Web Application Hosting in the AWS Cloud

Best Practices

May 2010

Matt Tavis

Abstract

Highly-available and scalable web hosting can be a complex and expensive proposition. Traditional scalable web architectures have not only needed to implement complex solutions to ensure high levels of reliability, but have also required an accurate forecast of traffic to provide a high level of customer service. Dense peak traffic periods and wild swings in traffic patterns result in low utilization rates of expensive hardware, yielding high operating costs to maintain idle hardware, as well as an inefficient use of capital for underutilized hardware. Amazon Web Services provides the reliable, scalable, secure, and highly performing infrastructure required for the most demanding web applications – while enabling an elastic, scale-out and scale-down infrastructure model to match IT costs with real-time customer traffic patterns.

Audience

This whitepaper is targeted at IT Managers and Systems Architects who are looking to the cloud to help them address their scalability and on demand computing needs.

An Overview of Traditional Web Hosting

Scalable web hosting is a well-known problem space and this section of the document should not reveal any surprises to those familiar with traditional web hosting models. This section will, however, introduce an example of a standard web hosting architecture which will be used for comparison when considering how this architecture could be implemented in the cloud.

An Example of a Traditional Web Application Hosting Architecture

Below is a classic illustration of a scalable web hosting architecture using a traditional web hosting model.

