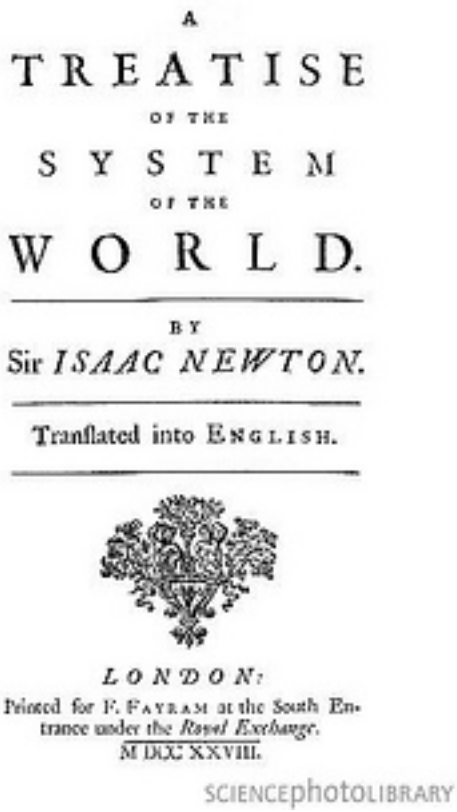
Startups are Creating a New System of the World for IT

Monday, May 7, 2012 at 9:15AM

Todd Hoff in cloud

It remains that, from the same principles, I now demonstrate the frame of the System of the World. -- Isaac Newton



The practice of IT reminds me a lot of the practice of science before Isaac Newton. Aristotelianism was dead, but there was nothing to replace it. Then Newton came along, created a scientific revolution with his System of the World. And everything changed. That was New System of the World number one.

New System of the World number two was written about by the incomparable Neal Stephenson in his incredible Baroque Cycle series. It explores the singular creation of a new way of organizing society grounded in new modes of thought in business, religion, politics, and science. Our modern world emerged Enlightened as it could from this roiling cauldron of forces.

In IT we may have had a Leonardo da Vinci or even a Galileo, but we've never had our Newton. Maybe we don't need a towering genius to make everything clear? For years startups, like the frenetically inventive age of the 17th and 18th centuries, have been creating a New System of the

World for IT from a mix of ideas that many thought crazy at first, but have turned out to be the founding principles underlying our modern world of IT.

If you haven't guessed it yet, I'm going to make the case that the New System of the World for IT is that much over hyped word: **cloud**. I hope to show, using many real examples from real startups, that the cloud is built on a powerful system of ideas and technologies that make it a superior model for delivering IT.

IT has had an explosion of creativity: open source, deep and powerful tool chains, lean and agile development, cloud computing, virtualization, BigData, parallel programming, distributed monitoring, distributed programming, NoSQL, cost driven programming, dynamic languages, real-time processing, asynchronous programming, distributed teams, mobile platforms, viral loops, flat networks, software defined networking, wimpy cores, DevOps, everything as a service, infrastructure as code, and so on and so on. Astounding innovation wherever you look.

We are just now figuring out what new structures and systems are replacing the old, but if you step back a bit, what seems to be happening is we are creating a new "frame" using a bottom up methodology that just may be a new System of the World for IT. What is merging is a new way of working synthesised from all the diverse forces catalogued above. We've created a sort of new physics of development in place of a collection of prescientific alchemical lore.

Since it is startups tackling problems that can't be solved using traditional methods, it is through them that we'll explore this new System of the World or IT.

It's Not All About the Cloud, but It's Mostly

About the Cloud

These days the story of startups primarily revolves around the cloud in one way or another. Not completely, not totally, but usually. That's my inescapable observation based on all the architecture profiles I've written on HighScalability.com. Most involve the cloud.

