Netflix: Developing, Deploying, and Supporting Software According to the Way of the Cloud

Monday, December 12, 2011 at 9:05AM

Todd Hoff in Strategy, netflix

At a Cloud Computing Meetup, Siddharth "Sid" Anand of Netflix, backed by a merry band of Netflixians, gave an interesting talk: Keeping Movies Running Amid Thunderstorms. While the talk gave a good overview of their move to the cloud, issues with capacity planning, thundering herds, latency problems, and simian armageddon, I found myself most taken with how they handle **software deployment in the cloud**.

I've worked on half a dozen or more build and deployment systems, some small, some quite large, but never for a large organization like Netflix in the cloud. The cloud has this amazing capability that has never existed before that enables a novel approach to fault-tolerant software deployments: *the ability to spin up huge numbers of instances to completely run a new release while running the old release at the same time*.

The process goes something like:

- A **canary machine** is launched first with the new software load running real traffic to sanity test the load in a production environment. If the canary doesn't die they move on with the complete upgrade.
- **Spin up** an entirely new cluster of instances to run the new software

release. For Netflix this could be hundreds of machines.
- **Keep the old cluster running** on the old release, but tell your load balancer to switch all requests to the new cluster running the new release.
- Let the new **cluster bake** for a while.
- If there aren't any problem **tear down the old** cluster and the new cluster is now the operational cluster of record.
- If there are problems **redirect requests to the old** cluster, tear down the new cluster, and figure out what went wrong.
- Downstream services see the same traffic volume, so the process is transparent.

Previously:

- You would **never have enough free machines** that new code could run in parallel with the old. That would be over provisioning to an impossible to afford degree. That made clumsy rolling upgrades the technique of choice.
- It's unlikely you would have the load balancer infrastructure under enough **programmer control** to pull off the switch and rollback. That stuff in usually under the control of the network group and they don't like messing with their configuration.

The cloud makes both of these concerns **old school**. The elasticity of the cloud (blah blah) makes spinning up enough instances for a hosting a new release both affordable and easy. The load balancer infrastructure in the cloud both has an API and is usually under programmer control, so programmers can mess everything up with abandon.

You might at this point raise your hand and say: **what about that database**? Schema migration issues are always a pain, so this approach does not apply to the database. The database service layer still uses a

rolling upgrade path, but it is hidden behind an API and load balancer so the process is somewhat less painful to clients.

How Netflix's Team Organization Implies a Software Architecture and Policies

Netflix has some attributes about their software team infrastructure and software architecture that makes this approach a good fit for them:

- **Their architecture is service based**. Many small teams of 3-5 person teams are completely responsible for their service: development, support, deployment. They are on the pager if things go wrong so they have every incentive to get it right. Doubly so because most of Netflix's traffic happens on the weekend and who wants to get paged on the weekend?
- **There's virtually no process at Netflix. They don't believe in it. They don't like to enforce anything.** It slows progress and stunts innovation. They want **high velocity** development. Each team can do what they want and release whenever they want, how often they want. Teams release software all the time, independent of each other. They call this an "optimistic" approach to development.
- **Optimism causes outages**. If you are going to be optimistic and have an absence of process, you have to have a way to detect problems and recover from them. Netflix's Chaos Monkey approach was created to find any problems caused by their optimism. Similarly, their rollback approach to software releases makes deployment almost hitless and transparent, so you can tolerate faults in a release. Code is first tested in a staging environment, but production is always the real test.

Use Load Balancers for Isolation

Their architecture is divided into layers: an Internet facing Edge layer; Middle layer serving the edge layer; Backend layer. All this rests on Amazon services. Each layer is fronted by a load balancer and each layer has the ultimate goal of being auto scaling.

The load balancer allows the isolation of components. It's possible to spin up a parallel cluster and route requests behind the load balancer without any other service layer having an idea that it is being done.

For the Internet facing services they use Amazon's load balancers because they have public IP addresses. For the other layers they use their own service discovery service which handles load balancing internally.

Function follows form. We conform to and innovate on the capabilities of the underlying system and our forms eventually come to reflect that function. It will be curious to see how experience in the cloud will cause changes in long held development practices. That's the *Way of the Cloud*.

Related Articles

- LinkedIn: Quick Deploy: a distributed systems approach to developer productivity
- Google: Build in the Cloud: How the Build System works
- Etsy: Design for Continuous Deployment