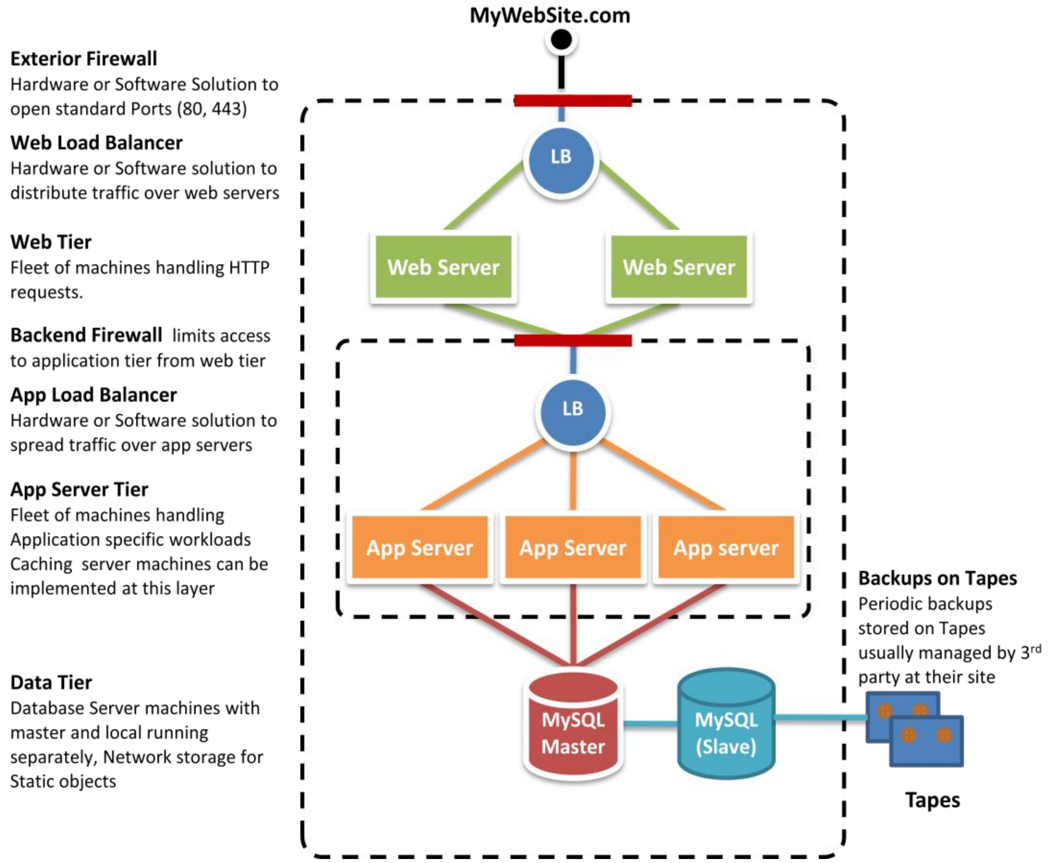

MyWebSite.com
Exterior Firewall
Hardware or Software Solution to open standard Ports (80, 443)
Web Load Balancer
Hardware or Software solution to distribute traffic over web servers
Web Tier
Fleet of machines handling HTTP requests.
Backend Firewall limits access to application tier from web tier
App Load Balancer
Hardware or Software solution to spread traffic over app servers
App Server Tier
Fleet of machines handling Application specific workloads Caching server machines can be implemented at this layer
Data Tier
Database Server machines with master and local running separately, Network storage for Static objects
LB
Web Server
Web Server
LB
App Server
App Server
App server
MySQL Master
MySQL (Slave)
Backups on Tapes
Periodic backups stored on Tapes usually managed by 3rd party at their site
Tapes

Figure 1 - A Traditional Web Application Architecture

This traditional web hosting architecture is built around a common three-tier web application model that separates the architecture into presentation, application and persistence layers. This architecture has already been designed to scale out by adding additional hosts at the presentation, persistence or application layers and has built-in performance, failover and availability features. The following sections will look at why and how such an architecture should be and could be migrated into the Amazon Web Services cloud.

Web Application Hosting in the Cloud using Amazon Web Services

The traditional web hosting architecture (figure 1) is easily portable to the cloud services provided by the AWS products with only a small number of modifications, but the first question that should be asked is whether it makes sense to move this application into the cloud:

What is the value of moving a classic web application hosting solution into the AWS cloud?

How AWS Can Solve Common Web Application Hosting Issues

If you are responsible for running a web application then there are a variety of infrastructure and architecture issues that you face for which AWS can give you easy, seamless and cost-effective solutions. The following are just some of the benefits of using AWS over a traditional hosting model:

A Cost-Effective Alternative to Oversized Fleets Needed to Handle Peaks

In the traditional hosting model, servers need to be provisioned to handle peak capacity and the unused cycles are wasted outside of peak periods. AWS-hosted web applications can leverage on-demand provisioning of additional servers, which allows capacity and costs to follow the traffic model.

For example, the following graph shows a web application with a peak from 9 AM to 3 PM that is less utilized the remainder of the day. An auto-scaling approach based on following actual traffic trends and provisioning resources only when needed would result in a capacity and cost reduction of greater than 50%.