Not all startups choose the cloud, many do not, but even if a startup doesn't join a formal cloud, we still see the development of cloud-like infrastructures and the deployment of cloud inspired tool chains. So we'll just skip all the old arguments about OpEx vs CapEx, IaaS vs PaaS vs SaaS, virtualization vs bare metal, public vs private vs hybrid clouds, and open vs closed clouds. Those are all just business decisions made in the pursuit of business goals.

Which specific choices are made isn't all that important, which is why I'll use the term cloud in a generic sense. By cloud I do not mean any particular cloud provider or technology. Zynga, for example, used Amazon extensively, now they've built their own cloud to have more control, use fewer servers, and save money. But what they built is still a cloud.

There is a line of controversy worth pursuing that goes something like this: the cloud is no different than what we have been doing in datacenters for years, so what's the big deal? The cloud is certainly a systematization and productization of capabilities traditionally found in a well staffed datacenter. So in that way the cloud is nothing new.

The key differentiators between a cloud and a datacenter are often said to be multitenancy, geographical distribution, and elasticity. I want to say the key difference between a cloud and a datacenter is **democratization**. Where once only a few companies could leverage advanced datacenter services, now everyone, great and small can exploit the same capabilities.

What was once private is now public. What was once specialized is now generic. What was once scarce is now abundant. Programmers jumped on all these new capabilities and turned them into the most sophisticated ecosystem for IT that we've ever seen. That's a big deal.

So it is in cloud inspired features that a New System of the World can be found, not any particular instance of the cloud.

The Old Datacenter Versus The New Cloud

The quickest way I can think of to illustrate what the New System of the World for IT looks like is to consider the innovative work Netflix is doing in replacing their "in-house IT with the cloud for non-trivial applications with hundreds of developers and thousands of systems."

Netflix is the poster child for moving from the datacenter to the cloud because they've actually done it. Netflix ran their own datacenter and are now 100% cloud. Along the way they've done a lot original thinking and on what it means to run an IT-centric business in the cloud. Adrian Cockcroft, a Cloud Architect at Netflix, has created an amazing Cloud Architecture Tutorial documenting what they've learned.

What follows is a list of some major transitions Netflix has made in going from the datacenter to the cloud. The list is a synthesis of slides in the tutorial. It paints a clear picture of how IT in the cloud is different than IT in the datacenter:

Old Datacenter	**New Cloud**
Licensed and Installed Applications	SaaS (Workday, Pagerduty, EMR)

Central SQL Database	Distributed Key/Value NoSQL
Sticky In-Memory Session	Shared Memory Cache Session
Tangled Service Interfaces	Layered Service Interfaces
Instrumented Code	Instrumented Service Patterns
Fat Complex Objects	Lightweight Serialized Objects
Components as Jar Files	Components as Services
Chatty Protocols	Latency Tolerant Protocols
Manual and Static Tools	Automated and Scalable Tools
SA/Database/Storage/Networking Admins	NoOps/OpsDoneMaturelyButStillOps
Monolithic Software Development	Teams Organized around Services
Monolithic Applications	Building Your Own PaaS
Static and Slow Growing Capacity	Incremental and Fast Growing Capacity
Heavy Process/Meetings/Tickets/Waiting	Better Business Agility
Single Location	Massive Geographical Distribution
Vendor Supply Chains	Direct to Developer
Focus on How Much it Costs	Focus on How Much Value it Brings
Ownership/CapEx	Leasing/OpEx/Spot/Reserved/On Demand

Some principles we see at work are a move to distributed architectures, a focus on generating business value through agility and flexibility, a move away from ownership as a core competency, a separation of concerns

along services boundaries, a decentralization and reorganization of processes around services, and a push of responsibility to as close to the developer as possible.

We'll explore some of these ideas in later sections, but I think this makes it clear we aren't just talking business as usual, when taken altogether we are talking about something new. It's a complete transformation at every level.

If you want to say we can do all this in the datacenter I can't argue, because clouds are built on datacenters. Though I would argue, that once a datacenter can do all these things, it has become a cloud.

