By Slavik Dimitrovich, Solutions Architect, AWS

When architecting a system, how often do you start with availability requirements or service level agreement (SLA)? Most of us don't. And most of us shouldn't, because trying to build high availability into a system early on may slow things down and lead to over engineering. Yet, we should consider eventual desired level of availability and ensure that we can grow our system into it when the time comes.

In this article, I will use simple terms and a simple web application stack to discuss high availability, how it is calculated and how it can be practically achieved. I will also briefly touch on scalability and how similar approach can achieve higher availability and greater scalability of our simple web stack.

Start Simple

When we start a new system, it makes sense to keep things simple. At the early stage of any product, you don't really know whether it is even going to fly. Speed of iteration, ability to quickly respond to feedback is the most important attribute of your system.

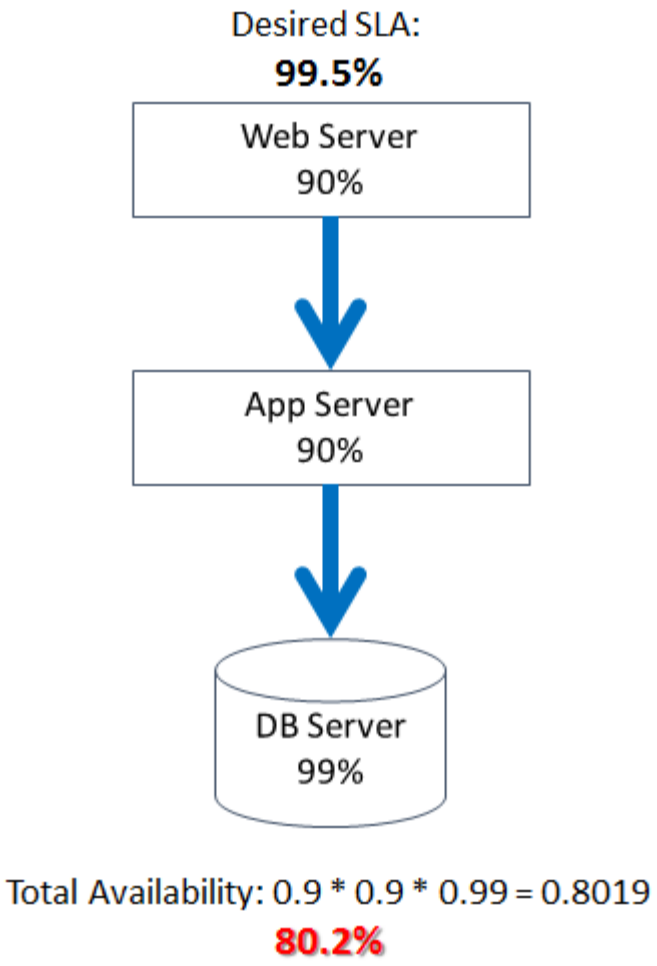
AWS provides you with a variety of high level services that manage entire parts of the system for you and if I were to start today, I would maximize the use of these resources, and thus maximize the time spent doing that special thing that my product is good at. For the purposes of this article, though, let us assume that I am a bit of an old school guy and I prefer your plain vanilla LAMP stack. In that case, the simplest way for me to start would be by keeping everything on a single box (or a single EC2 instance if I were to do it in the cloud). The main issues with this approach are of course scale and availability. You can only support as many customers as your single box and should this single box fail, you won't support any at all. And while these may

not be your primary concerns in the early stages, as your system matures and your customer base grows, availability and scale will become more important.

Split into Tiers

As the system grows, we often split it into tiers (also known as layers), which sets it on the path of greater scale and availability. Now every tier can be placed on its own box of the appropriate size and cost. We can choose bigger, better boxes with multiple levels of hardware redundancy for our database servers and cheaper, commodity-grade hardware for our web and application servers.