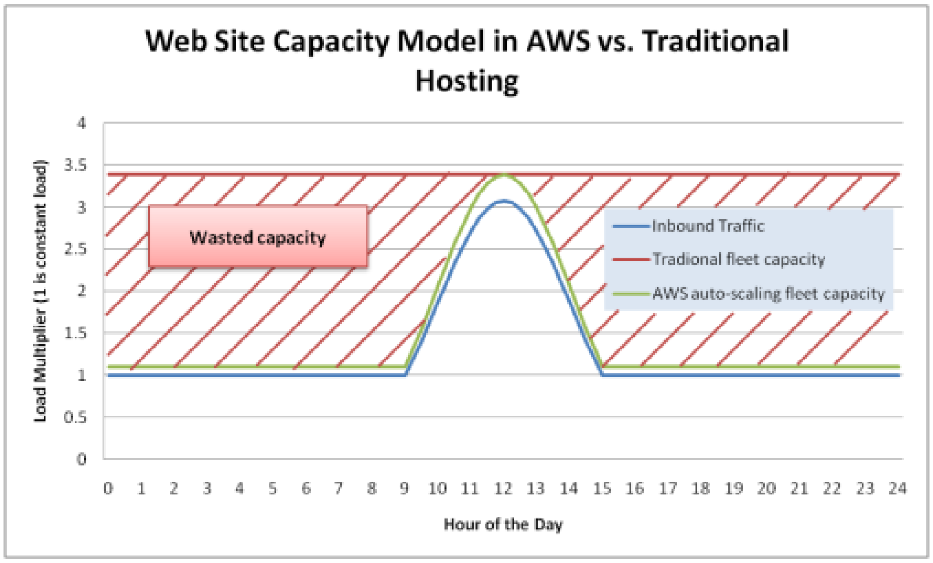


Figure 2 - An Example of Wasted Capacity in a Classic Hosting Model

A Scalable Solution to Handling Unexpected Traffic Peaks

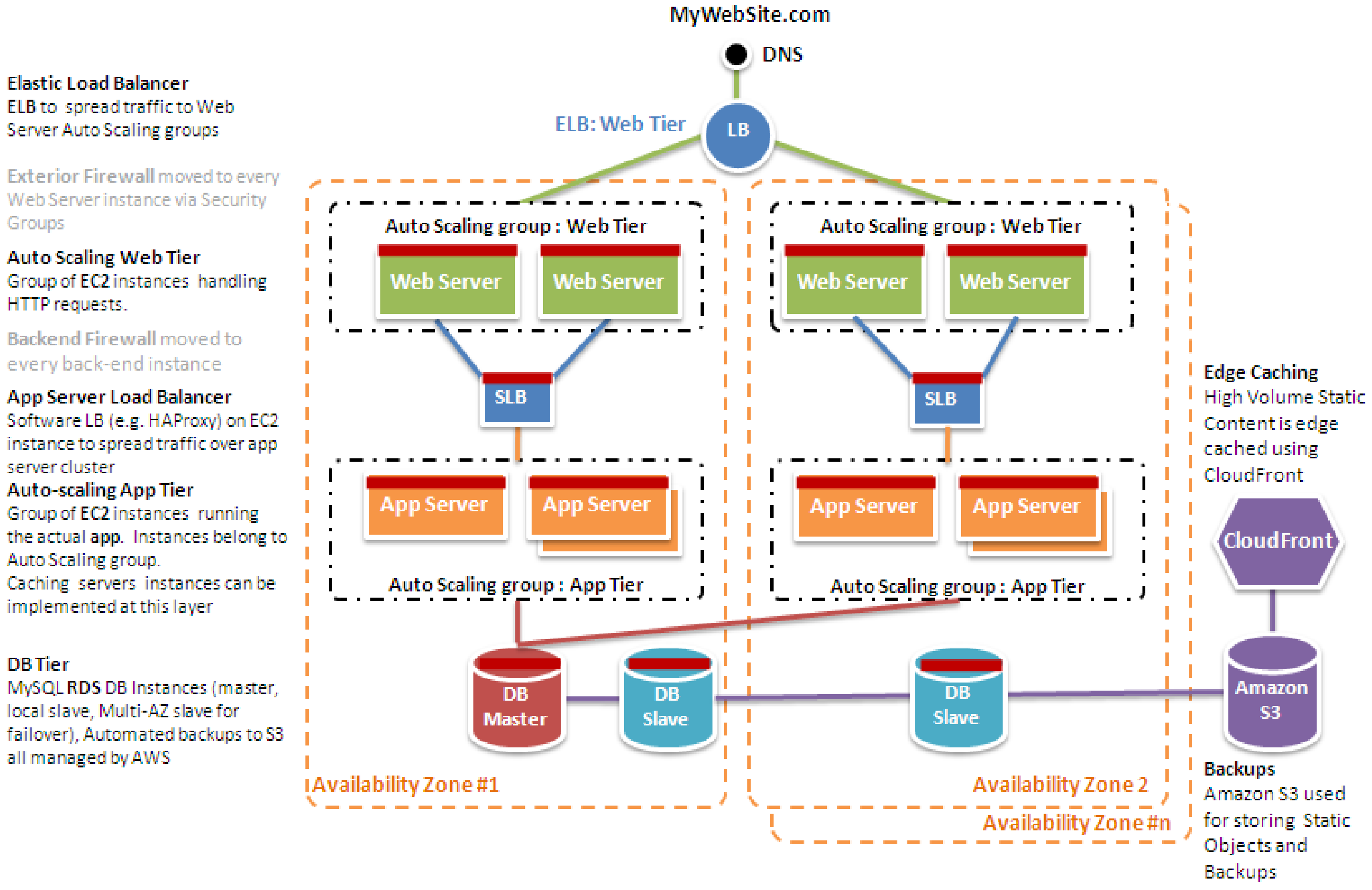
An even more dire consequence of the slow provisioning associated with a traditional hosting model is the inability to respond in time to unexpected traffic spikes. There are many stories about web applications going down due to an unexpected spike in traffic due to a mention in the popular media. The same on-demand capability that helps web applications scale-out and scale-in to match regular traffic spikes can also be used to handle an unexpected load as new hosts can be launched and ready in a matter of minutes.