The IT World is Now Flat

Although the New System of the World was pioneered by startups, what has developed, strangely enough, serves to make any enterprise development group just as agile as any startup. The IT world has become flat. There's now a level playing field across all of IT. The cloud has changed the core economic concepts of delivering business value on top of IT.

A small team in any company can recognize an opportunity, create a product within a week, have it run in many different locations worldwide, with almost no startup capital, and with a low sysadmin burden. Idea to innovation in the time it would have previously taken to work up a hardware request budget proposal.

For some time we've had practices like: agile development, extreme automation, short development iterations, continuous integration, continuous deployment, continuous testing, small dedicated teams, and so on. These practices, although much talked about, were seldom implemented. What slowed adoption was a missing element: the cloud's

programmable IT fabric.

Previously a complex and highly specialized stack was required to follow the agile path. Now it's easy for any group to develop software this way. And we've seen startup after startup adopt these strategies, creating a total revolution in practice on everything about how software is created, distributed, and maintained.

One reason for this revolution is explained by Etsy in terms of Conway's Law:

> *When a team makes a product the product ends up resembling the team that made it.*

I'll extend this notion to say the team and thus the product end up resembling the underlying technology used to make it. When you change the underlying development infrastructure, by moving to a cloud, you are bound to change teams and processes they create.

Here are a few examples from startups of how pretty much everything has changed:

- Instagram: **Give me a place to stand and with a lever I will move the whole world**. An organization with 2 backend engineers can now scale a system to 30+ million users and be bought for a one billion dollars. Regardless of your opinion on the purchase price, the ability for a small organization to handle such a huge user base is an unprecedented amount of leverage.
- Fidelity. Fidelity is not a startup, but they are creating a next generation internal cloud, saying that the cloud and BigData are creating new rules for IT organizations to innovate. No longer will they be hampered by the organization.
- Netflix: **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.

- Netflix: **NoOps**. “We have hundreds of developers using NoOps to get their code and datastores deployed in our PaaS and to get notified directly when something goes wrong. We have built tooling that removes many of the operations tasks completely from the developer, and which makes the remaining tasks quick and self service. There is no ops organization involved in running our cloud, no need for the developers to interact with ops people to get things done, and less time spent actually doing ops tasks than developers would spend explaining what needed to be done to someone else.”
- Etsy: **Continuous deployment**. Any engineer in Etsy can deploy the whole site to production at anytime. Happens 25 times a day because it’s so easy. It’s a one button deploy. Small change sets are going out all the time, not large deployments. If things go wrong they can quickly figure out what went wrong and fix it. Compare this to the infrequent big bang software updates that are typical.
- Etsy: **QA is performed by developers**. Development makes production changes themselves. This has the effect of bringing them closer to production, which enables having an operability mindset. This is opposed to having a ship-to-QA-and-consider-it-done mindset. Developers deploying their own code also brings accountability, responsibility, and the requisite authority to influence production. No Operations engineers stand in the way of a Development engineer from deploying.
- Facebook: **Small, independent teams with both responsibility and control**. Small teams allow work to be done efficiently, quickly, and carefully. Only three people work on photos, for example, the largest photo site on the Internet. But responsibility

requires control. If a team is responsible for something they must control it. For example, Facebook pushes code into production everyday. The person who wrote the code is there to fix anything that goes wrong. If the responsibility of pushing and wring code are split, then the code writer doesn't feel the effect of code that breaks the system. Compare this to the typical separation of developers, QA, and DevOps.

- Facebook: **Move Fast**. At every level of scale there are surprises. Surprises are quickly dealt with using a highly qualified cross disciplinary team that is flexible and skilled enough to deal with anything that comes up. Flexibility is more important than any individual technical decision. By moving fast Facebook is also able to try more options and figure out which ones work best. Compare this to the typically heavy weight planning and development processes.
- TripAdvisor: **No architects, engineers work across the entire stack**. You own your project end to end, and are responsible for design, coding, testing, monitoring. Most projects are 1-2 engineers. If you do not know something, you learn it. The only thing that gets in the way of delivering your project is you, as you are expected to work at all levels. Compare this to the islands of specialization that are typical in IT.
- Amazon: **You build it, you run it**. Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.

