Data Center Feng Shui: Reliability is not the Absence of Failure

Lori MacVittie, 2011-23-03

But rather it is the ability to compensate for it.

Redundancy. It's standard operating procedure for everyone who deals with technology – even consumers. Within IT we're a bit more stringent about how much redundancy we build into the data center.

Before commoditization and the advent of cheap computing (a.k.a. cloud computing) we worried about redundant power supplies and network connections. We leveraged fail-over as a means to ensure that when the inevitable happened, a second, minty-fresh server/application/switch was ready to take over without dropping so much as a single packet on the data center floor.

Notice I said "inevitable." That's important, because we know with near-absolute certainty that inevitably hardware and software fails. Interestingly, it is only hardware that comes with an MTBF (Mean Time Between Failures) rating. It is nearly as inevitable that software (applications) will also experience some sort of failure – whether due to error or overload or because of a dependency on some other piece of software that will, too, inevitably fail because of hardware or internal issues.

Failure happens. That doesn't mean, however, that an application or an architecture or a network is unreliable.

Reliability is not an absence of failure in the data center, it's a measure of how quickly such failures can be compensated for. Being able to rely upon an application does not mean it never fails, it simply means that such failures as do occur are corrected for or otherwise addressed quickly, before they have an impact on the availability of the application. And that means the entire application delivery chain.

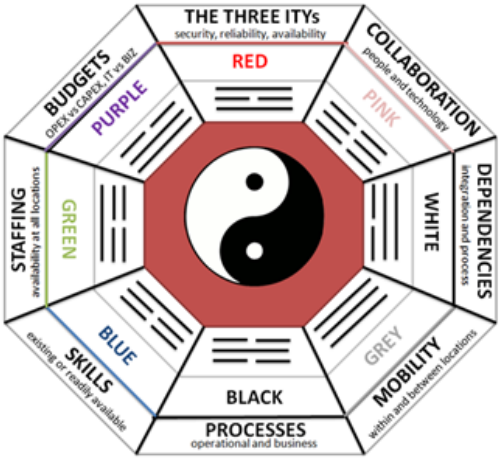
ARCHITECTURAL RELIABILITY

We've been building application delivery architectures for a while now with a key design goal in mind: built to fail. We assume that any given piece of the architecture might go belly up at any time, and architect a solution that takes that into consideration to ensure that availability is never (or as minimally as possible) impacted.

Hardware. Software. Any given piece of a critical system should be able to fail without negatively impacting availability. Performance may degrade, but availability itself is maintained. This often takes the form of "standby" systems; duplicates of a given infrastructure or application service that, in the event of a failure, are ready to stand in for the primary and continue doing what needs to be done. They're the second-stringers, the bench warmers, the idle resources that are the devil's playground in the data center.

And we're getting rid of them.

As we optimize the data center for cost and efficiency, we're eliminating the redundant duplication (see what I did there?) within the architecture and replacing it with something more aligned with the business goals of maximizing the return on investment associated with all that hardware and software that makes the business go. We're automating fail-over processes that no longer assume a secondary exists: instead, we automatically provision a new primary in the event of a failure from the much larger pool of resources that were once reserved. We're modifying the notion of architectural reliability to mean we don't need to fail-over, we'll just fail-through instead.



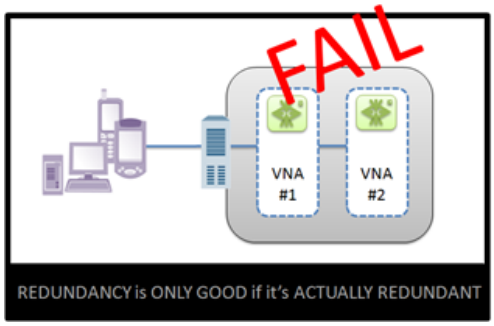
And that works, except when it doesn't.

SINGLE-POINTS of FAILURE

The danger here is two-fold: first, that we will run short of resources and be unable to handle any failure and second, that we can guarantee that the provisioning process can occur nearly simultaneously with the failure.