An On-Demand Solution for Test, Load, Beta and Pre-Production Environments

The hardware costs of building out a traditional hosting environment for production web application don't stop with the production fleet. Quite often pre-production, beta and testing fleets need to be created as well to ensure the quality of the web application at each stage of the development lifecycle before it is launched on the production fleet. While various optimizations can be made to ensure the highest possible utilization of this testing hardware, it is often not the case that these parallel fleets are utilized optimally. A lot of expensive hardware sits unused for long periods of time. When using the AWS cloud, testing fleets can be provisioned as needed to ensure that these fleets are available only when required. Additionally, the AWS cloud can be utilized to simulate user traffic to load test a release candidate. These parallel fleets can also be used as a staging environment for a new production release, which allows for the quick switch over from current production to a new application version with little to no service outages.

An AWS Cloud Architecture for Web Hosting

Below is another look at that classic web application architecture and how it could leverage the AWS cloud computing infrastructure:



Elastic Load Balancer
ELB to spread traffic to Web Server Auto Scaling groups

Exterior Firewall moved to every Web Server instance via Security Groups

Auto Scaling Web Tier
Group of **EC2** instances handling HTTP requests.

Backend Firewall moved to every back-end instance

App Server Load Balancer
Software LB (e.g. HAProxy) on EC2 instance to spread traffic over app server cluster

Auto-scaling App Tier
Group of **EC2** instances running the actual **app**. Instances belong to Auto Scaling group.
Caching servers instances can be implemented at this layer

DB Tier
MySQL **RDS** DB Instances (master, local slave, Multi-AZ slave for failover), Automated backups to S3 all managed by AWS

Edge Caching
High Volume Static Content is edge cached using CloudFront

Backups
Amazon S3 used for storing Static Objects and Backups

Figure 3 - An Example of a Web Hosting Architecture on AWS

Key Components of an AWS Web Hosting Architecture

The following sections outline some of the key components of an AWS web hosting architecture and how they differ from a traditional web hosting architecture.

Content Delivery

Edge caching is still relevant when using the Amazon Web Service cloud computing infrastructure. Any existing solutions in your web application infrastructure should work just fine with the AWS cloud. One additional option, however, is made available when using AWS, which is to utilize the Amazon CloudFront service[1] for edge caching of your application

[1] Amazon CloudFront - http://aws.amazon.com/cloudfront/

assets stored in the Amazon Simple Storage Service [2](Amazon S3). A major benefit to using a edge caching solution with Amazon EC2 is the ability to accelerate your application performance in the eyes of your customers via a local edge cache point-of-presence (POP), which makes the loading of streaming or downloaded static content (e.g., Flash videos or images) much faster by serving this content for a closer location. An additional benefit of the CloudFront edge cache is that it follows the AWS *no commitments, no minimums and no contracts* model , which allows you to use what you need and only as long as you need it for edge caching.