This is not how software development has been done in the past. What makes it possible is the leverage gained by an IT programming fabric that treats a datacenter and its contained services as being software scriptable. From this base very powerful tool chains like Rightscale, Chef, Puppet, and dozens of others have been developed to make it possible for small teams to quickly do a lot with a little.

Reducing the Mean Time Between Big Ideas

The most startling change post cloud is the increasing pace of innovation. Netflix sees the cloud as laboratory for reducing the mean time between big ideas. James Urquhart says the cloud is a lottery system for developers. Developers can implement something quickly and cheaply, hope it gets 10 million users, hope it succeeds big and if it doesn't, it wasn't that expensive to fail.

The software startup landscape itself has been changed forever. Previously you would go to a VC for the many millions needed to even begin an idea, now you are expected to have a prototype ready before even seeing a VC.

Here are a few examples of how startups are making use of these new capabilities to innovate:

- Robert Scoble talks about how the flood of new startups is just starting. Startups are starting all over the world, not just Silicon Valley or New York. Now you can start a startup in the middle of nowhere India. The costs of starting a startup have gone way down. Ycombinator used to just have 10 companies a class come out, now they have 60. And each month there are more and more incubators. Two kids can start Instagram and they can start it anywhere.

- TripAdvisor: Engineering can be best compared to running two dozen simultaneous startups, all working on the same code base and running in a common distributed computing environment. Each of these teams has their own business objectives, and each team is able to, and responsible for, all aspects of their business. Each of the teams operates in the way that best fits their distinct business and personal needs, this process is best described as "post agile/scrum".
- Netflix: Runs in Amazon so they can innovate and not have to worry about growth in the future.
- Netflix: "We built a completely cloud based infrastructure in the US and did some work extracting it so we could actually deploy it anywhere. We set up a bunch of test machines in the AWS Ireland facility and we built the ability to replicate data across both sites. In total we set up 1,000 machines in Ireland. If we had built our own data centre then we would have had to lay down a large amount of money in, say, six months in advance for a really efficient build out, and instead we could use that money to buy movies."
- Playfish: The cloud allows Playfish to innovate and try new features and new game with very low friction, which is key in a fast moving market. The cloud allows them to concentrate on what makes them special, not building and managing servers.
- Zynga: Zynga uses the cloud to deploy their applications and prove them out while handling the load during the process. They then fold applications back into their datacenter once the growth trajectory has been established. It's not about saving money, it's about growing business.
- Steve Lacy: Amazon's EC2 is a better ecosystem for fast iteration and innovation than Google's internal cluster management system. EC2 gives me reliability, and an easy way to start and stop entire services, not just individual jobs.

Typically a datacenter is a lock, a point of serialization for developers that creates a vertical barrier through the entire stack. By unshackling developers from IT infrastructure people it opens up the possibility space and developers can do new things they could never do before.

Of course, the distributed infrastructure of the Internet is essential to the low friction creation and dribution idea and the building of teams and sharing code. And the web and mobile are far more fertile niches for startups than any enterprise landscape. Yet the cloud, by creating an elastic usage model for all services developers, has unshackled developers. Developers can now be sure everything will just work without first having to ask permission. The entire cycle is now developer driven which has thrown an accelerant on the fire of innovation.

It's Open Source All the Way Down

The foundations for this New System of the World sit squarely on Open Source software. There is virtually no startup you can name that is not built primarily on Open Source. Take a look at Tumblr's stack as a quick example: Linux, Apache, PHP, Scala, Ruby, Redis, HBase, MySQL, Varnish, HA-Proxy, nginx, Memcache, Gearman, Kafka, Kestrel, Finagle, Thrift, HTTP, Func, Git, Capistrano, Puppet, and Jenkins.

