We're Not Building a Piano: Design Patterns for Resilient Application Infrastructure – Part 1

When I was doing work as a general contractor and building a house, one of the teams I worked with was a Father-Son pair who were helping to build the house with me, and the owner of the house. The son of the team was nicknamed "Lumpy" and he was learning the trade. When Lumpy was hammering in a nail that went crooked and folded over, he began pulling the nail out. This happened a couple of times and was noted by his father by the phrase that I will never forget (apologies for the salty language):

> For Christ's sake, Lumpy, we're not building a f%&*ing piano

The choice of wording aside, the issue is that too much time was being spent trying to make each part of the building process ideal, which was likened to a finely-tuned piano. Lumpy was spending 3-4 times as long trying to remove the bent nail as he would have spent just hammering it in crooked and then putting another nail beside it.

This series takes the same concept and puts it into practice for infrastructure design. Only once we understand the design patterns can we apply them to where it really matters, which is in service of application availability.

Infrastructure Resiliency – Stop Building Pianos

Crooked nails will be all over your virtual and physical infrastructure. They should be. What we need focus on as infrastructure operations teams is moving further up the proverbial stack with our understanding of resiliency. We should be designing infrastructure using the same patterns as the "at-scale" folks use where possible. This is the concept behind what Alex Polvi (founder of CoreOS) calls GIFEE (Google Infrastructure for Everyone Else).

You don't need to be Google to think like them. The term SRE (Site Reliability Engineer) may seem like a buzzword title, but the concepts are sound and the fundamentals can be adopted more easily than we realize. It takes a little rethinking of what the goal of resiliency in infrastructure really is.

Bottom Up – Understanding N+1 and N+2

We use the phrases N+n to illustrate systems component resiliency. When we talk about things like server availability for virtualization clustering, N+1 and N+2 are often confused in their meaning. N+n is a measure of how many single components in a system can fail before critically affecting that system. If you have 12 hosts in a cluster, N+1 would indicate you have enough resources to survive a single node (virtualization host) failure and to continue service the remaining workloads. N+2 becomes a 2-node loss, and so on.

For host clustering and N+n illustration, we measure as a percentage of resources which is left over for the surviving nodes. A single-node system obviously cannot sustain any lost at all. A 2-node cluster can sustain a single node failure and survive but will have 50% of the total compute

resources.

Host Resource Loss Percentages							
Resiliency	1-Node	2-Node	3-Node	4-Node	5-Node	6-Node	7-Node
N+1	💀	50%	33%	25%	20%	17%	14%
N+2	💀	💀	66%	50%	40%	33%	28%
N+3	💀	💀	💀	75%	60%	50%	42%

This is just a sample showing N+1, N+2, and N+3 node loss effects in clusters up to 7-nodes. The calculations can be made quite easily using a simple formula:

> Remaining Resource Percentage = (Nodes Lost / Nodes Available) * 100

The interesting thing with cluster sizing is that we spend a surprising amount of time designing the clusters and then forget to keep track of the dynamic workloads. That's another blog all unto itself. Our goal in this series is to uncover the upper layers in which we can understand resiliency. Even if you do not directly affect these layers yourself as an operations admin or IT architect, it's my believe that we have a responsibility to know more to truly design and build resilient application infrastructure.

Understanding the True Full-Stack Infrastructure Resilience Approach

When somebody is described as a full-stack application designer, it usually means they are competent in both front-end (visual)) and back-end (application logic and data) design. For full-stack infrastructure architects, there are a lot more layers. An IT architect needs to understand the physical layer (servers, storage, network), the data layers (relational databases, NoSQL databases), the application layers (application logic, code logic and code deployment), and the access layers (front-end load balancing and caching). All of these need to also be understood on traditional virtualization and on private or public cloud infrastructure. Yikes!

Have no fear, we are going to take these topics on in some simple and meaningful examples, and you will have a crash course in resilient application infrastructure. Using these fundamentals will give us the foundation to then apply these patterns to specific infrastructure deployments like AWS, Microsoft Azure and private cloud products.

Strap in and enjoy the ride, and I hope that you find this series to be helpful!

NEXT POST: Full-Stack Infrastructure Understanding