Managing Public DNS using CNAMEs and Elastic IPs

Migrating a web application to the AWS cloud requires making some DNS changes. AWS does not provide a public DNS management service so redirecting your public Internet traffic to your application in the AWS cloud would require changing public DNS to point to an Elastic Load Balancing CNAME or to an Elastic IP address. DNS, however, restricts the use of CNAMEs to sub-domains so the root domain (e.g., example.com) cannot point to an Elastic Load Balancer CNAME. Note that the IP addresses behind an Elastic Load Balancer can change over time, so it is not currently possible to point your root DNS A-record at the IPs behind the Elastic Load Balancer CNAME. A simple workaround for this is to assign Elastic IPs, which is a dynamically assignable static IP address, to two or more EC2 web servers in your application and have those web servers redirect web traffic to the proper sub-domain that routes traffic to the Elastic Load Balancer CNAME (e.g., www.example.com). The domain name registrar used for purchasing your public domain name should provide a simple mechanism for applying the Elastic Load Balancer CNAME to the proper sub-domain (e.g., www.example.com) and for setting the list of Elastic IP addresses for the root domain (e.g., example.com) A-records.

Host Security

Unlike a traditional web hosting model, inbound network traffic filtering should not be confined to the edge, but rather be applied at the host level. Amazon EC2 provides a feature called *security groups*, which are analogous to an inbound network firewall, that allow you to specify the protocols, ports and source IPs ranges that are allowed to reach your EC2 instances. Each EC2 instance may be assigned one or more security groups, which routes the appropriate traffic to each instance. Security groups can be configured such that only specific subnets or IP address have access to the EC2 instance, or they can reference other security groups to limit access to EC2 instances that are in specific groups. For instance, in the example AWS web hosting architecture (above), the security group for the web server cluster might only allow access for any host over TCP on ports 80 and 443 (HTTP and HTTPS) and from instances in the application server security group on port 22 (SSH) for direct host management. The security group of the application server cluster, on the other hand, might allow access from the Web Server security group for handling web requests and from your corporate subnet over TCP on port 22 (SSH) for direct host management. In this model, your support engineers could log directly into the application servers from the corporate network and then access the other clusters from the application server boxes. A deeper discussion on security can be found at the AWS Security Center[3]. This site contains security bulletins, certification information and security whitepapers that explain the security capabilities of AWS.

[2] Amazon S3 – http://aws.amazon.com/s3

[3] AWS Security Center – http://aws.amazon.com/security

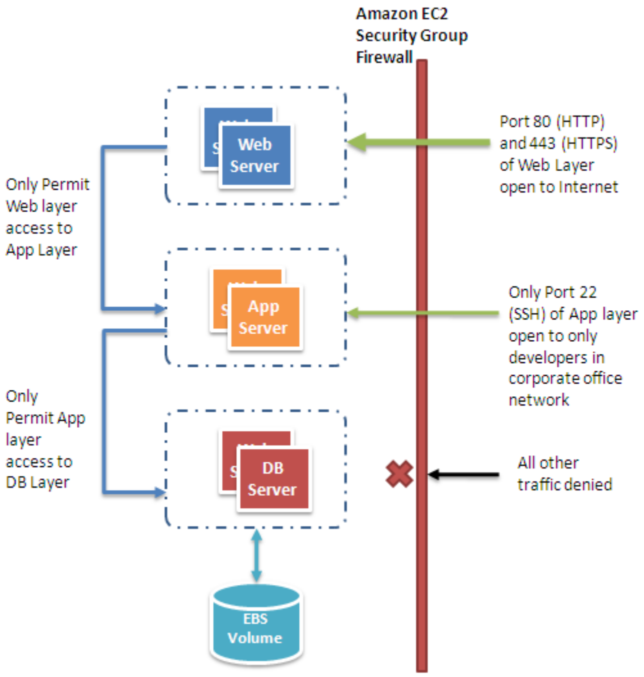

Amazon EC2
Security Group
Firewall
Web Server
Port 80 (HTTP) and 443 (HTTPS) of Web Layer open to Internet
Only Permit Web layer access to App Layer
App Server
Only Port 22 (SSH) of App layer open to only developers in corporate office network
Only Permit App layer access to DB Layer
DB Server
All other traffic denied
EBS Volume