It's all open source and Tumblr is by no means unique, this is a common pattern.

Open Source started with small libraries and has moved up stack with ever larger and more sophisticated components, applications, tools, languages, and operating systems. Now we are seeing movement into Open Source hardware, networking, and even Open Source clouds. At one time this was not true. At one time most software was developed with closed source tool chains. That has completely changed.

While Open Source was firmly established in the programming tools arena, LiveJournal was probably the early example of creating and open sourcing more sophisticated infrastructure tools like memcached and MogileFS. And possibly even more important was that they took the time to talk about the architecture challenges they faced and how they solved them. LiveJournal was the prototype for the early web.

This attitude helped create a virtuous circle in the development community, spawning a tradition that has continuously become more generous and more productive over time. Major companies like Netflix, Twitter, LinkedIn, Google, and Facebook are not only first to tackle scaling challenges, but they Open Source many of the solutions. And more importantly, they share their experiences and lessons learned with the whole community.

The impact of Open Source on productivity and innovation has been transformative. The advantage Open Source gives you is time. You can do more in less time. If you want to plug into this productivity cycle then you need to align yourself with the Open Source ecosystem. It’s not just for startups, it’s for anyone developing products. Use closed source where it offers a competitive advantage, but the fastest innovation is happening in the Open Source community and that’s with whom you want to make alliances.

It’s Loosely Coupled Services All the Way Down

If Open Source is the foundation for the New System of the World then Service Oriented Architectures are the load bearing walls. As we’ll see, services are not just a software architecture feature anymore, but they’ve become the organizing principle around how teams and software are constructed.

Services have been around forever. Client-server programming was invented as a way for applications to take advantage of networks of computers. This idea was lost on early web architectures that stuffed everything into two or three tier architectures. A browser talked to a web server that invoked code that would return a web page. That code might talk to a database, but it was always a monolithic self-contained blob. As web sites needed to scale, programmers rediscovered client-server programming and started breaking down monolithic applications into cooperating collections of services. Services started talking to other services and soon web servers weren't application servers anymore, but just a thin layer around a set of service calls. The dependence of rich UIs and mobile applications on backend services has simply continued this evolution.

Here's a how a number of startups are using Service Oriented Architectures:

- Wordnik: "We've made a significant architectural shift. We have split our application stack into something called Micro Services. The idea is that you can scale your software, deployment and team better by having smaller, more focused units of software. The idea is simple — take the library (jar) analogy and push it to the nth degree. If you consider your "distributable" software artifact to be a server, you can better manage the reliability, testability, deployability of it, as well as produce an environment where the performance of any one portion of the stack can be understood and isolated from the rest of the system. Now the question of "whose pager should ring" when there's an outage is easily answered! The owner of the service, of course."
- Playfish: Service Oriented Architectures are used at Playfish to manage complexity. As new games are added code is split into

different components that are managed by different teams. This helps keep the overall complexity of the system down, which helps make everything easier to scale.

- Amazon:
  - The big architectural change that Amazon made was to move from a two-tier monolith to a fully-distributed, decentralized, services platform serving many different applications. Their architecture is loosely coupled and built around services. A service-oriented architecture gave them the isolation that would allow building many software components rapidly and independently. Grew into hundreds of services and a number of application servers that aggregate the information from the services.
  - Services are the independent units delivering functionality within Amazon. It's also how Amazon is organized internally in terms of teams. If you have a new business idea or problem you want to solve you form a team. Limit the team to 8-10 people because communication hard. They are called two pizza teams. The number of people you can feed off two pizzas. Teams are small. They are assigned authority and empowered to solve a problem as a service in anyway they see fit.
- Amazon: “If you think about infrastructure as a service and platform as a service (PaaS), what we've built is a PaaS over the top of the AWS infrastructure, which is as thin a layer as we could build, leveraging as many Amazon features as seemed interesting and useful. Then we put a thin layer over that to isolate our developers from it.”
- Netflix: “If you think about infrastructure as a service and platform as a service (PaaS), what we've built is a PaaS over the top of the AWS infrastructure, which is as thin a layer as we could build, leveraging as many Amazon features as seemed interesting and

useful. Then we put a thin layer over that to isolate our developers from it."

