# Software- as- a- Service (SaaS) on AWS

***Business and Architecture Overview***

## SaaS and AWS Introduction

Software- as- a –Service (SaaS) is an application delivery model that enables users to utilize a software solution over the Internet. SaaS revenue models are typically subscription based, where users pay a fixed recurring fee over a period of time (often monthly or annually). SaaS providers are drawn to Amazon Web Services (AWS) as an ideal infrastructure platform for their SaaS business – **as the AWS usage based pricing model and scale on demand infrastructure** aligns well with their revenue and operating models. This whitepaper explores the business and technical considerations of deploying a SaaS offering on AWS versus a co-location or traditional hosting environment.

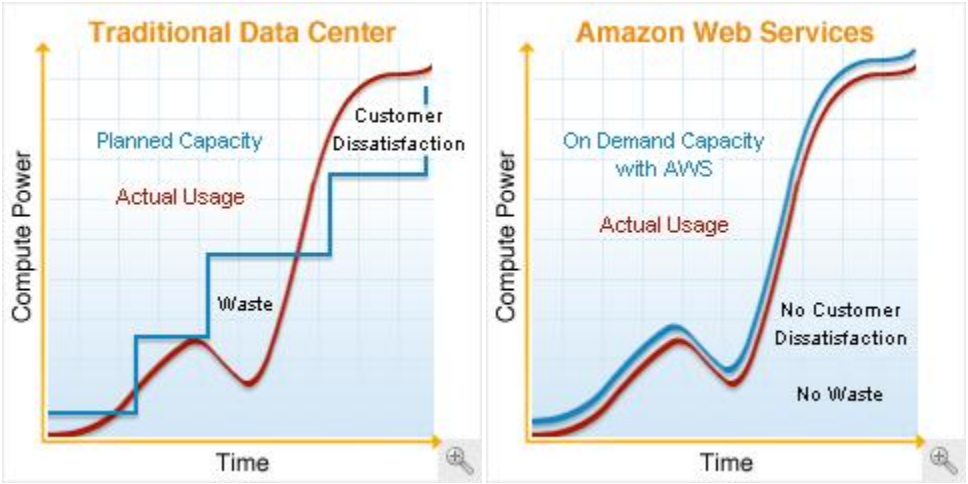
## Business Considerations

SaaS offerings present a number of unique challenges when compared to traditional software product models. Specifically, providers must take into account the following:

### Managing Cash Flow

Subscription pricing models, in contrast to perpetual licensing models, don't have large, one time, upfront license revenue coming in as customers on-board. Instead, customers pay smaller, recurring fees over the lifetime of the contract. When choosing a hosting platform, this is an important difference to keep in mind. Traditional hosting environments require large initial capital outlays to procure required infrastructure before a single customer can be on-boarded. Forecasting demand well ahead of actual usage becomes necessary, which in turn exposes two distinct business risks: If the business performs better than expected, there may not be enough servers on hand to support new paying customers. On the other hand, if demand is over-forecasted, you will have low asset utilization, high fixed operating expenses, and ultimately increase the required time duration to recognize positive cash flow for a given customer (**see figure below)**.

AWS enables providers to avoid the expense of owning servers or operating data centers. Our service enables SaaS providers to add and remove resources as needed based on the real-time demands of their business. This significantly decreases forecasting risk, and improves cash flow positions by only paying for resources when they are actually needed. AWS services can be consumed without minimum usage contracts or long-term commitments, helping providers retain maximum business flexibility.



While there is a transition required from the traditional perpetual license business model, ISVs and shareholders find a recurring subscription model to be favorable on the balance sheet as it builds a more predictable, deferred revenue stream. In difficult economic times, net new license revenue tends to be the first to go.