Figure 4: Security Groups in a Web Application

Load balancing across clusters

Hardware load balancers are a common network appliance used in traditional web application architectures. AWS provides this same capability through the Elastic Load Balancing[4] service, which is a configurable load-balancing solution that supports health-checks on hosts, distribution of traffic to EC2 instances across multiple availability zones and the dynamic addition and removal of EC2 hosts from the load-balancing rotation. Elastic Load Balancing also can dynamically grow and shrink the load-balancing capacity to meet growing and shrinking traffic demands, while providing a predictable entry point via a persistent CNAME. The Elastic Load Balancing service also supports sticky sessions to address more advanced routing needs. If your application should require more advanced load-balancing capabilities then an alternative approach would be to use a software load-balancing package on top of EC2 instances (e.g. Zeus, HAProxy, nginx) and assign Elastic IPs to those load-balancing EC2 instances to minimize DNS changes.

Finding other hosts and services

Another change from the traditional web hosting architecture is that most of your hosts will have dynamic IP addresses. Although every EC2 instance can have both public and private DNS entries and will be addressable over the Internet, the DNS entries and the IP addresses will be assigned dynamically upon starting the instance and cannot be manually assigned. Static IPs (Elastic IPs in AWS terminology) can be assigned to running instances once they are launched, but only those hosts in the AWS cloud with Elastic IPs will have consistent endpoints for network communications. Elastic IPs should be used for instances and services that require consistent endpoints, such as, master databases, central file servers and EC2-hosted load-balancers. Instance types that can easily scale out and in, such as web servers, should be made discoverable at their dynamic endpoints via the more persistent services mentioned above. A simple solution to

[4] Amazon Elastic Load Balancing - http://aws.amazon.com/elasticloadbalancing

this is to centrally maintain the configuration of instances and the required network service endpoints, which can then be accessed as needed via consistent, Elastic IP-based endpoints that can be used during instance startup via bootstrapping scripts. Since most web application architectures have a database server, which is always on, this is a common repository for this type of configuration information. It should be noted that, Reserved Instances[5] should be considered for any of the persistent services in your EC2 infrastructure to further reduce your costs. Using this model, newly added hosts can request the list of necessary endpoints for communications from the database as part of a bootstrapping phase. The location of the database can be provided as user data [6]passed into each instance during the launching of the instance. Alternatively, the SimpleDB service can be used for storing and maintaining configuration information since it is a highly-available service that is available at a well-defined endpoint.

Caching within the web application

Software-based caching solutions within your existing web application architecture can most likely be used as-is in the AWS cloud. Simply building an EC2 instance with your caching software solution is sufficient to enable caching in the AWS cloud. Web and application layer caching can be done in this way and the centralized configuration in the database can help web and application servers find the appropriate cache servers.

Database configuration, backup and failover

Many web applications contain some form of persistence and usually in the form of a database. AWS offers two main solutions for databases in the cloud; hosting a relational database (RDBMS) on an Amazon EC2 instance or using the Amazon Relational Database Service (RDS) for a hosted RDBMS solution. AWS supports a large number of database solutions on EC2 including MySQL, Oracle, SQLServer and DB2. Using RDS offers the connectivity (e.g., ODBC or JDBC) that developers are familiar with, but offers simplified management that is made available via web service APIs. Common tasks, such as, adding storage, backing up the data and migrating the database to a larger EC2 instance can be automated via simple API calls. Just like in the traditional hosting model, databases solutions in the cloud should have both master and slave instances to support backup and failover. AWS customers hosting a database on EC2 have successfully used a variety of master/slave and replication models on EC2 instances, including mirroring for read-only copies and log shipping for always-ready passive slaves. Often web applications use a specific database backup and failure model and chances are that many of these can easily be replicated in the AWS cloud. AWS customers using RDS are provided with backup and failover mechanisms built in via simple API calls. Deploying a RDBMS across multiple Availability Zones using multiple slaves for failover is recommended to maximize the availability of the database. Using a second Availability Zone is very similar to having a back-up datacenter since each Availability Zone is entirely separated from the other zones in the same region to ensure maximum availability of the region. AWS customers using RDS can take advantage of the Multi-AZ functionality which will automatically deploy a hot standby slave instance in a different Availability Zone.

