Architecting for the Cloud: Best Practices

January 2011

Jinesh Varia
jvaria@amazon.com

Introduction

For several years, software architects have discovered and implemented several concepts and best practices to build highly scalable applications. In today's "era of tera", these concepts are even more applicable because of ever-growing datasets, unpredictable traffic patterns, and the demand for faster response times. This paper will reinforce and reiterate some of these traditional concepts and discuss how they may evolve in the context of cloud computing. It will also discuss some unprecedented concepts such as elasticity that have emerged due to the dynamic nature of the cloud.

This paper is targeted towards *cloud architects* who are gearing up to move an enterprise-class application from a fixed physical environment to a virtualized cloud environment. The focus of this paper is to highlight concepts, principles and best practices in creating new *cloud applications* or migrating existing applications to the cloud.

Background

As a cloud architect, it is important to understand the benefits of cloud computing. In this section, you will learn some of the business and technical benefits of cloud computing and different AWS services available today.

Business Benefits of Cloud Computing

There are some clear business benefits to building applications in the cloud. A few of these are listed here:

Almost zero upfront infrastructure investment: If you have to build a large-scale system it may cost a fortune to invest in real estate, physical security, hardware (racks, servers, routers, backup power supplies), hardware management (power management, cooling), and operations personnel. Because of the high upfront costs, the project would typically require several rounds of management approvals before the project could even get started. Now, with utility-style cloud computing, there is no fixed cost or startup cost.

Just-in-time Infrastructure: In the past, if your application became popular and your systems or your infrastructure did not scale you became a victim of your own success. Conversely, if you invested heavily and did not get popular, you became a victim of your failure. By deploying applications in-the-cloud with just-in-time self-provisioning, you do not have to worry about pre-procuring capacity for large-scale systems. This increases agility, lowers risk and lowers operational cost because you scale only as you *grow* and only pay for what you use.

More efficient resource utilization: System administrators usually worry about procuring hardware (when they run out of capacity) and higher infrastructure utilization (when they have excess and idle capacity). With the cloud, they can manage resources more effectively and efficiently by having the applications request and relinquish resources on-demand.

Usage-based costing: With utility-style pricing, you are billed only for the infrastructure that has been used. You are not paying for allocated but unused infrastructure. This adds a new dimension to cost savings. You can see immediate cost savings (sometimes as early as your next month's bill) when you deploy an optimization patch to update your cloud application. For example, if a caching layer can reduce your data requests by 70%, the savings begin to accrue immediately and you see the reward right in the next bill. Moreover, if you are building platforms on the top of the cloud, you can pass on the same flexible, variable usage-based cost structure to your own customers.

Reduced time to market: Parallelization is the one of the great ways to speed up processing. If one compute-intensive or data-intensive job that can be run in parallel takes 500 hours to process on one machine, with cloud architectures [6], it would be possible to spawn and launch 500 instances and process the same job in 1 hour. Having available an elastic infrastructure provides the application with the ability to exploit parallelization in a cost-effective manner reducing time to market.

Technical Benefits of Cloud Computing

Some of the technical benefits of cloud computing includes:

Automation – "Scriptable infrastructure": You can create repeatable build and deployment systems by leveraging programmable (API-driven) infrastructure.

Auto-scaling: You can scale your applications up and down to match your unexpected demand without any human intervention. Auto-scaling encourages automation and drives more efficiency.

Proactive Scaling: Scale your application up and down to meet your anticipated demand with proper planning understanding of your traffic patterns so that you keep your costs low while scaling.

More Efficient Development lifecycle: Production systems may be easily cloned for use as development and test environments. Staging environments may be easily promoted to production.

Improved Testability: Never run out of hardware for testing. Inject and automate testing at every stage during the development process. You can spawn up an "instant test lab" with pre-configured environments only for the duration of testing phase.

Disaster Recovery and Business Continuity: The cloud provides a lower cost option for maintaining a fleet of DR servers and data storage. With the cloud, you can take advantage of geo-distribution and replicate the environment in other location within minutes.

"Overflow" the traffic to the cloud: With a few clicks and effective load balancing tactics, you can create a complete overflow-proof application by routing excess traffic to the cloud.

Understanding the Amazon Web Services Cloud

The Amazon Web Services (AWS) cloud provides a highly reliable and scalable infrastructure for deploying web-scale solutions, with minimal support and administration costs, and more flexibility than you've come to expect from your own infrastructure, either on-premise or at a datacenter facility.

AWS offers variety of infrastructure services today. The diagram below will introduce you the AWS terminology and help you understand how your application can interact with different Amazon Web Services and how different services interact with each other.

Amazon Elastic Compute Cloud (Amazon EC2)[1] is a web service that provides resizable compute capacity in the cloud. You can bundle the operating system, application software and associated configuration settings into an *Amazon Machine Image* (AMI). You can then use these AMIs to provision multiple virtualized *instances* as well as decommission them using simple web service calls to scale capacity up and down quickly, as your capacity requirement changes. You can purchase *On-Demand Instances* in which you pay for the instances by the hour or *Reserved Instances* in which you pay a low, one-time payment and receive a lower usage rate to run the instance than with an On-Demand Instance or *Spot Instances* where you can bid for unused capacity and further reduce your cost. Instances can be launched in one or more geographical *regions*. Each region has multiple *Availability Zones*. Availability Zones are distinct locations that are engineered to be insulated from failures in other Availability Zones and provide inexpensive, low latency network connectivity to other Availability Zones in the same Region.