**Reduce Platform Support**

A SaaS delivery model allows an ISV to dedicate their valuable engineering resources to develop technology that leads to greater innovation and differentiation of their solution in the market. With the distributed client server computing model, the end user takes on the responsibility of building the platform required to support the business application. As such, there are an infinite number of configurations that an ISV must "certify" to in order to support their customers. Some call this the Matrix of Pain. Consider an ISV who has 10 products, that needs support on 10 different platform configurations (i.e. Windows 2003 running SQL Server on 32-bit, Windows 2008 running SQL on 64-bit, RHEL 4 running Oracle on OVM, RHEL 5 running MySQL on VMWare, etc….). Each time an update is made to one of the ISV products it mustbe tested against all 10 certified platforms. One update for all 10 software products is equal to100 different variations that need to be tested against and supported indefinitely. SaaS delivery allows an ISV to free themselves of the Matrix of Pain. Instead of spending tens of millions of dollars on sustaining engineering and legacy support, SaaS allows an ISV to focus on one platform of their choosing.

**Increase Sales Velocity and Customer Satisfaction**

TheInternet has introduced a number of conveniences in our personal and professional lives. One of the most noticeable is a reduction in time to value. Whether it is accessing movies, music, shopping for cars, or downloading a book, the internet has revolutionized customers' expectations around instant gratification. Like cloud computing, SaaS allows an ISV to provide this level of instant gratification to their customer, as well as their sales team and channel partners. No longer does an enterprise sales executive have to wait 4 weeks for a customer to acquire hardware, operating system licenses and data center space followed by installation and configuration of the business application to start an evaluation. The reality is that cloud computing allows ISVs to deliver software in a truly on demand manner. Reduce sales cycles from months to days. Reduce time to value for the customer from months to hours. Win/win.

SaaS allows the ISV to control the software upgrade cycle for the customer. In many cases, end users are reluctant to migrate to newer versions of software simply due to the hassle and uncertainty associated with doing upgrades. This impacts the ISV equally. Customers who don't upgrade need support on legacy platforms. With SaaS the customer is abstracted from the software installation, configuration, backup, and upgrade process. Therefore, the ISV can continuously upgrade the software to the latest and greatest. Meaning, the end user is always using the newest, most feature-rich, most secure and fastest version of the software. The ISV benefits from seamlessly moving customers along the upgrade path, thus improving customer satisfaction and reducing legacy support costs.

**Leverage Existing Software and Architecture**

AWS is unique in our approach to true utility based Infrastructure as a Service. Since AWS provides discrete building blocks of low level computing infrastructure, ISVs can leverage existing software and architecture and simply move it to AWS with little to no re-engineering or re-architecting. This is a critical point to note. In order for a SaaS provider to benefit from the financial advantages of a SaaS model, they must be able to achieve the greatest return on infrastructure assets. This requires running systems at the highest level of utilization possible. An ISV cannot afford to have dedicated hardware for each customer and run those systems at 20% utilization. The choices are to re-architect, in many cases rewrite software from scratch, which is very costly. Or, write new software to leverage Platform as a Service

offerings. Or, leverage AWS to achieve the economic benefits of multi-tenancy without having to re-write their existing software. As mentioned above, AWS allows the ISV to easily spin up unique environments for each new customer and pay for that infrastructure only when they need it.

### Managing Uptime for a Global Customer Base

A highly reliable and available IT infrastructure requires SaaS providers to not only maintain reliable storage and backup devices, but also operate a reliable network with redundant networking devices, transit connections, and physical connections between data centers. In addition to backup and reliable networking, SaaS providers must also have a tested, working solution for disaster recovery. This includes deploying data and applications across multiple data centers – either with failure resilient software or in a more traditional hot/cold standby approach. To achieve realistic disaster recovery, all of the data centers and servers involved have to be constantly utilized; if they sit idle, it's almost certain they won't function as desired when activated from a cold start. SaaS providers need to account for both the cost and the complexity of this redundancy when evaluating their deployment. AWS includes all this in its simple usage charges, and lets customers easily do things like deploy servers in any one of our global regions (East Coast US, West Coast US, EU, and Singapore). Within a region, ISVs can further enable the availability of their application by deploying servers across multiple Availability Zones, which provides the ability to remain resilient in the face of most failure modes including natural disasters or system failures.