- Netflix: 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. They've built a decoupled system where every service is capable of withstanding the failure of every service it depends on. Everyone is sitting in the middle of a bunch of supplier and consumer relationships and every team is responsible for knowing what those relationships are and managing them. It's completely devolved — they don't have any centralised control. They can't provide an architecture diagram, it has too many boxes and arrows. There are literally hundreds of services running.
- Facebook: Each layer is connected via well defined interface that is the sole entry point for accessing that service. This prevents nasty complicated interdependencies. Clients hide behind an application API. Applications use a data access layer. Application logic is encapsulated in application servers that provide an API endpoint. Application logic is implemented in terms of other services. The application server tier also hides a write-through cache as this is the only place user data is written or retrieved, it is the perfect spot for a cache.
- Tumblr: Built a kind of Rails scaffolding, but for services. A template is used to bootstrap services internally. All services look identical from an operations perspective. Checking statistics, monitoring, starting and stopping all work the same way for all services.
- Justin.tv: "The shift first started with the ascendancy of native mobile apps. Now, developers had to seriously start considering their HTTP APIs as first-class citizens and not nice-to-haves. Once that happened, it's not a big leap to realize that treating your web

> application as somehow different from any of your native clients is a bit, well, insane."

Now everything is kind of like it was before: service based, message passing based, distributed, real-time, queue based, and completely asynchronous. The tools to accomplish all this are different of course, but in principle they are similar.

What's radically different from the past is the unification of services by rearchitecting entire products as a PaaS. This is made possible by a suite of scalable services linked together using a distributed IT fabric. Architectures can now be elastic and adaptive in ways that are still being explored.

Lifecycle of a Project: Public Cloud to Private Cloud -- or Vice Versa -- Or Both

New in this New System of the World is the idea of federated compute spaces that applicaton functionality can flow between depending on business objectives.

Zynga is the most famous practioner of this form of cloud thermodynamics. Zynga used the public Amazon cloud to deploy their applications, prove them out, and handle load during the initial phases of the release process. Then, once the growth trajectory had been established, they folded the application back into their own datacenter.

It wasn't an architecture decision based on saving money, it was about growing the business. Zynga has matured and they are now moving off Amazon, into their own private cloud, in search of lower costs and better performance, but they've created an enduring architectural pattern that will work for anyone.

The ability for a business to target business goals with this degree of risk management flexibility was virtually impossible in the rack'em and stack'em age.

Cost Driven Architectures

In the New System of the World how applications are architected has changed forever with the introduction of pay for use models like SaaS, PaaS, and IaaS.

Historically in programming the costs we talk about are time, space, latency, bandwidth, storage, person hours, etc. Infrastructure costs have been part of the capital budget. Someone ponies up for the hardware and software is then "free" until more infrastructure is needed. The dollar cost of software design isn't usually an explicit factor considered.

Now software design decisions are part of the operations budget. Every algorithm decision you make will have dollar cost associated with it and it may become more important to craft algorithms that minimize operations cost across a large number of resources (CPU, disk, bandwidth, etc) than it is to trade off our old friends space and time.

Different resource costs will force very different design decisions. On Amazon do you use a spot instance, a reserved instance, or an on demand instance? Do you need a small or extra large or one of another dozen instance choices? Do you need to span multiple regions are is working across multiple availability zones acceptable? Should you build your own or used a built-in SaaS? Should you risk lock-in and use more of the built-in services are try to keep as independent as possible?

Just a few short years ago these are all issues you would never have considered before. A phase change has happened in architecture. Even if

you aren't in a public cloud it's likely you'll conceptualize your architecture in this way because that's how the infrastructure tools will be patterned.