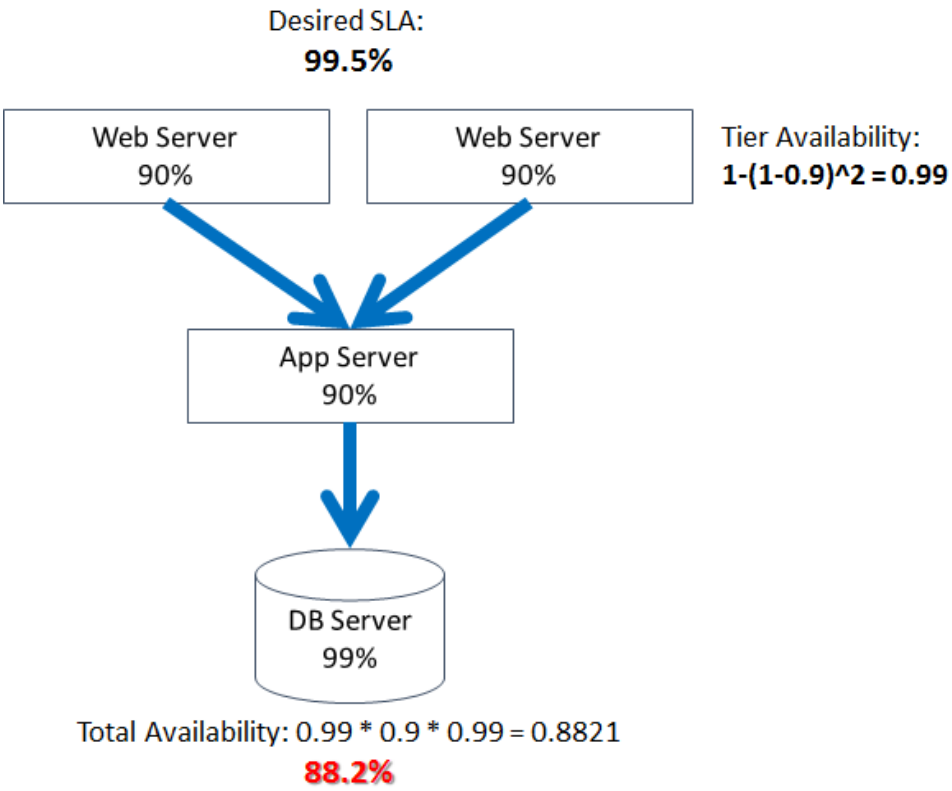
Let's say that our business requirements call for 99.5% uptime. In other words, it is allowed to be down no more than 44 hours in any given 12-month period. The total availability of a system of sequentially connected components is the product of individual availabilities. Let's for example assume that individual servers that host our web and application tiers have 90% availability and the server hosting our database has 99%. For simplicity's sake, let's also assume that these availability numbers include hardware, OS, software, and connectivity.



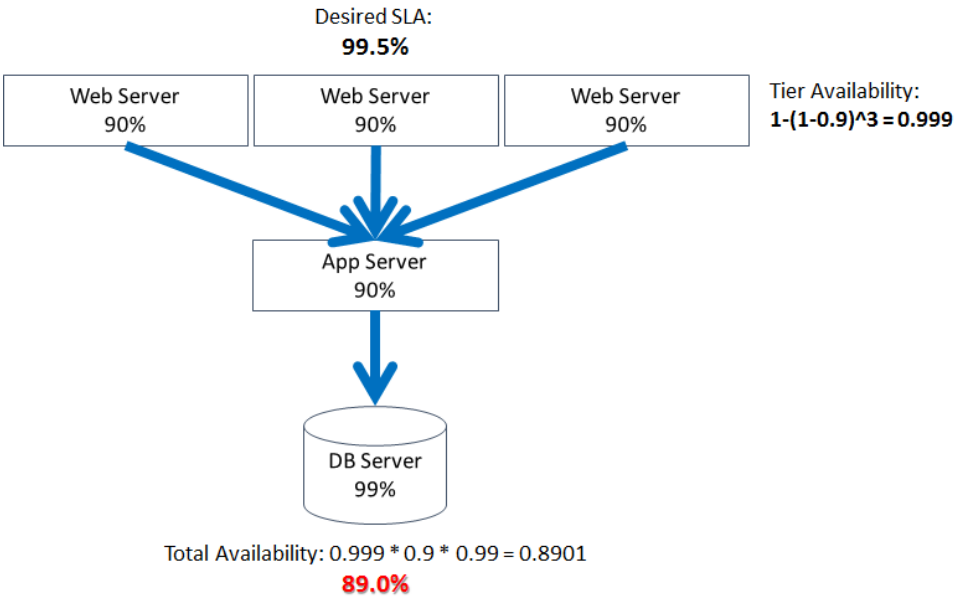
Introduce Redundancy

At the first glance, we are not doing too well: The availability of the whole system is way below our desired uptime and even below the availability of any single component of the system. However, by splitting the system into tiers and by pushing the state to the database tier, we have set the stage for solving (to some degree) the problems of availability and scale.