[1] More info about Amazon EC2 is available at http://aws.amazon.com/ec2

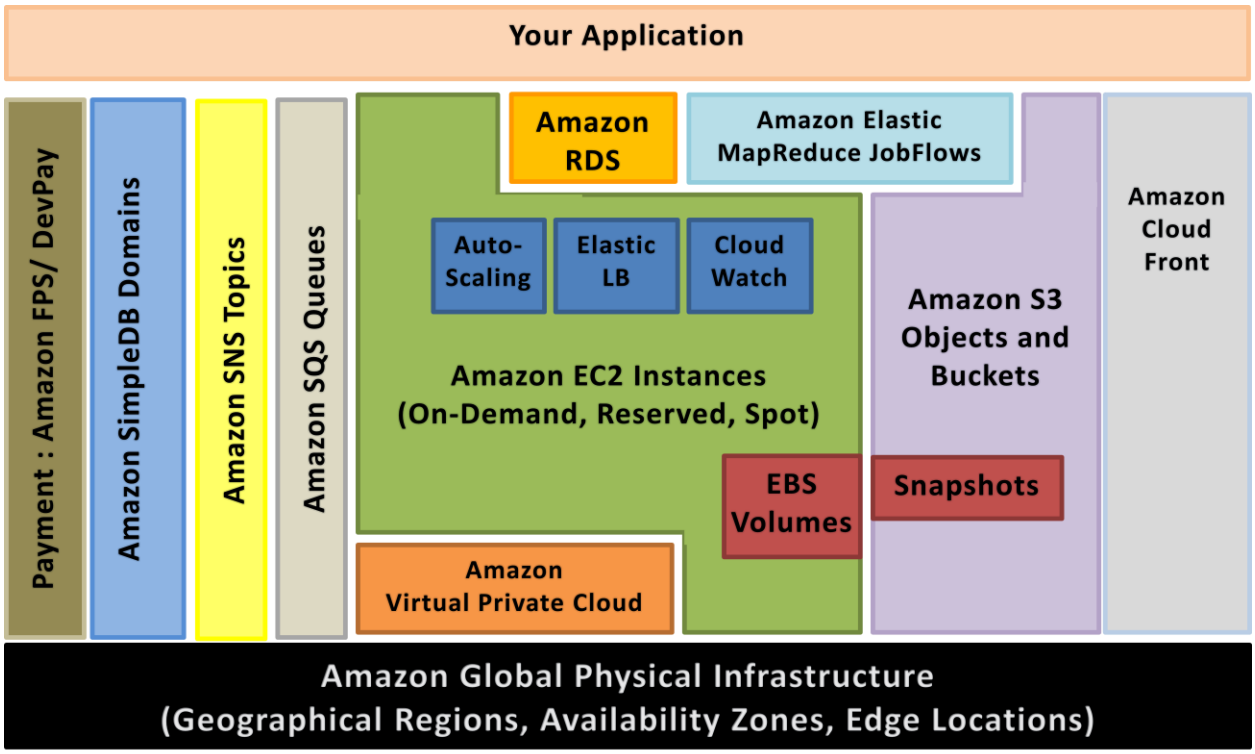

Figure 1: Amazon Web Services

Elastic IP addresses allow you to allocate a static IP address and programmatically assign it to an instance. You can enable monitoring on an Amazon EC2 instance using Amazon CloudWatch[2] in order to gain visibility into resource utilization, operational performance, and overall demand patterns (including metrics such as CPU utilization, disk reads and writes, and network traffic). You can create *Auto-scaling Group* using the Auto-scaling feature[3] to automatically scale your capacity on certain conditions based on metric that Amazon CloudWatch collects. You can also distribute incoming traffic by creating an *elastic load balancer* using the Elastic Load Balancing[4] service. Amazon Elastic Block Storage (EBS)[5] *volumes* provide network-attached persistent storage to Amazon EC2 instances. Point-in-time consistent *snapshots* of EBS volumes can be created and stored on Amazon Simple Storage Service (Amazon S3)[6].

Amazon S3 is highly durable and distributed data store. With a simple web services interface, you can store and retrieve large amounts of data as *objects* in *buckets* (containers) at any time, from anywhere on the web using standard HTTP verbs. Copies of objects can be distributed and cached at 14 *edge locations* around the world by creating a *distribution* using Amazon CloudFront[7] service – a web service for content delivery (static or streaming content). Amazon SimpleDB[8] is a web service that provides the core functionality of a database- real-time lookup and simple querying of structured data - without the operational complexity. You can organize the dataset into *domains* and can run queries across all of the data stored in a particular domain. Domains are collections of *items* that are described by *attribute-value pairs*.

---

[2] More info about Amazon CloudWatch is available at http://aws.amazon.com/cloudwatch/
[3] More info about Auto-scaling feature is available at http://aws.amazon.com/auto-scaling
[4] More info about Elastic Load Balancing feature is available at http://aws.amazon.com/elasticloadbalancing
[5] More info about Elastic Block Store is available at http://aws.amazon.com/ebs
[6] More info about Amazon S3 is available at http://aws.amazon.com/s3
[7] More info about Amazon CloudFront is available at http://aws.amazon.com/cloudfront
[8] More info about Amazon SimpleDB is available at http://aws.amazon.com/simpledb

Amazon Relational Database Service[9] (Amazon RDS) provides an easy way to setup, operate and scale a relational database in the cloud. You can launch a *DB Instance* and get access to a full-featured MySQL database and not worry about common database administration tasks like backups, patch management etc.

Amazon Simple Queue Service (Amazon SQS)[10] is a reliable, highly scalable, hosted distributed queue for storing *messages* as they travel between computers and application components.

Amazon Simple Notifications Service (Amazon SNS) [11]provides a simple way to notify applications or people from the cloud by creating *Topics* and using a publish-subscribe protocol.

Amazon Elastic MapReduce[12] provides a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) and allows you to create customized *JobFlows*. JobFlow is a sequence of MapReduce *steps.*

Amazon Virtual Private Cloud (Amazon VPC)[13] allows you to extend your corporate network into a private cloud contained within AWS. Amazon VPC uses IPSec tunnel mode that enables you to create a secure connection between a gateway in your data center and a gateway in AWS.

Amazon Route53 is a highly scalable DNS service that allows you manage your DNS records by creating a *HostedZone* for every domain you would like to manage.

AWS Identity and Access Management (IAM)[14] enable you to create multiple Users with unique security credentials and manage the permissions for each of these Users within your AWS Account. IAM is natively integrated into AWS Services. No service APIs have changed to support IAM, and exiting applications and tools built on top of the AWS service APIs will continue to work when using IAM.

AWS also offers various payment and billing services[15] that leverages Amazon's payment infrastructure.

All AWS infrastructure services offer utility-style pricing that require no long-term commitments or contracts. For example, you pay by the hour for Amazon EC2 instance usage and pay by the gigabyte for storage and data transfer in the case of Amazon S3. More information about each of these services and their pay-as-you-go pricing is available on the AWS Website.

Note that using the AWS cloud doesn't require sacrificing the flexibility and control you've grown accustomed to:

- You are free to use the programming model, language, or operating system (Windows, OpenSolaris or any flavor of Linux) of your choice.
- You are free to pick and choose the AWS products that best satisfy your requirements—you can use any of the services individually or in any combination.

---

[9] More info about Amazon RDS is available at http://aws.amazon.com/rds
[10] More info about Amazon SQS is available at http://aws.amazon.com/sqs and
[11] More info about Amazon SNS is available at http://aws.amazon.com/sns
[12] More info about Amazon ElasticMapReduce is available at http://aws.amazon.com/elasticmapreduce
[13] More info about Amazon Virtual Private Cloud is available at http://aws.amazon.com/vpc
[14] More info about Amazon Identity and Access Management is available at http://aws.amazon.com/iam
[15] More info at Amazon Flexible Payments Service is available at http://aws.amazon.com/fps and Amazon DevPay is available at http://aws.amazon.com/devpay

- Because AWS provides resizable (storage, bandwidth and computing) resources, you are free to consume as much or as little and only pay for what you consume.
- You are free to use the system management tools you've used in the past and extend your datacenter into the cloud.