It can't. At least not yet. And while we're getting quite good at leveraging intelligent health monitoring and collaborative infrastructure architectures, we still haven't figured out how to predict a failure. Auto-scaling works because it does not account for failure. It assumes infinite resources and consistent availability. We can tell when an application is nearing capacity and adjust the resources accordingly ***before it becomes necessary***. And it is exactly that "before" that is important in maintaining availability and thus providing a reliable application. But we can't predict a failure, and thus we can't know when to begin provisioning until it's essentially too late. There are only two viable solutions: pre-provisioning, which defeats the purpose of such real-time automation and scalability services in the first place and reserved resources, which can have a deleterious affect on efficiency and costs – you're purposefully creating a pool of idle resources again. Both tactics have the same effect: idle resources waiting to be needed which runs contrary to one of the desired intents of implementing a virtualized or cloud computing-based infrastructure in the first place.

Thus, the definition of reliability as it pertains to our new, agile and cloud-based applications is directly related to the longest time required to either replace or provision any comprised component. Single points of failure, you see, are very bad for reliability. Especially when they are virtualized and it may the case that there are no resources available that can be used to "replace" the failed component. This is particularly important to note as we start to virtualize the infrastructure. It sounds like a good deal: virtual network appliances dramatically decrease the CapEx associated with such investments, but operationally you still have the same challenges to address. You still need a redundant system and they must reside on physically separate systems, in case the hardware upon which the virtual network appliance is deployed itself fails. That's true as well for applications; redundancy must be system-wide which means two instances of the same application on the same physical device invites unreliability.



And when you realize that you're going to need a physical system for every instance of a virtual network appliance, you might start wondering why it was that you virtualized them in the first place. Especially when you consider you exchanged nearly instantaneous serial-based fail-over for pretty fast network-based failure and a largely reduced capacity per instance. And of course any gains provided by purpose-built hardware acceleration that cannot easily (or cheaply) be duplicated in a virtualized environment. Oh, and let's not forget the potential of creating a single point of failure where there was none by eliminating the fail-to-wire option of so many infrastructure components. Almost every proxy-based network component "fails to wire" in the event of a failure resulting in a loss of functionality but not the ability to pass data, which means availability of the application is not compromised in the event a failure, although security or other functionality might be.Yes, you gained architectural multi-tenancy and simplified provisioning, but the need for such an implementation is quickly being erased by the rush by vendors to provide true multi-tenancy for network-based infrastructure and many of the gains in provisioning can be achieved using the same infrastructure 2.0 capable methods (APIs, SDKs) that are used to integrate virtual form factors.

The ability to react quickly, for agile operations, depends heavily on underlying architectural decisions. And it is the ability to react nearly instantaneously to failures throughout the entire infrastructure that enables a reliable, consistent application. Consider carefully the pros and cons of virtualization in every aspect of a deployment as it relates specifically to reliability with an eye toward aligning architectural decisions with business and operational requirements. This includes the business making decisions regarding “mission critical” applications. Not every application is mission critical, and understanding which applications are truly vital will go a long way toward cutting costs in infrastructure and management. A mission-critical application reliability requirement of 100% will likely remove some components from the virtualization list and potentially impact decisions regarding resource allocation/reservation systems. Single points of failure must be eliminated in critical application delivery chains to ensure reliability. Failure will happen, eventually, and a reliable infrastructure takes that into account and ensures a timely response as a means to avoid downtime and its associated costs.

- Data Center Feng Shui
- Operational Risk Comprises More Than Just Security
- The Number **of** the Counting Shall be Three (Rules **of** Thumb for Application Availability)
- Data Center Feng Shui: Fault Tolerance and Fault Isolation
- All Data Center Feng Shui posts on DevCentral
- Architectural Multi-tenancy
- The Question Shouldn’t Be Where are the Network Virtual Appliances but Where is the Architecture?
- I CAN HAS DEFINISHUN **of** SoftADC and vADC?
- The Devil is in the Details
- VM Sprawl is Bad but Network Sprawl is Badder

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com