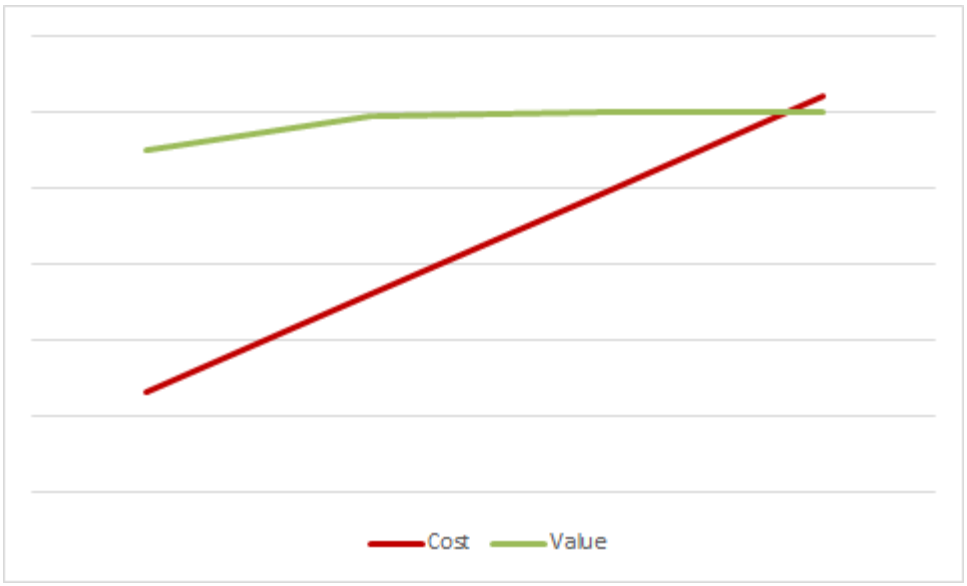
The simplest way to increase availability is by adding redundancy, which is much easier to do with the stateless parts of the system. As we add web servers to our web tier, the probability of its total failure decreases, pushing the availability of this tier and of the entire system up:



Adding even one extra server to the web tier brings its availability by 8%. Unfortunately, adding another web server does not do as much to our overall availability, increasing it only by 0.8%:



The cost of this tier grows linearly with every server, but availability grows logarithmically. Consequently, we will soon reach the point of diminishing returns where the value of additional availability will be less than the cost of adding another server:

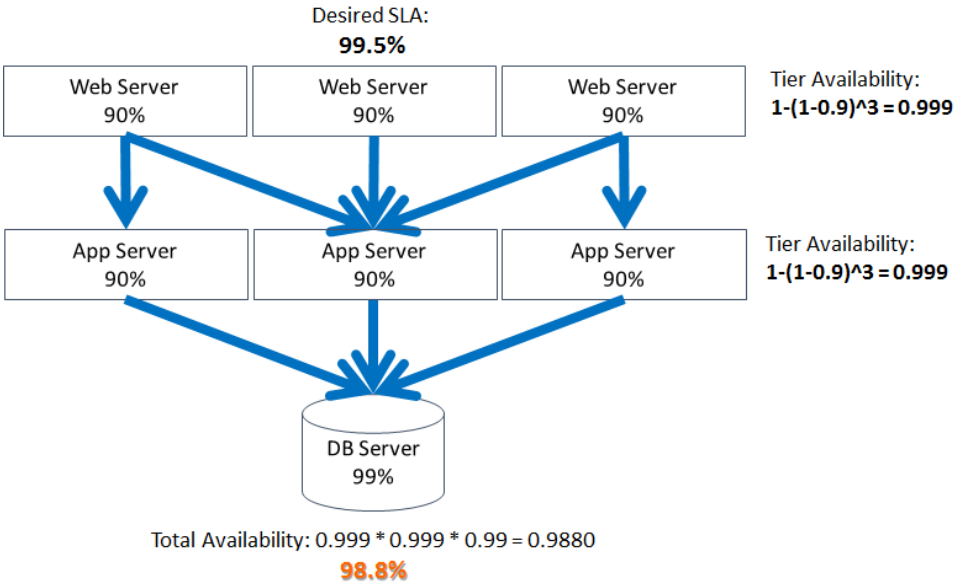


Thus, we have noticed the first pattern: Adding redundancy to a single component or tier leads to logarithmic increase in availability, eventually reaching the point of diminishing returns, at least from the availability standpoint.

Now, this is oversimplification of course, and there could be other reasons for adding more servers to your fleet. Overscaling in order to reduce the impact of server failure (AKA “blast radius”) could be one. This is an example of so-called N+1 deployment. Another reason could be scale (which will talk about later).