Cloud Concepts

The cloud reinforces some old concepts of building highly scalable Internet architectures [13] and introduces some new concepts that entirely change the way applications are built and deployed. Hence, when you progress from concept to implementation, you might get the feeling that "Everything's changed, yet nothing's different." The cloud changes several processes, patterns, practices, philosophies and reinforces some traditional service-oriented architectural principles that you have learnt as they are even more important than before. In this section, you will see some of those new cloud concepts and reiterated SOA concepts.

Traditional applications were built with some pre-conceived mindsets that made economic and architectural-sense at the time they were developed. The cloud brings some new philosophies that you need to understand and are discussed below:

Building Scalable Architectures

It is critical to build a scalable architecture in order to take advantage of a scalable infrastructure.

The cloud is designed to provide conceptually infinite scalability. However, you cannot leverage all that scalability in infrastructure if your architecture is not scalable. Both have to work together. You will have to identify the monolithic components and bottlenecks in your architecture, identify the areas where you cannot leverage the on-demand provisioning capabilities in your architecture and work to *refactor* your application in order to leverage the scalable infrastructure and take advantage of the cloud.

Characteristics of a truly scalable application:

- Increasing resources results in a proportional increase in performance
- A scalable service is capable of handling heterogeneity
- A scalable service is operationally efficient
- A scalable service is resilient
- A scalable service should become more cost effective when it grows (Cost per unit reduces as the number of units increases)

These are things that should become an inherent part of your application and if you design your architecture with the above characteristics in mind, then both your architecture and infrastructure will work together to give you the scalability you are looking for.

Understanding Elasticity

The graph below illustrates the different approaches a cloud architect can take to scale their applications to meet the demand.

Scale-up approach: not worrying about the scalable application architecture and investing heavily in larger and more powerful computers (vertical scaling) to accommodate the demand. This approach usually works to a point, but could either cost a fortune (See "Huge capital expenditure" in diagram) or the demand could out-grow capacity before the new "big iron" is deployed (See "You just lost your customers" in diagram).

The traditional scale-out approach: creating an architecture that scales horizontally and investing in infrastructure in small chunks. Most of the businesses and large-scale web applications follow this pattern by distributing their application components, federating their datasets and employing a service-oriented design. This approach is often more effective than a scale up approach. However, this still requires predicting the demand at regular intervals and then deploying infrastructure in chunks to meet the demand. This often leads to excess capacity ("burning cash") and constant manual monitoring ("burning human cycles"). Moreover, it usually does not work if the application is a victim of a viral fire (often referred to as the Slashdot Effect[16]).

Note: Both approaches have initial start-up costs and both approaches are reactive in nature.

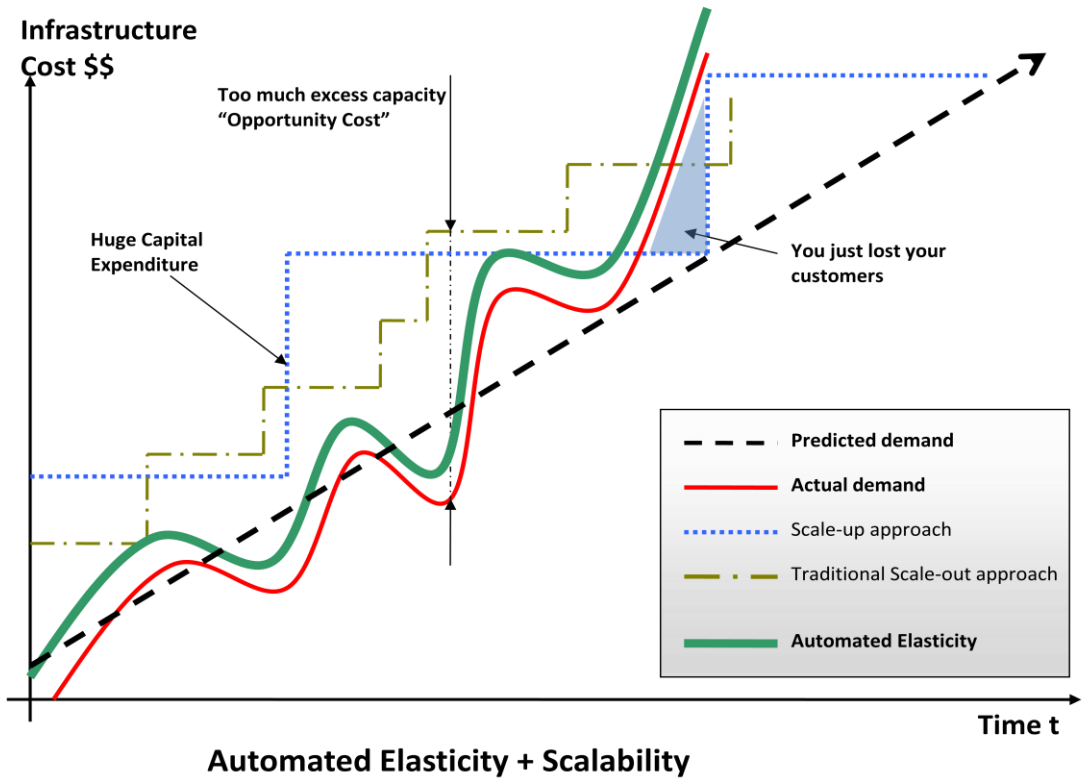

Figure 2: Automated Elasticity

[16] http://en.wikipedia.org/wiki/Slashdot_effect

Traditional infrastructure generally necessitates predicting the amount of computing resources your application will use over a period of several years. If you under-estimate, your applications will not have the horsepower to handle unexpected traffic, potentially resulting in customer dissatisfaction. If you over-estimate, you're wasting money with superfluous resources.

The on-demand and elastic nature of *the cloud approach* (Automated Elasticity), however, enables the infrastructure to be closely aligned (as it expands and contracts) with the actual demand, thereby increasing overall utilization and reducing cost.

Elasticity is one of the fundamental properties of the cloud. Elasticity is the power to scale computing resources up and down easily and with minimal friction. It is important to understand that elasticity will ultimately drive most of the benefits of the cloud. As a cloud architect, you need to internalize this concept and work it into your application architecture in order to take maximum benefit of the cloud.

Traditionally, applications have been built for fixed, rigid and pre-provisioned infrastructure. Companies never had the need to provision and install servers on daily basis. As a result, most software architectures do not address the rapid deployment or reduction of hardware. Since the provisioning time and upfront investment for acquiring new resources was too high, software architects never invested time and resources in optimizing for hardware utilization. It was acceptable if the hardware on which the application is running was under-utilized. The notion of "elasticity" within an architecture was overlooked because the idea of having new resources in minutes was not possible.

With the cloud, this mindset needs to change. Cloud computing streamlines the process of acquiring the necessary resources; there is no longer any need to place orders ahead of time and to hold unused hardware captive. Instead, cloud architects can request what they need mere minutes before they need it or automate the procurement process, taking advantage of the vast scale and rapid response time of the cloud. The same is applicable to releasing the unneeded or under-utilized resources when you don't need them.