### Providing a Secure Environment

Another direct cost for SaaS providers running their applications is ensuring the confidentiality, integrity, and availability of business critical data. Examples of security costs for SaaS providers include capital expenditures for network security devices, security software licenses, staffing of an information security organization, costs associated with information security regulatory compliance, physical security requirements, smart cards for access control, and so on. To provide end-to-end security and end-to-end privacy in the cloud, AWS builds services in accordance with security best practices and features, and clearly documents how developers can effectively use those features. AWS customers thus take advantage of Amazon's reliable and secure global computing infrastructure, which has been the backbone of Amazon.com's multi-billion dollar retail business for more than 15 years, at no additional cost to the customer. For more information on AWS security, consult the *Amazon Web Services: Overview of Security Processes* whitepaper at aws.amazon.com/security.

### Overall Cost

AWS passes on to providers the financial benefits of operating at Amazon's scale. In addition to server, power, and network infrastructure costs, personnel costs also need to be accounted for. These include cost of the sizable IT infrastructure teams that are needed to handle the "heavy lifting" – managing heterogeneous hardware and the related supply chain, staying up-to-date on data center design, negotiating contracts, dealing with legacy software, operating data centers, moving facilities, scaling and managing physical growth, etc. – all the things that AWS's services handle on behalf of SaaS providers.

## SaaS Architecture on AWS

SaaS architectures are often variations of the classic three-tier web application hosting model. Design priorities are typically reliability, security, availability, performance, and cost. To illustrate best practices for deploying this model on AWS, let's first review the traditional web hosting architecture:

***Traditional Web Application Hosting Architecture***

Below is a classic example of a scalable web hosting architecture using a traditional web hosting model:

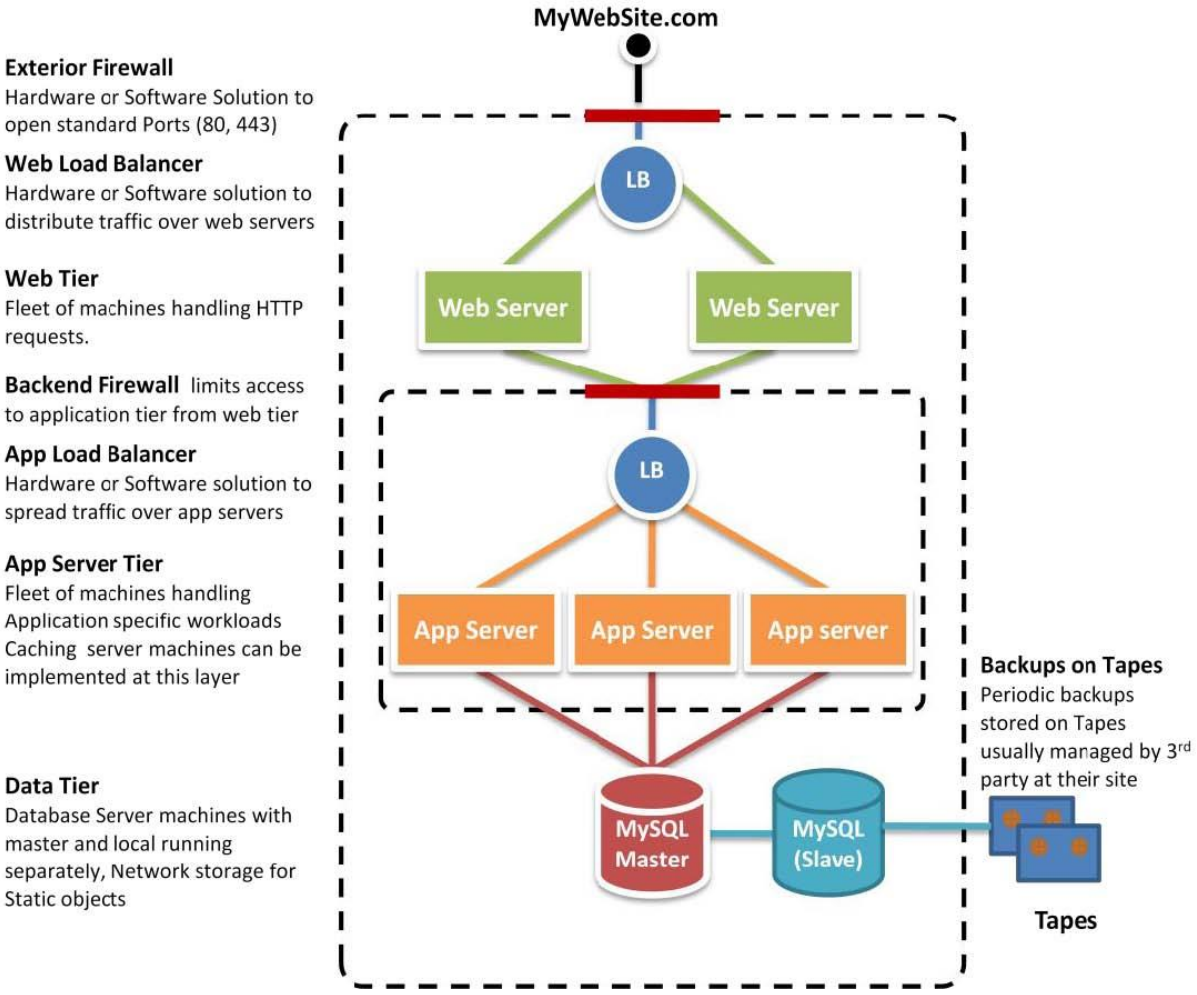



*Figure 1 - A Traditional Web Application Architecture*

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 persistence or application layers and has built-in performance, failover and availability features. The following section will look at how such architecture can be migrated in the Amazon Web Services cloud.

### *AWS Cloud Architecture for hosting SaaS Environments*

Below is another look at that classic web application architecture that many SaaS providers are built on, but this version leverages the AWS cloud computing infrastructure:

