Build to Scale or Build to Fail

Robert Haynes, 2015-21-09

Huge market opportunities in the internet space don't come along very often. When they do, you need to be ready.

In theory, the elastic and pseudo-unlimited resources of public cloud computing make scaling up easy. That's only true if your underlying architecture is designed to work that way. Often what is quick and effective to get a beta project off the ground is about the worst architecture you can pick to absorb a 100X spike in traffic.

This is pretty much what happened to voat.co a couple of months ago. Disgruntled users from the popular social news site reddit.com tried to flock to the greener pastures of voat.co, a Swiss based social site running in Microsoft Azure. Sadly their infrastructure wasn't ready for the influx. A huge market opportunity missed.

In fairness to them, voat started as a hobby project to entertain a developer, and not as a business venture, and I'm not criticizing them for not being able to capitalize on the opportunity, but it's an interesting case study, none the less.

Ten years ago the guys at voat would have been running around trying to rack servers and load balancers, and, even with luck and caffeine, scaling up would take days. But with the cloud it's supposed to just scale on demand, right? Only if you have built a scalable architecture to start from.

For a great technical write up on the scalability challenges that voat.co will face, I suggest you take a look at this excellent blog post from Aaron Stannard who delves more deeply into the voat.co code and the fundamental problems they will have scaling up. Reading his post and looking at the other scalability challenges I've seen customers face over the years, it's clear that two key principles would have prevented, or at least lessened a lot of them:

1. Get the logical data storage layer right.

2. Make sure every component can scale independently.

Here's where I have to admit that I'm not an expert on scaling database layers – but there's plenty of information and expertise out there.

Independently scaling components is just as important. It can be hard to predict which components will be bottlenecks before you have built the system. Tightly coupling too many components makes scaling harder and less efficient. Moving towards a more decomposted architecture (such as the micro services model that is gaining popularity at the moment) allows you to scale individual components or processes on demand - assuming you have a way to horizontally scale them out – (I might know a company that is good at that sort of thing). By being able to build fine grain scaling into your application you can both meet fluctuating demand more effectively and save on overprovisioning larger instances of monolithic code.

This kind of architecture also allows you to optimize just the components that are the current bottleneck, without rebuilding the application. Place each set of components behind a decent load balancer and you can bleed in the new optimized code versions and retire the older version without disrupting service.

So, if you are building a hobby project for fun and interest, or are aiming to wow consumers with the Next Big Thing^{TM} - build for scale from day one. Otherwise, by the time you need to, it will probably be too late.

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

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