If you cannot embrace the change and implement elasticity in your application architecture, you might not be able to take the full advantage of the cloud. As a cloud architect, you should think creatively and think about ways you can implement elasticity in your application. For example, infrastructure that used to run daily nightly builds and perform regression and unit tests every night at 2:00 AM for two hours (often termed as the "QA/Build box") was sitting idle for rest of the day. Now, with elastic infrastructure, one can run nightly builds on boxes that are "alive" and being paid for only for 2 hours in the night. Likewise, an internal trouble ticketing web application that always used to run on peak capacity (5 servers 24x7x365) to meet the demand during the day can now be provisioned to run on-demand (5 servers from 9AM to 5 PM and 2 servers for 5 PM to 9 AM) based on the traffic pattern.

Designing intelligent elastic cloud architectures, so that infrastructure runs only when you need it, is an art in itself. Elasticity should be one of the architectural design requirements or a system property. Question that you need to ask: What components or layers in my application architecture can become elastic? What will it take to make that component *elastic*? What will be the impact of implementing elasticity to my overall system architecture?

In the next section, you will see specific techniques to implement elasticity in your applications. To effectively leverage the cloud benefits, it is important to architect with this mindset.

Not fearing constraints

When you decide to move your applications to the cloud and try to map your system specifications to those available in the cloud, you will notice that cloud might not have the exact specification of the resource that you have on-premise. For example, "Cloud does not provide X amount of RAM in a server" or "My Database needs to have more IOPS than what I can get in a single instance".

You should understand that cloud provides *abstract resources* and they become powerful when you combine them with the on-demand provisioning model. You should not be afraid and constrained when using cloud resources because it is important to understand that even if you might not get an exact replica of your hardware in the cloud environment, you have the ability to get more of those resources in the cloud to compensate that need.

For example, if the cloud does not provide you with exact or greater amount of RAM in a server, try using a distributed cache like *memcached*[17] or partitioning your data across multiple servers. If your databases need more IOPS and it does not directly map to that of the cloud, there are several recommendations that you can choose from depending on your type of data and use case. If it is a read-heavy application, you can distribute the read load across a fleet of synchronized slaves. Alternatively, you can use a *sharding* [10] algorithm that routes the data where it needs to be or you can use various database clustering solutions.

In retrospect, when you combine the on-demand provisioning capabilities with the flexibility, you will realize that apparent constraints can actually be broken in ways that will actually improve the scalability and overall performance of the system.

Virtual Administration

The advent of cloud has changed the role of System Administrator to a "Virtual System Administrator". This simply means that daily tasks performed by these administrators have now become even more interesting as they learn more about applications and decide what's best for the business as a whole. The System Administrator no longer has a need to provision servers and install software and wire up network devices since all of that grunt work is replaced by few clicks and command line calls. The cloud encourages automation because the infrastructure is programmable. System administrators need to move up the technology stack and learn how to manage abstract cloud resources using scripts.

Likewise, the role of Database Administrator is changed into a "Virtual Database Administrator" in which he/she manages resources through a web-based console, executes scripts that add new capacity programmatically in case the database hardware runs out of capacity and automates the day-to-day processes. The virtual DBA has to now learn new deployment methods (virtual machine images), embrace new models (query parallelization, geo-redundancy and asynchronous replication [11]), rethink the architectural approach for data (sharding [9], horizontal partitioning [13], federating [14]) and leverage different storage options available in the cloud for different types of datasets.

In the traditional enterprise company, application developers may not work closely with the network administrators and network administrators may not have a clue about the application. As a result, several possible optimizations in the network layer and application architecture layer are overlooked. With the cloud, the two roles have merged into one to some extent. When architecting future applications, companies need to encourage more cross-pollination of knowledge between the two roles and understand that they are merging.

[17] http://www.danga.com/memcached/

Cloud Best Practices

In this section, you will learn about best practices that will help you build an application in the cloud.

Design for failure and nothing will fail

Rule of thumb: Be a pessimist when designing architectures in the cloud; assume things will fail. In other words, always design, implement and deploy for automated recovery from failure.

In particular, assume that your hardware *will* fail. Assume that outages *will* occur. Assume that some disaster *will* strike your application. Assume that you *will* be slammed with more than the expected number of requests per second some day. Assume that with time your application software will fail too. By being a pessimist, you end up thinking about recovery strategies during design time, which helps in designing an overall system better.

If you realize that things fail over time and incorporate that thinking into your architecture, build mechanisms to handle that failure before disaster strikes to deal with a scalable infrastructure, you will end up creating a fault-tolerant architecture that is optimized for the cloud.

Questions that you need to ask: What happens if a node in your system fails? How do you recognize that failure? How do I replace that node? What kind of scenarios do I have to plan for? What are my single points of failure? If a load balancer is sitting in front of an array of application servers, what if that load balancer fails? If there are master and slaves in your architecture, what if the master node fails? How does the failover occur and how is a new slave instantiated and brought into sync with the master?

Just like designing for hardware failure, you have to also design for software failure. Questions that you need to ask: What happens to my application if the dependent services changes its interface? What if downstream service times out or returns an exception? What if the cache keys grow beyond memory limit of an instance?

Build mechanisms to handle that failure. For example, the following strategies can help in event of failure:

1. Have a coherent backup and restore strategy for your data and automate it
2. Build process threads that resume on reboot
3. Allow the state of the system to re-sync by reloading messages from queues
4. Keep pre-configured and pre-optimized virtual images to support (2) and (3) on launch/boot
5. Avoid in-memory sessions or stateful user context, move that to data stores.

Good cloud architectures should be impervious to reboots and re-launches. In GrepTheWeb (discussed in the Cloud Architectures paper [6]), by using a combination of Amazon SQS and Amazon SimpleDB, the overall controller architecture is very resilient to the types of failures listed in this section. For instance, if the instance on which controller thread was running dies, it can be brought up and resume the previous state as if nothing had happened. This was accomplished by creating a pre-configured Amazon Machine Image, which when launched dequeues all the messages from the Amazon SQS queue and reads their states from an Amazon SimpleDB domain on reboot.

Designing with an assumption that underlying hardware will fail, will prepare you for the future when it actually fails.

This design principle will help you design operations-friendly applications, as also highlighted in Hamilton's paper [11]. If you can extend this principle to pro-actively measure and balance load dynamically, you might be able to deal with variance in network and disk performance that exists due to multi-tenant nature of the cloud.