An additional consideration when running databases directly on EC2 (i.e., not using RDS) is the availability of fault-tolerant and persistent storage. For this purpose, it is recommended that databases running on Amazon EC2 utilize Amazon Elastic Block Storage (Amazon EBS) volumes, which are akin to network attached storage for running EC2

[5] Reserved Instances - http://aws.amazon.com/ec2/reserved-instances/

[6] User Data - http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/index.html?ApiReference-ItemType-RunInstancesType.html

instances. For EC2 instances running a database, all database data and logs should be placed on Amazon EBS volumes, which will remain available even if the database host fails. This allows for a simple failover scenario where a new EC2 instance can be launched in the case of a host failure and the existing Amazon EBS volumes can simply be attached to the new instance to allow the database to pick up where it left off. Additionally, Amazon EBS volumes automatically provide redundancy within the Availability Zone, which increases their availability over simple disks. Should the performance of a single Amazon EBS volume not be sufficient for your databases needs then volumes can be striped to increase IOPS performance for your database. When using RDS, the management of Amazon EBS volumes is handled by the RDS service.

In addition to support for relational databases on EC2, AWS also offers the SimpleDB service, which can provide a lightweight, highly available and fault-tolerant core non-relational database service offering querying and indexing of data without the requirement of a fixed schema. SimpleDB can be a very effective replacement for databases in data access scenarios that require one big, highly-indexed and flexible schema table. Example uses for SimpleDB are configuration management data, product catalogs and session data. Additionally, EC2 supports the hosting of many other emerging technologies in the NoSQL movement, such as Cassandra, CouchDB and MemcacheDB.

Storage and backup of data and assets

There are numerous options within the AWS cloud for storing, accessing and backing up your web application data and assets. The Amazon Simple Storage Service (Amazon S3) provides a highly-available and redundant object store. This is a great storage solution for somewhat static or slow-changing objects, such as, images, videos and other static media. Amazon S3 also supports the edge caching and streaming of these assets via the CloudFront service. For attached file system like storage, EC2 instances can have Amazon Elastic Block Storage volumes attached, which can act as mountable disks for running EC2 instances. Amazon EBS is great for data that needs to be accessed as block storage and requires persistence outside the life of the running instance, such as, database partitions and application logs. In addition to being persistent outside of the EC2 instance, snapshots of Amazon EBS volumes can be taken and stored in Amazon S3, which can be used for backing up running instance data. EBS snapshots are incremental in terms of data backed up and more frequent snapshots can reduce snapshot times. Amazon EBS volumes can be created up to 1 TB in size and multiple Amazon EBS volumes can be striped for even larger volumes or for increased I/O performance. Another useful feature of Amazon EBS snapshots is that they can be used as a baseline for replicating data across multiple Amazon EBS volumes and attaching to other running instances.