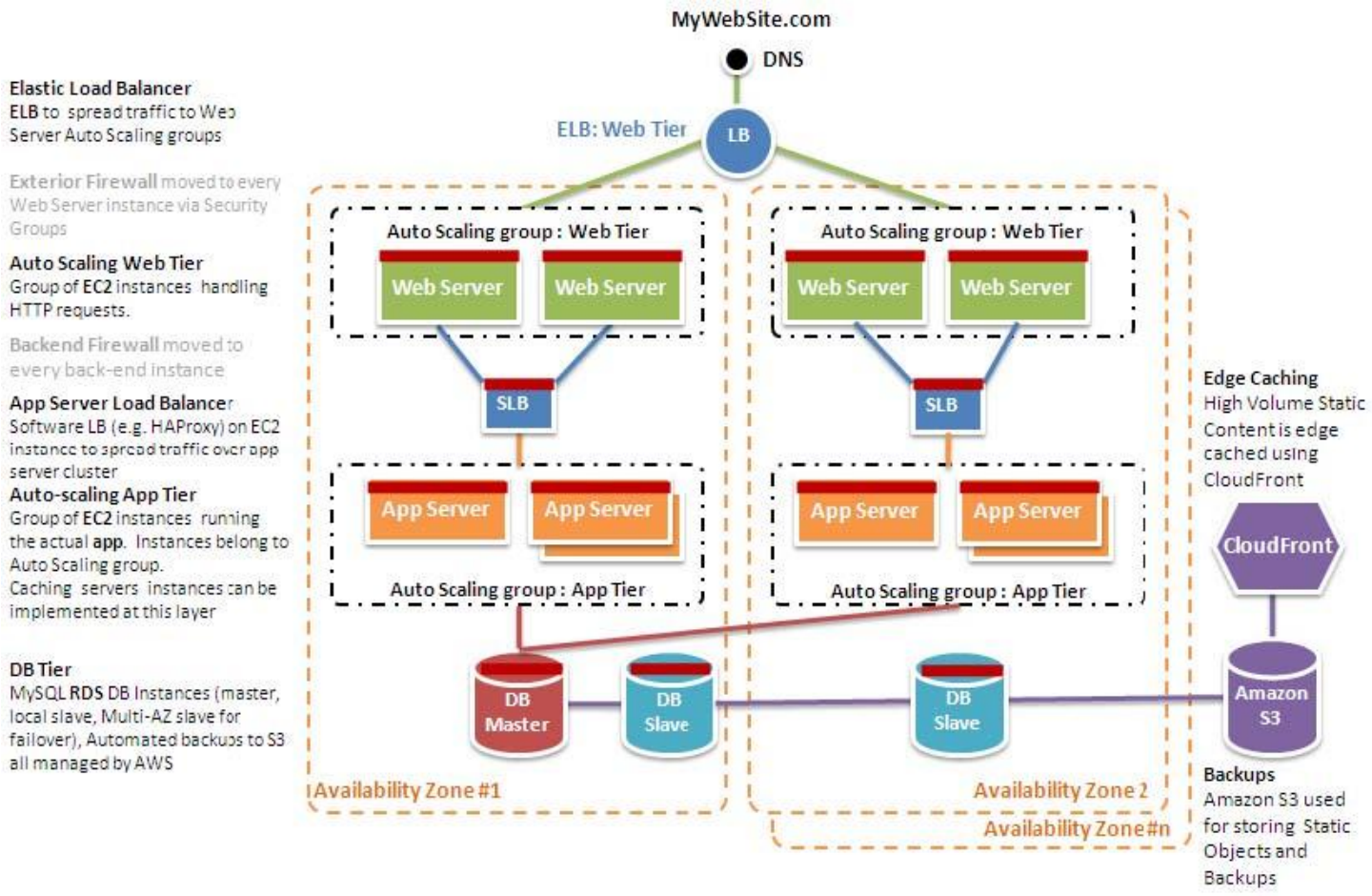



*Figure 2 - An Example of a Web Hosting Architecture on AWS*

### *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.

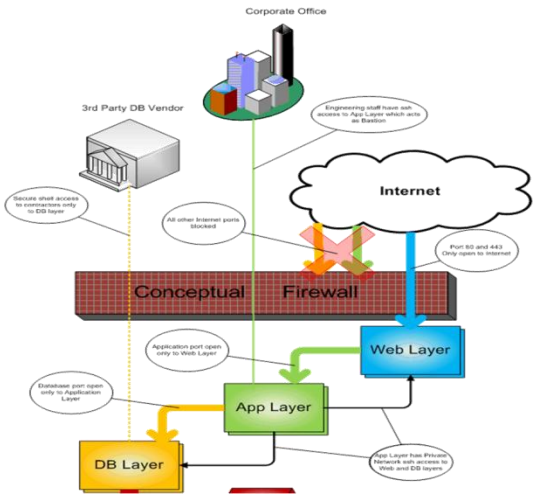
#### Edge caching

Edge caching doesn't differ when using the Amazon Web Service cloud computing infrastructure. Any existing solutions in place 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 AWS CloudFront service (http://aws.amazon.com/cloudfront/) for edge caching of your application assets stored in the AWS Simple Storage Service (S3).

### Elastic IPs and DNS

Migrating a web application to the AWS cloud does require making some DNS changes. AWS provides a DNS management service, called Route 53, but you will still need to request a number of Elastic IPs, which are static IP addresses that you can dynamically assign to running EC2 instances. A new AWS account can request five (5) Elastic IPs (EIPs) to start with but more can be requested if needed. As can be seen in the AWS Web Hosting architecture (Figure 2), these EIPs can be assigned to the public entry points of your web application and then you can point the DNS entries for your domain to these EIPs. The domain name registrar used for purchasing your public domain name should provide a simple mechanism to setting this list of IPs addresses and your incoming traffic will be round-robin load-balanced across the EIPs you register.