AWS specific tactics for implementing this best practice:

1. Failover gracefully using Elastic IPs: Elastic IP is a static IP that is dynamically re-mappable. You can quickly remap and failover to another set of servers so that your traffic is routed to the new servers. It works great when you want to upgrade from old to new versions or in case of hardware failures
2. Utilize multiple Availability Zones: Availability Zones are conceptually like logical datacenters. By deploying your architecture to multiple availability zones, you can ensure highly availability. Utilize Amazon RDS Multi-AZ [21] deployment functionality to automatically replicate database updates across multiple Availability Zones.
3. Maintain an Amazon Machine Image so that you can restore and clone environments very easily in a different Availability Zone; Maintain multiple Database slaves across Availability Zones and setup hot replication.
4. Utilize Amazon CloudWatch (or various real-time open source monitoring tools) to get more visibility and take appropriate actions in case of hardware failure or performance degradation. Setup an Auto scaling group to maintain a fixed fleet size so that it replaces unhealthy Amazon EC2 instances by new ones.
5. Utilize Amazon EBS and set up cron jobs so that incremental snapshots are automatically uploaded to Amazon S3 and data is persisted independent of your instances.
6. Utilize Amazon RDS and set the retention period for backups, so that it can perform automated backups.

Decouple your components

The cloud reinforces the SOA design principle that *the more loosely coupled the components of the system, the bigger and better it scales*.

The key is to build components that do not have tight dependencies on each other, so that if one component were to die (fail), sleep (not respond) or remain busy (slow to respond) for some reason, the other components in the system are built so as to continue to work as if no failure is happening. In essence, loose coupling isolates the various layers and components of your application so that each component interacts asynchronously with the others and treats them as a "black box". For example, in the case of web application architecture, you can isolate the app server from the web server and from the database. The app server does not know about your web server and vice versa, this gives decoupling between these layers and there are no dependencies code-wise or functional perspectives. In the case of batch-processing architecture, you can create *asynchronous* components that are independent of each other.

Questions you need to ask: Which business component or feature could be isolated from current monolithic application and can run standalone separately? And then how can I add more instances of that component without breaking my current system and at the same time serve more users? How much effort will it take to encapsulate the component so that it can interact with other components asynchronously?

Decoupling your components, building *asynchronous* systems and scaling horizontally become very important in the context of the cloud. It will not only allow you to scale out by adding more instances of same component but also allow you to design innovative hybrid models in which a few components continue to run in on-premise while other components can take advantage of the cloud-scale and use the cloud for additional compute-power and bandwidth. That way with minimal effort, you can "overflow" excess traffic to the cloud by implementing smart load balancing tactics.

One can build a loosely coupled system using *messaging queues*. If a queue/buffer is used to connect any two components together, it can support concurrency, high availability and load spikes. As a result, the overall system continues to perform even if parts of components are momentarily unavailable. If one component dies or becomes temporarily unavailable, the system will buffer the messages and get them processed when the component comes back up.

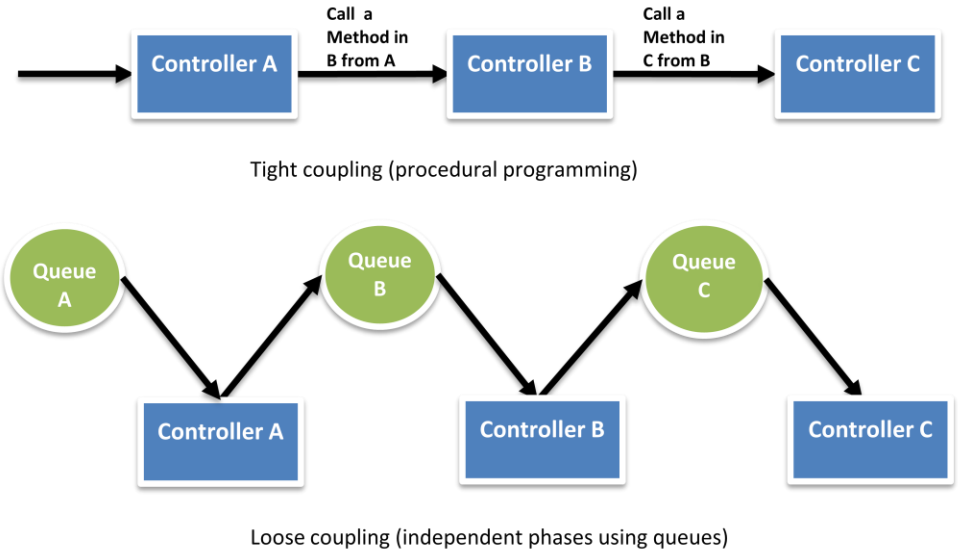

Figure 3: decoupling components using Queues

You will see heavy use of queues in GrepTheWeb Architecture epitomized in the Cloud Architectures paper [6]. In GrepTheWeb, if lots of requests suddenly reach the server (an Internet-induced overload situation) or the processing of regular expressions takes a longer time than the median (slow response rate of a component), the Amazon SQS queues buffer the requests in a durable fashion so that those delays do not affect other components.

AWS specific tactics for implementing this best practice:

1. Use Amazon SQS to isolate components [18]
2. Use Amazon SQS as buffers between components [18]
3. Design every component such that it expose a service interface and is responsible for its own scalability in all appropriate dimensions and interacts with other components asynchronously
4. Bundle the logical construct of a component into an Amazon Machine Image so that it can be deployed more often
5. Make your applications as stateless as possible. Store session state outside of component (in Amazon SimpleDB, if appropriate)

Implement elasticity

The cloud brings a new concept of elasticity in your applications. Elasticity can be implemented in three ways:

1. Proactive Cyclic Scaling: Periodic scaling that occurs at fixed interval (daily, weekly, monthly, quarterly)
2. Proactive Event-based Scaling: Scaling just when you are expecting a big surge of traffic requests due to a scheduled business event (new product launch, marketing campaigns)
3. Auto-scaling based on demand. By using a monitoring service, your system can send triggers to take appropriate actions so that it scales up or down based on metrics (utilization of the servers or network i/o, for instance)

To implement "Elasticity", one has to first automate the deployment process and streamline the configuration and build process. This will ensure that the system can scale without any human intervention.

This will result in immediate cost benefits as the overall utilization is increased by ensuring your resources are closely aligned with demand rather than potentially running servers that are under-utilized.

Automate your infrastructure

One of the most important benefits of using a cloud environment is the ability to use the cloud's APIs to automate your deployment process. It is recommended that you take the time to create an automated deployment process early on during the migration process and not wait till the end. Creating an automated and repeatable deployment process will help reduce errors and facilitate an efficient and scalable update process.

To automate the deployment process:

- Create a library of "recipes" – small frequently-used scripts (for installation and configuration)
- Manage the configuration and deployment process using agents bundled inside an AMI
- Bootstrap your instances

Bootstrap your instances

Let your instances ask you a question at boot "who am I and what is my role?" Every Instance should have a role ("DB server", "app server", "slave server" in the case of a Web application) to play in the environment. This role may be passed in as an argument during launch that instructs the AMI when instantiated the steps to take after it has booted. On boot, instances should grab the necessary resources (code, scripts, configuration) based on the role and "attach" itself to a cluster to serve its function. Benefits of bootstrapping your instances:

1. Recreate the (Dev, staging, Production) environment with few clicks and minimal effort
2. More control over your abstract cloud-based resources
3. Reduce human-induced deployment errors
4. Create a Self Healing and Self-discoverable environment which is more resilient to hardware failure

AWS specific tactics to automate your infrastructure

1. Define Auto-scaling groups for different clusters using the Amazon Auto-scaling feature in Amazon EC2.
2. Monitor your system metrics (CPU, Memory, Disk I/O, Network I/O) using Amazon CloudWatch and take appropriate actions (launching new AMIs dynamically using the Auto-scaling service) or send notifications.
3. Store and retrieve machine configuration information dynamically: Utilize Amazon SimpleDB to fetch

config data during boot-time of an instance (eg. database connection strings). SimpleDB may also be used to store information about an instance such as its IP address, machine name and role.