Expand Redundancy to Other Tiers

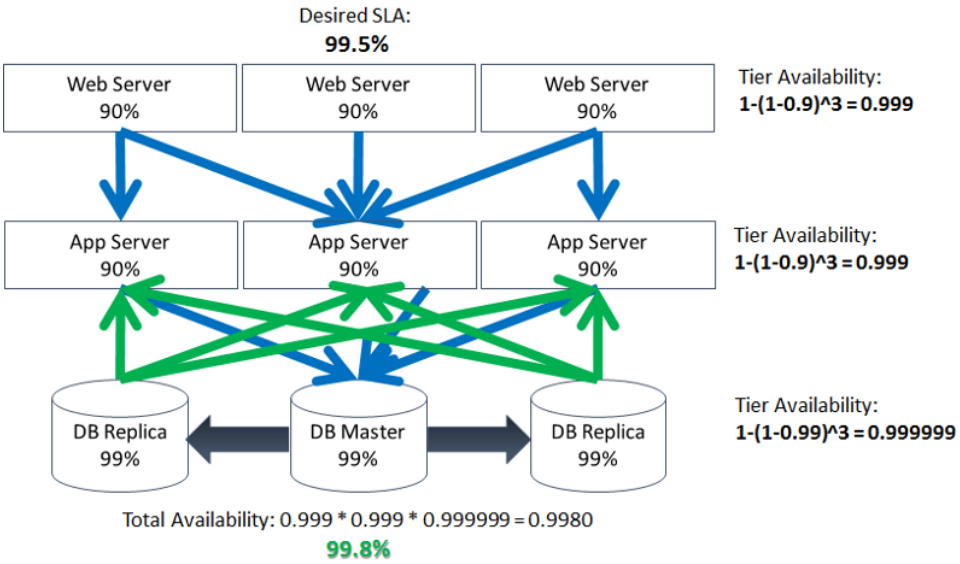
As we have exhausted our ability to have significant impact on the availability of our system by adding redundancy to the web tier, we need to look for new opportunities elsewhere. The next most logical place is another stateless part of the system, the application tier:



Again, we have gained around 8% after adding a second box and the incremental availability gains started diminishing quickly after that, eventually also reaching the point of diminishing returns.

As this point, we have noticed a second pattern: The availability of your entire stack cannot exceed that of its least-available component or tier.

It looks inevitable that we have to add redundancy to your data tier as well:



And sure enough, doing so solves the problem, at least on paper. In reality, adding redundancy to a statefull component bring with it additional challenges, but that is a topic of another blog.

Data Center Redundancy

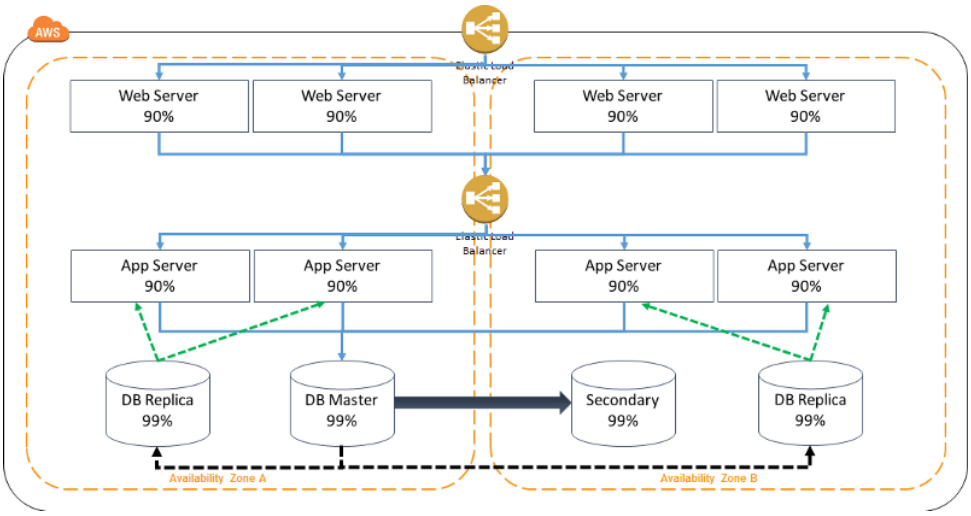
Let's now take this one step further and take a look at another part of the stack that has been assumed all along, but never brought to light, the data center that hosts it. Should it go down due to power outage, Internet connectivity issues, fire or natural disaster, your entire stack will become inaccessible, rendering all investments of time and money that we made into your multi-node multi-tiered redundant stack useless.

We can choose to host our stack in a Tier IV data center, which often will cost us a bundle and still may not offer sufficient protection against natural disasters. Or, we can apply the same principle of redundancy to the data center itself and spread our stack across multiple data centers. Wait, what? Multiple data centers? What about latency? What about maintenance overhead?