Flow Architectures - The Firehose

One of the consequences of using a Service Oriented Architecture is a lot of messages need to be targeted to a lot of different endpoints. And because in the cloud you aren’t standing up a few servers and nailing down connections between them anymore, you need a robust message bus to connect everything together.

The solution that has evolved is the Firehose. A firehose is a message bus that can handle elastic components, message queueing, fault isolation, asynchronous processing, low latency communication, and operating at a high scale.

Here are a few examples of startups using firehose architectures:

- Tumblr: Internally applications need access to the activity stream of information about users creating/deleting posts, liking/unliking posts, etc. A challenge is to distribute so much data in real-time. An internal firehose was created as a message bus. Services and applications talk to the firehose via Thrift. LinkedIn’s Kafka is used to store messages. Internally consumers use an HTTP stream to read from the firehose. The firehose model is very flexible, not like Twitter’s firehose in which data is assumed to be lost. The firehose stream can be rewound in time and it retains a week of data. On connection it’s possible to specify the point in time to start reading. Multiple clients can connect and each client won’t see duplicate data. Each consumer in a consumer group gets its own messages and won’t see duplicates.

- DataSift: Created an Internet scale filtering system that can quickly evaluate very large filters. It is essentially a giant firehose. 0mq is used for replication, message broadcasting, and round-robin workload distribution. Kafka (LinkedIN's persistent and distributed message queue) is used for high-performance persistent queues.

An interesting architecture evolution we are seeing in the cloud is how systems continually reorganize themselves to give components better access to information flows. This allows services to be isolated yet still have access to all the information they need to carry out their specialized function. Before firehose style architectures the easiest path was to create monolithic applications because information was accessible only in one place. Now that information can flow freely and reliably between services, much more sophisticated architectures are possible.

Cell Architectures

Another consequence of Service Oriented Architectures is providing services at scale. The architecture that has evolved to satisfy these requirements is a little known technique called the Cell Architecture.

A Cell Architecture is based on the idea that massive scale requires parallelization and parallelization requires components be isolated from each other. These islands of isolation are called cells. A cell is a self-contained installation that can satisfy all the operations for a shard. A shard is a subset of a much larger dataset, typically a range of users, for example.

Cell Architectures have several advantages:

- Cells provide a unit of parallelization that can be adjusted to any size as the user base grows.

- Cell are added in an incremental fashion as more capacity is required.
- Cells isolate failures. One cell failure does not impact other cells.
- Cells provide isolation as the storage and application horsepower to process requests is independent of other cells.
- Cells enable nice capabilities like the ability to test upgrades, implement rolling upgrades, and test different versions of software.

- Cells can fail, be upgraded, and distributed across datacenters independent of other cells.

A number of startups make use of Cell Architectures:

- Tumblr: Users are mapped into cells and many cells exist per data center. Each cell has an HBase cluster, service cluster, and Redis caching cluster. Users are homed to a cell and all cells consume all posts via firehose updates. Background tasks consume from the firehose to populate tables and process requests. Each cell stores a single copy of all posts.
- Flickr: Uses a federated approach where all a user's data is stored on a shard which is a cluster of different services.
- Facebook: The Messages service has as the basic building block of their system a cluster of machines and services called a cell. A cell consists of ZooKeeper controllers, an application server cluster, and a metadata store.
- Salesforce: Salesforce is architected in terms of pods. Pods are self-contained sets of functionality consisting of 50 nodes, Oracle RAC servers, and Java application servers. Each pod supports many thousands of customers. If a pod fails only the users on that pod are impacted.

While the internal structure of a cell can be quite complex, the programmability of the cloud makes it relatively easy to configure, start, stop, failover and respond elastically to load.

Conclusion

We are still figuring out the New System of the World for IT. What was strange just a few years ago is now commonplace. Many discoveries and innovations wait to be made, it will never be complete, but the path has been set.