4. Design a build process such that it dumps the latest builds to a bucket in Amazon S3; download the latest version of an application from during system startup.
5. Invest in building resource management tools (Automated scripts, pre-configured images) or Use smart open source configuration management tools like Chef[18], Puppet[19], CFEngine [20]or Genome[21].
6. Bundle Just Enough Operating System (JeOS[22]) and your software dependencies into an Amazon Machine Image so that it is easier to manage and maintain. Pass configuration files or parameters at launch time and retrieve user data[23] and instance metadata after launch.
7. Reduce bundling and launch time by booting from Amazon EBS volumes[24] and attaching multiple Amazon EBS volumes to an instance. Create snapshots of common volumes and share snapshots[25] among accounts wherever appropriate.
8. Application components should not assume health or location of hardware it is running on. For example, dynamically attach the IP address of a new node to the cluster. Automatically failover and start a new clone in case of a failure.

Think parallel

The cloud makes parallelization effortless. Whether it is requesting data from the cloud, storing data to the cloud, processing data (or executing jobs) in the cloud, as a cloud architect, you need to internalize the concept of parallelization when designing architectures in the cloud. It is advisable to not only implement parallelization wherever possible but also automate it because the cloud allows you to create a repeatable process every easily.

When it comes to accessing (retrieving and storing) data, the cloud is designed to handle massively parallel operations. In order to achieve maximum performance and throughput, you should leverage *request parallelization*. Multi-threading your requests by using multiple concurrent threads will store or fetch the data faster than requesting it sequentially. Hence, wherever possible, the processes of a cloud application should be made thread-safe through a share-nothing philosophy and leverage multi-threading.

When it comes to processing or executing requests in the cloud, it becomes even more important to leverage parallelization. A general best practice, in the case of a web application, is to distribute the incoming requests across multiple asynchronous web servers using load balancer. In the case of batch processing application, you can master node can spawn up multiple slave worker nodes that processes task in parallel (as in distributed processing frameworks like Hadoop[26])

[18] More info about Chef can be found at http://wiki.opscode.com/display/chef/Home
[19] More info about Puppet can be found at http://reductivelabs.com/trac/puppet/
[20] More info about CFEngine can be found at http://www.cfengine.org/
[21] More info about Genome can be found at http://genome.et.redhat.com/
[22] http://en.wikipedia.org/wiki/Just_enough_operating_system
[23] Instance metadata and userdata info can be found at http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/index.html?AESDG-chapter-instancedata.html
[24] More info about Boot From Amazon EBS feature can be found at http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3121
[25] See How to Share a Snapshot at http://aws.amazon.com/ebs/
[26] http://hadoop.apache.org/

The beauty of the cloud shines when you combine elasticity and parallelization. Your cloud application can bring up a cluster of compute instances which are provisioned within minutes with just a few API calls, perform a job by executing tasks in parallel, store the results and terminate all the instances. The GrepTheWeb application discussed in [6] is one such example.

AWS specific tactics for parallelization:

1. Multi-thread your Amazon S3 requests as detailed in Best practices paper [2]
2. Multi-thread your Amazon SimpleDB GET and BATCHPUT requests [3][4] [5]
3. Create a JobFlow using the Amazon Elastic MapReduce Service for each of your daily batch processes (indexing, log analysis etc.) which will compute the job in parallel and save time.
4. Use the Elastic Load Balancing service and spread your load across multiple web app servers *dynamically*

Keep dynamic data closer to the compute and static data closer to the end-user

In general it's a good practice to keep your data as close as possible to your compute or processing elements to reduce latency. In the cloud, this best practice is even more relevant and important because you often have to deal with Internet latencies. Moreover, in the cloud, you are paying for bandwidth in and out of the cloud by the gigabyte of data transfer and the cost can add up very quickly.

If a large quantity of data that needs to be processed resides outside of the cloud, it might be cheaper and faster to "ship" and transfer the data to the cloud first and then perform the computation. For example, in the case of a data warehousing application, it is advisable to move the dataset to the cloud and then perform parallel queries against the dataset. In the case of web applications that store and retrieve data from relational databases, it is advisable to move the database as well as the app server into the cloud all at once.

If the data is generated in the cloud, then the applications that consume the data should also be deployed in the cloud so that they can take advantage of in-cloud free data transfer and lower latencies. For example, in the case of an e-commerce web application that generates logs and clickstream data, it is advisable to run the log analyzer and reporting engines in the cloud.

Conversely, if the data is static and not going to change often (for example, images, video, audio, PDFs, JS, CSS files), it is advisable to take advantage of a content delivery service so that the static data is cached at an edge location closer to the end-user (requester) thereby lowering the access latency. Due to the caching, a content delivery service provides faster access to popular objects.

AWS-specific tactics for implementing this best practice:

1. Ship your data drives to Amazon using the Import/Export service[27]. It may be cheaper and faster to move large amounts of data using the sneakernet[28] than to upload using the Internet.
2. Utilize the same Availability Zone to launch a cluster of machines
3. Create a distribution of your Amazon S3 bucket and let Amazon CloudFront caches content in that bucket across all the 14 edge locations around the world

[27] More info about Amazon Import Export Services can be found at http://aws.amazon.com/importexport
[28] http://en.wikipedia.org/wiki/Sneakernet

Security Best Practices

In a multi-tenant environment, cloud architects often express concerns about security. *Security should be implemented in every layer of the cloud application architecture.* Physical security is typically handled by your service provider (Security Whitepaper [7]), which is an additional benefit of using the cloud. Network and application-level security is your responsibility and you should implement the best practices as applicable to your business. In this section, you will learn about some specific tools, features and guidelines on how to secure your cloud application in the AWS environment. It is recommended to take advantage of these tools and features mentioned to implement basic security and then implement additional security best practices using standard methods as appropriate or as they see fit.

Protect your data in transit

If you need to exchange sensitive or confidential information between a browser and a web server, configure SSL on your server instance. You'll need a certificate from an external certification authority like VeriSign[29] or Entrust[30]. The public key included in the certificate authenticates your server to the browser and serves as the basis for creating the shared session key used to encrypt the data in both directions.

Create a Virtual Private Cloud by making a few command line calls (using Amazon VPC). This will enable you to use your own logically isolated resources within the AWS cloud, and then connect those resources directly to your own datacenter using industry-standard encrypted IPSec VPN connections.