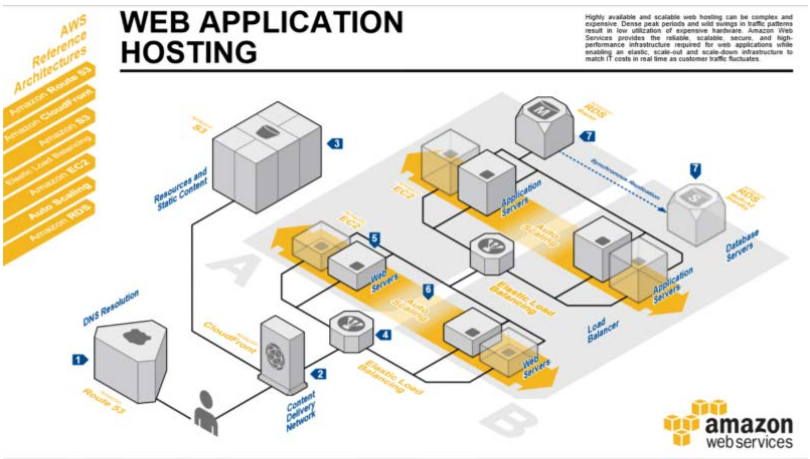
All these and other questions may cross your mind when you read the last suggestion. And under normal circumstances, you would be correct. In order for this approach to be successful and cost effective, we need data centers that are:

- Built for high reliability so that nothing short of a natural disaster brings one down
- Located close to one another and connected via low-latency high-bandwidth links to ensure low latency between the facilities
- Located far enough from each other that a single natural disaster cannot bring all of them down at the same time.

This definition sounds familiar to some of us, doesn't it? It sounds like an AWS Availability Zone (or AZ for short)! By moving our stack to AWS, we can spread it across multiple data centers just as easily as within a single one. And it will cost us just as much:



If we add more bells and whistles, such as hosting our static assets in AWS Simple Storage Service (S3), serving our content through a CDN—AWS CloudFront and add the ability to scale both stateless tiers automatically (AWS Auto-Scaling), we'll arrive at this canonical highly available web architecture:



Web application architecture:
http://aws.amazon.com/architecture/

A Few Words About Scalability

And to top things off, let's also briefly talk about scalability. By splitting our stack into tiers, we made it possible to increase the scale of each tier independently. When scaling a component of a system, two approaches can be used: Getting a bigger box (also known as scaling up or vertical scaling) or getting more boxes (also knowns as scaling out or horizontal scaling).

Vertical scaling may be easier from operational standpoint, but has two major limitations: First, bigger boxes tend to get more expensive faster than the additional scale they provide. Second, there is a limit to how big of a box you can get. Horizontal scaling offers more room to grow and better cost

efficiency, but introduces additional level of complexity, requiring higher level of operational maturity and high degree of automation.

The added redundancy that we discussed earlier from the point of view of high availability, can also serve as a scaling mechanism. As we get more traffic (and hopefully more revenue), we can add more nodes to each tier to support this increase in load. Unlike the high-availability approach, horizontal scaling does not suffer from the effect of diminishing return, as long as the value of additional traffic grows at least linearly. However, in order to make this scaling solution even more cost effective, we need the ability to add *and remove* nodes dynamically, in response to fluctuations in traffic. Wait, this sound familiar, at least to some of us. This sounds like Auto Scaling—a feature natively supported by AWS EC2!

With Auto Scaling, we can let our stack grow out and back in horizontally in response to actual traffic fluctuations, completely automatically, and across multiple data centers.

By hosting your stack on AWS, you can achieve highest levels of availability, including data center redundancy, as well as dynamic horizontal scalability in an easy and cost effective way.

Summary

a) Splitting a system into tiers allows managing the availability and scalability of each tier independently.

b) Adding parallel nodes to a tier/component increases overall availability of that tier and the system as whole. The increase is logarithmic however, which leads to diminishing returns.

c) The availability of the whole system cannot be greater than that of its least available components/tier.

d) Data center availability should be considered as part of total system's availability.

e) AWS makes it very easy to build highly redundant, multi-data-center systems, reaching several nines of availability that also support dynamic horizontal scalability.

f) Adding redundancy to the data tier becomes necessary once it becomes the availability bottleneck. Doing so however, brings with it new challenges which we will cover in Part 2 of this blog series.

References

- AWS Architecture Center

- AWS re:Invent 2014 | (BAC404) Deploy High Availability & Disaster Recovery Architectures with AWS
- Scaling on AWS for the first 10 million users

TAGS: Guides & Best Practices, Startup Spotlight

AWS Admin

Amazon Web Services