### Controlling access to hosts




Unlike a traditional web hosting model, there is no border firewall that controls all traffic into the data center. Instead, every EC2 host has a local firewall that you can configure for access. This is done via Security Groups, which allow you to specify the protocols, ports and source IP ranges that can send traffic to your EC2 hosts. Security groups can even reference other security groups or themselves to limit access to EC2 hosts that are in specific clusters. For example, in the example AWS web hosting architecture, the web server cluster might only allow access for any host over TCP on ports 80 and 443 (HTTP and HTTPS) and from the Application Server security group for direct host management. 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.

### Load balancing across clusters

A common network appliance in a traditional web hosting is a load-balancer. In the AWS cloud, however, there are no network appliances, which is why the example AWS web hosting architecture has an EC2 instance running a software load balancer. Providers also have the option of leveraging Amazon's Elastic Load Balancing Service, which automatically distributes incoming application traffic across multiple Amazon EC2 instances. It seamlessly provides the amount of load balancing capacity needed in response to incoming application traffic. Elastic Load Balancing detects unhealthy instances within a pool and automatically reroutes traffic to healthy instances until the unhealthy instances have been restored. Customers can enable Elastic Load Balancing within a single Availability Zone or across multiple zones for even more consistent application performance.

Alternatively, customers can deploy their own custom load-balancing solution, with many packages available to meet this need, and many have been used in the AWS cloud. Currently, the most common solution is to use HAProxy (http://haproxy.1wt.eu/), which is free to use and provides many configuration options for web server and application server load-balancing. This is a common instance type in AWS cloud architectures because the ability to spin up new hosts on demand requires a lightweight mechanism for routing traffic when hosts are dynamically being provisioned and

removed. AWS has announced future support for auto-scaling, load balancing and monitoring both individually and as a suite of services to help solve this problem in a simpler fashion. (http://www.allthingsdistributed.com/2008/10/using_the_cloud_to_build_highl.html)

**Finding other hosts and services**

Another change from the traditional web hosting architecture is that most of your hosts will have dynamic IP addresses. In fact, only those hosts in the AWS cloud with Elastic IPs (EIPs) will have predictable endpoints for network communications. A simple solution to this is to centrally maintain the configuration of hosts and the required network service endpoints. Since most web application architectures have a database server, which is always on, this is a common repository for configuration. 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 passed into each instance as part of the data passed during the launching of the instance.

**Caching within the web application**

Any software-based caching solutions within your existing web application architecture need not be changed when moving to the AWS cloud. Simply building an EC2 instance with your caching software solutions is sufficient to enable this 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 caches.

**Database configuration, backup and failover**

Many web applications contain some form of persistence and it is usually in the form of a database. AWS has support for a large number of database solutions, including MySQL, Oracle, SQLServer and DB2. Not only do these database packages run on EC2 hosts but they also have licensing solutions that support the cloud, which alleviates much of the licensing confusion around database products. 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 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 offline slaves. Often web applications have a specific databases backup and failure model that is used and chances are that many of these can easily be replicated in the AWS cloud. As is shown in the AWS web hosting architecture, the use of multiple Availability Zones (AZs) for a second failover slave within EC2 is recommended to maximize availability of the database. Using a second availability zone is exactly like having a back-up datacenter since each AZ is entirely separated to ensure maximum availability. An additional consideration when running databases on EC2 is the availability of fault-tolerant and persistent disks. For this purpose, it is recommended that databases running on AWS EC2 utilize Elastic Block Storage (EBS) volumes, which are akin to network attached storage for running EC2 instances. For EC2 instances running a database, all database data, logs and temp storage should be placed on 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 EBS volumes can simply be attached to the new instance to allow the database to pick up where it left off. EBS volumes automatically provide basic mirroring, which increases their availability over simple disks. It is also recommended to stripe EBS volumes to increase IOPS performance for your database hosts.