You can also setup [15] an OpenVPN server on an Amazon EC2 instance and install the OpenVPN client on all user PCs.

Protect your data at rest

If you are concerned about storing sensitive and confidential data in the cloud, you should encrypt the data (individual files) before uploading it to the cloud. For example, encrypt the data using any open source[31] or commercial[32] PGP-based tools before storing it as Amazon S3 objects and decrypt it after download. This is often a good practice when building HIPAA-Compliant applications [8] that need to store Protected Health Information (PHI).

On Amazon EC2, file encryption depends on the operating system. Amazon EC2 instances running Windows can use the built-in Encrypting File System (EFS) feature [16]. This feature will handle the encryption and decryption of files and folders automatically and make the process transparent to the users [19]. However, despite its name, EFS doesn't encrypt the entire file system; instead, it encrypts individual files. If you need a full encrypted volume, consider using the open-source TrueCrypt[33] product; this will integrate very well with NTFS-formatted EBS volumes. Amazon EC2 instances running Linux can mount EBS volumes using encrypted file systems using variety of approaches (EncFS[34], Loop-AES[35], dm-crypt[36], TrueCrypt[37]). Likewise, Amazon EC2 instances running OpenSolaris can take advantage of ZFS[38] Encryption Support [20]. Regardless of which approach you choose, encrypting files and volumes in Amazon EC2 helps protect files

[29] http://www.verisign.com/ssl/
[30] http://www.entrust.net/ssl-products.htm
[31] http://www.gnupg.org
[32] http://www.pgp.com/
[33] http://www.truecrypt.org/
[34] http://www.arg0.net/encfs
[35] http://loop-aes.sourceforge.net/loop-AES.README
[36] http://www.saout.de/misc/dm-crypt/
[37] http://www.truecrypt.org/
[38] http://www.opensolaris.org/os/community/zfs/

and log data so that only the users and processes on the server can see the data in clear text, but anything or anyone outside the server see only encrypted data.

No matter which operating system or technology you choose, encrypting data at rest presents a challenge: managing the keys used to encrypt the data. If you lose the keys, you will lose your data forever and if your keys become compromised, the data may be at risk. Therefore, be sure to study the key management capabilities of any products you choose and establish a procedure that minimizes the risk of losing keys.

Besides protecting your data from eavesdropping, also consider how to protect it from disaster. Take periodic snapshots of Amazon EBS volumes to ensure it is highly durable and available. Snapshots are incremental in nature and stored on Amazon S3 (separate geo-location) and can be restored back with a few clicks or command line calls.

Protect your AWS credentials

AWS supplies two types of security credentials: AWS access keys and X.509 certificates. Your AWS access key has two parts: your *access key ID* and your *secret access key*. When using the REST or Query API, you have to use your secret access key to calculate a signature to include in your request for authentication. To prevent in-flight tampering, all requests should be sent over HTTPS.

If your Amazon Machine Image (AMI) is running processes that need to communicate with other AWS web services (for polling the Amazon SQS queue or for reading objects from Amazon S3, for example), one common design mistake is embedding the AWS credentials in the AMI. Instead of embedding the credentials, they should be passed in as arguments during launch and encrypted before being sent over the wire [17].

If your secret access key becomes compromised, you should obtain a new one by rotating[39] to a new access key ID. As a good practice, it is recommended that you incorporate a key rotation mechanism into your application architecture so that you can use it on a regular basis or occasionally (when disgruntled employee leaves the company) to ensure compromised keys can't last forever.

Alternately, you can use X.509 certificates for authentication to certain AWS services. The certificate file contains your public key in a base64-encoded DER certificate body. A separate file contains the corresponding base64-encoded PKCS#8 private key.

AWS supports multi-factor authentication[40] as an additional protector for working with your account information on aws.amazon.com and AWS Management Console[41].

Manage multiple Users and their permissions with IAM

AWS Identity and Access Management (IAM)[42] enables you to create multiple Users and manage the permissions for each of these Users within your AWS Account. A User is an identity (within your AWS Account) with unique security credentials that can be used to access AWS Services. IAM eliminates the need to share passwords or access keys, and makes it easy to enable or disable a User's access as appropriate.

IAM enables you to implement security best practices, such as least privilege, by granting unique credentials to every User within your AWS account and only grant permission to access the AWS Services and resources required for the

[39] http://aws.amazon.com/about-aws/whats-new/2009/08/31/seamlessly-rotate-your-access-credentials/
[40] More info about Multi-factor Authentication is available at http://aws.amazon.com/mfa/
[41] AWS Management Console http://aws.amazon.com/console/
[42] More Info at http://aws.amazon.com.com/iam

Users to perform their job. IAM is secure by default; new Users have no access to AWS until permissions are explicitly granted.

IAM is natively integrated into most AWS Services. No service APIs have changed to support IAM, and applications and tools built on top of the AWS service APIs will continue to work when using IAM. Applications only need to begin using the access keys generated for a new User.

You should minimize the use of your AWS Account credentials as much as possible when interacting with your AWS Services and take advantage of IAM User credentials to access AWS Services and resources.

Secure your Application

Every Amazon EC2 instance is protected by one or more *security groups*[43], named sets of rules that specify which ingress (i.e., incoming) network traffic should be delivered to your instance. You can specify TCP and UDP ports, ICMP types and codes, and source addresses. Security groups give you basic firewall-like protection for running instances. For example, instances that belong to a web application can have the following security group settings:

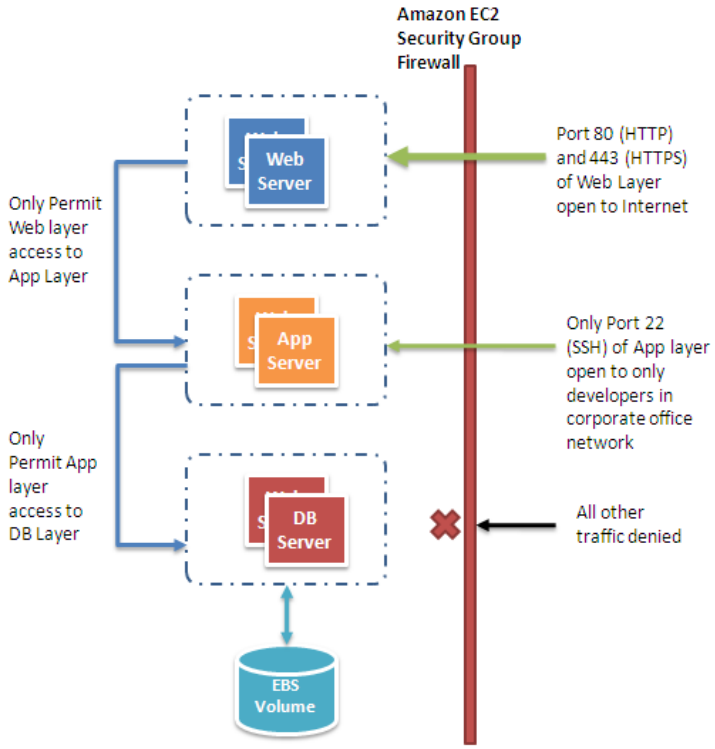

Figure 4: Securing your Web Application using Amazon EC2 Security Groups

Another way to restrict incoming traffic is to configure software-based firewalls on your instances. Windows instances can use the built-in firewall[44]. Linux instances can use *netfilter*[45] and *iptables*.

