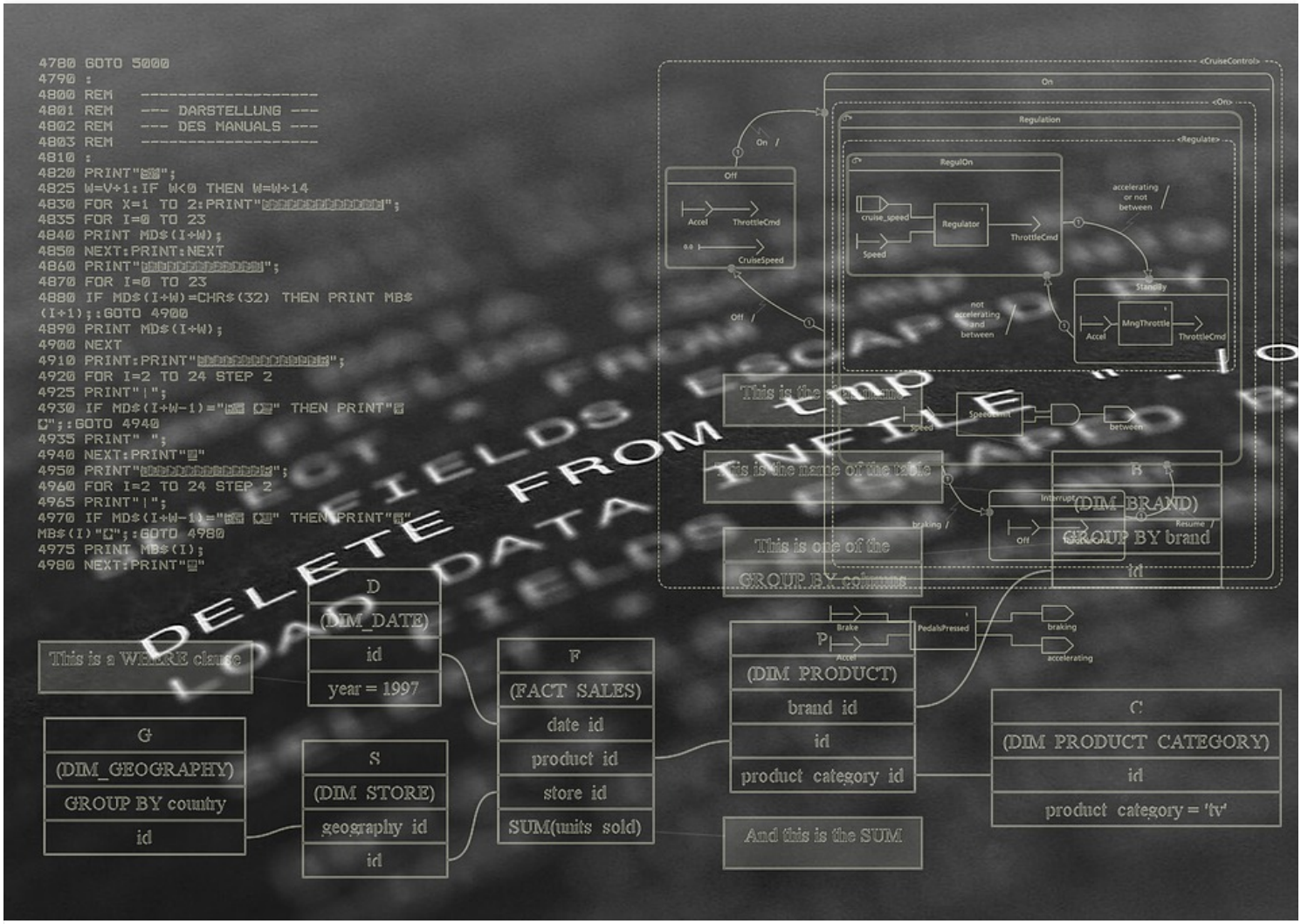


# UNDERSTANDING AWS DESIGN PRINCIPLES

Patrick McClory May 20, 2016 Amazon Web Services 13,129 Views

When utilizing AWS, enterprises need to plan for a distinct approach to handling infrastructure deployment patterns – one where the overall design favors availability and partition tolerance and accommodates for the reality of eventual consistency. This is a meaningful shift from traditional technology implementations in data centers where, over the last 20 years, consistency was key when balancing out considerations within a distributed computing system. When designing for AWS, consider Eric Brewer's CAP Theorem research. It is a critical piece of theoretical computing research that asserts any network shared system can have only two of three desirable properties:

- Consistency in having a single up-to-date copy of data
- High Availability of that data
- Tolerance to network partitions

Being honest, CAP Theorem isn't universally agreed upon. I reference it here as a great model with which to discuss distributed systems design on AWS. It's like the Newtonian physics model that is "good enough" for most scenarios versus The Theory of Relativity that goes into much more fine-grained detail. If interested in learning more about CAP Theorem, Mark Burgess has a phenomenal critique and criticism that spans two well-written and thought-out blog posts, albeit quite lengthy.

When applying the CAP Theorem to the real-world problems, it helps to consider each of the three properties individually, rather than trying to achieve all three at the same time. It really is a balancing act of these properties along a continuum.

With that having been said, when deploying software or systems that assume a specific balance of Brewer's CAP theorem (strong consistency, highly available, and less tolerant to partition failure) to an infrastructure platform that has a different set of objectives (eventual consistency, highly available and tolerant to partition failure), the mismatch in how availability is achieved can negatively impact the application's behavior. Werner Vogels, CTO and VP of Amazon, is famous for repeatedly articulating that 'everything fails, all the time' in infrastructure systems at scale. This statement is often paired with the recommendation that systems should be designed to tolerate failure from the infrastructure through to the software layer. The question we are left with in this scenario is a familiar one: how much effort (cost) are we willing to expend to achieve a specific balance of Consistency, Availability and Partition tolerance?

Designing highly available systems isn't anything new, but the approach we've taken for the past 20+ years has been focused on leveraging techniques and tactics within an infrastructure or hardware-focused world to build in this resiliency. In its most basic form, we mitigated that risk by purchasing hardware and support warranties with stringent SLA's so that in the event that hardware fails, you have an understanding with the vendor and a reasonable (and contractual) agreement as to when hardware availability will be restored. Add to this the time required to restore the services on top of that machine (depending on the type of failure), and you have your functional measure of your worst-case RTO (Recovery Time Objective) for a single machine. At its most basic level, this risk mitigation tactic is the simplest form of resiliency planning, though in today's world, the consensus would clearly be that this provides a path to recovery, but not necessarily a high degree of availability.

Fast-forwarding through the maturity curve of HA tooling in the infrastructure stack, tools like VMWare's vMotion become novel as they allow for the failure of hardware (which we know will happen eventually) and allows for applications generally deployed in single instances to achieve a high level of availability through this live migration process. These systems become very complex, and while they seem to provide

partition failure tolerance, it could also be interpreted that the true nature of this type of live-migration workflow is actually preventing the failure rather than being tolerant of it. In this case, when we begin to stretch the bounds of this partition tolerance dimension, this type of solution breaks, leaving us with a system that delivers consistency and availability but with less partition failure tolerance than would otherwise be desired.

So what do we do now? Amazon has a bias toward planning for and handling partition-level failure by embracing an eventual consistency model. On the whole, we have a lot of software, both commercial and custom, designed to essentially rely on these underlying 'failure prevention' tactics at the infrastructure level. This data point is necessary to understand as the technical-level tools and tactics used to enable these 'failure prevention' strategies are complete anti-patterns and/or are simply unavailable in the AWS shared infrastructure environment.

Because we now have to accept the risk that instances aren't as persistent or permanent as we once assumed, we must design for failure within the application management and configuration management layers of a system to ensure that there are well-exercised restore processes. A best practice is to design so we're either separating out the application and data layers to distribute the risk of failure, or we're persisting data externally and have automated the process of recovering instances upon failure. This isn't Amazon changing the game, but rather returning the responsibility for solving this problem to the logical application layer rather than handling it on an enterprise's behalf. This comes with enormous benefits in terms of potential cost structure changes as well as some unique and novel opportunities to leverage systems and services that Amazon, due to their unique size and scale, have had to develop for themselves.

This cost/benefit rebalancing is something that every customer we work with eventually contends with as they adopt AWS, though rationalizing it earlier rather than later certainly has its benefits. What most learn is that the true cost of the cloud isn't in the infrastructure but in the effort required to re-train and re-tool their teams to best benefit from their usage of the AWS platform.