Auto Scaling the fleet

One of the key differences between the AWS web architecture and the traditional hosting model is the ability to dynamically scale the web application fleet on-demand to handle increased or decreased traffic. In the traditional hosting model, traffic forecasting models are generally used to provision hosts ahead of projected traffic. In an AWS cloud architecture, instances can be provisioned on-the-fly based on a set of triggers for scaling the fleet out and back in. Amazon Auto Scaling can be used to create capacity groups of servers that can grow or shrink on-demand. Auto Scaling also works directly with CloudWatch for metrics data and the Elastic Load Balancing service for addition and removal of hosts for load distribution. For example, if the web servers are reporting greater than 80% CPU utilization over a period of time then an additional web server could be quickly deployed, and then automatically added to the Elastic Load Balancer for immediate inclusion in the load-balancing rotation. As shown in the AWS web hosting architecture model, multiple auto-scaling groups can be created for different layers of the architecture to allow each layer to scale independently. For example, the web server auto-scaling group might trigger scaling out and in on network I/O, whereas the application server auto-scaling group might scale out and in on CPU utilization. Minimums and maximums can also be set to help you ensure 24/7 availability and to cap to usage within a group. Auto Scaling triggers can be set to both grow and shrink the total fleet at that layer to match the resource utilization to actual traffic needs. In addition to the Auto Scaling service, EC2 fleets can be easily scaled through the EC2 APIs directly, which allow for launching, terminating and inspecting instances.

Failover with AWS

Another key advantage when using AWS versus traditional web hosting is the Availability Zones that give the web application developer easy access to multiple locations for the deployment of instances. 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. As can be seen in the AWS web hosting architecture, it is recommended to spread EC2 hosts across multiple Availability Zones since this provides for an easy solution to making your web application fault tolerant. Care should be taken to make sure that there are provisions for migrating single points of access across Availability Zones in the case of failure. For example, it is recommended that a database slave be setup in a 2nd Availability Zones to ensure that the persistence of data remains consistent and highly available even during an unlikely failure scenario.

While there are often some required architectural changes needed while migrating an existing web application onto the AWS cloud there are significant scalability, reliability and cost-effectiveness improvements that make utilizing the AWS cloud well worth the effort.

Key Considerations when Using AWS for Web Hosting

When migrating to the AWS cloud there are some key differences from a traditional hosting model. The previous section highlighted many of the key areas of consideration when deploying a web application to the cloud. The following section points out some of the key architectural shifts that need to be considered when bringing any application into the cloud.

No more physical network appliances

Physical network appliances may not be deployed in AWS. For example, firewalls, routers and load-balancers for your AWS applications can no longer reside on physical devices but rather need to be replaced with software solutions. There are quite a wide variety of enterprise quality solutions to any of these problems, whether it is load balancing (e.g., Zeus, HAProxy, nginx, Pound, …) or establishing a VPN connection (e.g., OpenVPN, OpenSwan, Vyatta, …). This is not a limitation of what can be run on the AWS cloud, but it will be an architectural change in your application if you utilize these devices today.

Firewalls everywhere

Where you once had a simple DMZ and then open communications between your hosts in a traditional hosting model, AWS enforces a more secure model where every host is locked down. One of the steps in planning an AWS deployment will be the analysis of traffic between hosts, resulting in decisions on exactly what ports need to be opened. Security Groups within EC2 can be created for each type of host in your architecture and a large variety of simple and tiered security models can be created to enable the minimum access between hosts within your architecture.

Multiple data centers available

Availability Zones within EC2 should be thought of as multiple datacenters. They are both logically and physically separated and provide an easy-to-use model for deploying your application across data centers for both high availability and reliability.

Treat hosts as ephemeral and dynamic

Probably the most important shift in how you might architect your AWS application is the fact that EC2 hosts should be considered ephemeral and dynamic. Any application built for the AWS cloud should not assume that a host will always be available and should know that any local data (i.e., not on an Amazon EBS volume) will be lost on failure. Additionally, when a new host is brought up, no assumptions should be made about its IP address or location within an Availability Zone. This forces a more flexible configuration model and a solid approach to bootstrapping a host, but these same techniques are critical for building and running a highly-scalable and fault-tolerant application.

Closing Thoughts

There are numerous architectural and conceptual considerations when contemplating the migration of a web application into the AWS cloud, but the benefits of having a cost-effective, highly-scalable and fault-tolerant infrastructure that grows with your business far outstrips the efforts of migrating to the AWS cloud.