[43] More info about Security Group is available at http://docs.amazonwebservices.com/AWSEC2/2009-07-15/UserGuide/index.html?using-network-security.html

[44] http://technet.microsoft.com/en-us/library/cc779199(WS.10).aspx, March 2003

Over time, errors in software are discovered and require patches to fix. You should ensure the following basic guidelines to maximize security of your application:

- Regularly download patches from the vendor's web site and update your AMIs
- Redeploy instances from the new AMIs and test your applications to ensure the patches don't break anything. Ensure that the latest AMI is deployed across *all* instances
- Invest in test scripts so that you can run security checks periodically and automate the process
- Ensure that the third-party software is configured to the most secure settings
- Never run your processes as *root* or *Administrator* login unless absolutely necessary

All the standard security practices pre-cloud era like adopting good coding practices, isolating sensitive data are still applicable and should be implemented.

In retrospect, the cloud abstracts the complexity of the physical security from you and gives you the control through tools and features so that you can secure your application.

[45] http://www.netfilter.org/

Future Research Directions

The day is not too far when applications will cease to be aware of physical hardware. Much like plugging in a microwave in order to power it doesn't require any knowledge of electricity, one should be able to *plug in* an application to the cloud in order to receive the power it needs to run, just like a utility. As an architect, you will manage abstract compute, storage and network resources instead of physical servers. Applications will continue to function even if the underlying physical hardware fails or is removed or replaced. Applications will adapt themselves to fluctuating demand patterns by deploying resources *instantaneously* and automatically, thereby achieving highest utilization levels at all times. Scalability, Security, High availability, Fault-tolerance, Testability and Elasticity will be configurable properties of the application architecture and will be an automated and intrinsic part of the platform on which they are built.

However, we are not there yet. Today, you can build applications in the cloud with some of these qualities by implementing the best practices highlighted in the paper. Best practices in cloud computing architectures will continue to evolve and as researchers, we should focus not only on enhancing the cloud but also on building tools, technologies and processes that will make it easier for developers and architects to plug in applications to the cloud easily.

Conclusion

This paper has provided prescriptive guidance to cloud architects for designing efficient cloud applications.

By focusing on concepts and best practices - like designing for failure, decoupling the application components, understanding and implementing elasticity, combining it with parallelization, and integrating security in every aspect of the application architecture - cloud architects can understand the design considerations necessary for building highly scalable cloud applications.

The AWS cloud offers highly reliable pay-as-you-go infrastructure services. The AWS-specific tactics highlighted in the paper will help design cloud applications using these services. As a researcher, it is advised that you play with these commercial services, learn from the work of others, build on the top, enhance and further invent cloud computing.

Acknowledgements

The author is profoundly grateful to Jeff Barr, Steve Riley, Paul Horvath, Prashant Sridharan and Scot Marvin for providing comments on early drafts of this paper. Special thanks to Matt Tavis for providing valuable insight. Without his contributions, the paper would not have been possible.

References and Further Reading

1. **Amazon S3 Team, Best Practices for using Amazon S3,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1904, 2008-11-26
2. **Amazon S3 Team, Amazon S3 Error Best Practices,** http://docs.amazonwebservices.com/AmazonS3/latest/index.html?ErrorBestPractices.html, 2006-03-01
3. **Amazon SimpleDB Team, Query 201: Tips and Tricks for Amazon SimpleDB Query,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1232&categoryID=176, 2008-02-07
4. **Amazon SimpleDB Team, Building for Performance and Reliability with Amazon SimpleDB,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1394&categoryID=176, 2008-04-11
5. **Amazon SimpleDB Team, Query 101: Building Amazon SimpleDB Queries,** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1231&categoryID=176, 2008-02-07
6. **J. Varia, Cloud Architectures,** http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf, 2007-07-01
7. **Amazon Security Team, Overview of Security Processes,** http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf, 2009-06-01
8. **Amazon Web Services Team, Creating HIPAA-Compliant Medical Data Applications With AWS,** http://awsmedia.s3.amazonaws.com/AWS_HIPAA_Whitepaper_Final.pdf, 2009-04-01
9. D. Obasanjo, Building Scalable Databases: Pros and Cons of Various Database Sharding Schemes, http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx, 2009-01-16
10. D. Pritchett, Shard Lessons, http://www.addsimplicity.com/adding_simplicity_an_engi/2008/08/shard-lessons.html, 2008-08-24
11. J. Hamilton, On Designing and Deploying Internet-Scale Services, 2007, *21st Large Installation System Administration conference (LISA 'O7)*, http://mvdirona.com/jrh/talksAndPapers/JamesRH_Lisa.pdf
12. J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters2004-12-01, In Proc. of the 6th OSDI, http://labs.google.com/papers/mapreduce-osdi04.pdf
13. T. Schlossnagle, *Scalable Internet Architectures*, Sams Publishing , 2006-07-31,
14. M. Lurie, The Federation: Database Interoperability, http://www.ibm.com/developerworks/data/library/techarticle/0304lurie/0304lurie.html, 2003-04-23
15. E. Hammond, Escaping Restrictive/Untrusted Networks with OpenVPN on EC2, http://alestic.com/2009/05/openvpn-ec2, 2009-05-02
16. R. Bragg, The Encrypting File System, http://technet.microsoft.com/en-us/library/cc700811.aspx, 2009
17. S. Swidler, How to keep your AWS credentials on an EC2 instance securely, http://clouddevelopertips.blogspot.com/2009/08/how-to-keep-your-aws-credentials-on-ec2.html, 2009-08-31
18. **Amazon SQS Team, Building Scalable, Reliable Amazon EC2 Applications with Amazon SQS,** http://sqs-public-images.s3.amazonaws.com/Building_Scalabale_EC2_applications_with_SQS2.pdf, 2008
19. Microsoft Support Team, Best Practices For Encrypting File System (Windows), http://support.microsoft.com/kb/223316, 2009
20. Solaris Security Team, ZFS Encryption Project (OpenSolaris), http://www.opensolaris.org/os/project/zfs-crypto/, 2009-05-01

21. **Amazon RDS Team, Amazon RDS Multi-AZ Deployments,** http://docs.amazonwebservices.com/AmazonRDS/latest/DeveloperGuide/Concepts.DBInstance.html#Concepts.MultiAZ. 2010-05-15

22. **Amazon SimpleDB Team, Amazon SimpleDB Consistency Enhancements** http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3572 2010-02-24