Since the database layer represents a stable and always-on component in the web architecture, it can also be used for storing and managing the configuration of the AWS web architecture. Within the AWS cloud, hosts and services can be brought up and down on-demand and the exact network addresses are not known prior to launch, which requires that hosts relying on cross-host communications need to have a configuration service from which to draw their configurations.

Amazon also offers a Relational Database Service (Amazon RDS), that makes it easy to set up, operate, and scale a relational database in the cloud (http://aws.amazon.com/rds/). It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business .

In addition to support for databases on EC2, AWS also offers the SimpleDB (SDB) service, which can provide a lightweight, highly available and fault-tolerant core database service offering for querying and indexing of data without the requirement of a fixed schema. SimpleDB can be a very effective replacement for simple databases and can be used for configuration information for web architectures that have no persistence layer.

**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 AWS Simple Storage Service (S3) provides a highly-available and redundant object store for files up to 5GB in size. This is a great storage solution for somewhat static objects, such as graphical and media assets and S3 also supports the edge caching of these assets via the CloudFront service. In addition to S3, EC2 instances can have Elastic Block Storage (EBS) volumes attached, which can act as local disks for running EC2 instances. 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 EBS volumes can be made for backup in S3, which can be used for backing up running instances and saving server state. EBS volumes can be created up to 1TB in size and multiple EBS volumes can be striped for even larger volumes with increased IOPS performance. RAID across EBS volumes is especially useful for databases running on EC2 since it gives better reliability and performance via striping. Another useful features of EBS snapshots is that they can be used as a baseline for creating multiple EBS volumes to attach to 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. Currently, it is incumbent upon the web application developer to manage the web application fleet. One model for performing this function is to utilize the always-on host that runs the database for scaling up and down of the fleet. If the DB also holds the configuration data then this also allows for a single host to scale the fleet and update its configuration. Auto-scaling can be easily accomplished through the EC2 APIs, which allow for launching, terminating and inspecting instances. Triggering the scaling of the fleet is left to the application developer but a simple model could rely on each of the various hosts (e.g., web servers, load balancers, app servers, …) to report its load and performance statistics to a monitoring component running on the database server. This component would then add or remove servers based on availability and load. For example, if the web servers are reporting greater than 80% CPU utilization then an additional web server could be quickly deployed and then added to the load balancer host. Once the web server responds correctly to the URL checks coming from the load balancer then it would take its place in the load rotation and future requests can now be served by this newly deployed web server. This same capability can be used for handling failover scenarios as well. For example, if the monitoring component has not received status updates from Load Balancer #1 then the Load Balancer #2 could be reconfigured to add the hosts behind Load Balancer #1 for a period of time. Once a new host has been deployed to replace Load Balancer #1 then Load Balancer #2 can again be reconfigured to have its original set of hosts.

**Failover with AWS**

Another key advantage when using AWS versus traditional web hosting is the availability of simple to manage availability zones that give the web application developer access to multiple datacenters for the deployment of virtual hosts. As can be seen in the AWS web hosting architecture, it is recommended to spread EC2 hosts across multiple AZs 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 AZs in the case of failure. For example, it is recommended that a database slave be setup in a 2nd AZ to ensure that the persistence of data remains consistent and highly available even during an unlikely failure scenario.

While there are some significant architectural changes that might be needed in migrating an existing web application onto the AWS cloud there are certainly significant scalability, reliability and cost-effectiveness improvements that are possible when utilizing the AWS cloud.

## Additional Architectural considerations

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 network appliances**

In the AWS cloud there are no network appliances that you can deploy. For example, firewalls, routers and load-balancers for your AWS applications can no longer reside on physical devices but rather need to be addressed using software solutions. There is quite a wide variety of enterprise quality solutions to any of these problems, whether it be load-balancing (e.g., HAProxy) or establishing a VPN connection (e.g., OpenVPN). 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 and exactly what ports need to be opened. Security Groups within EC2 can be created for each type of host in your architecture and 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.

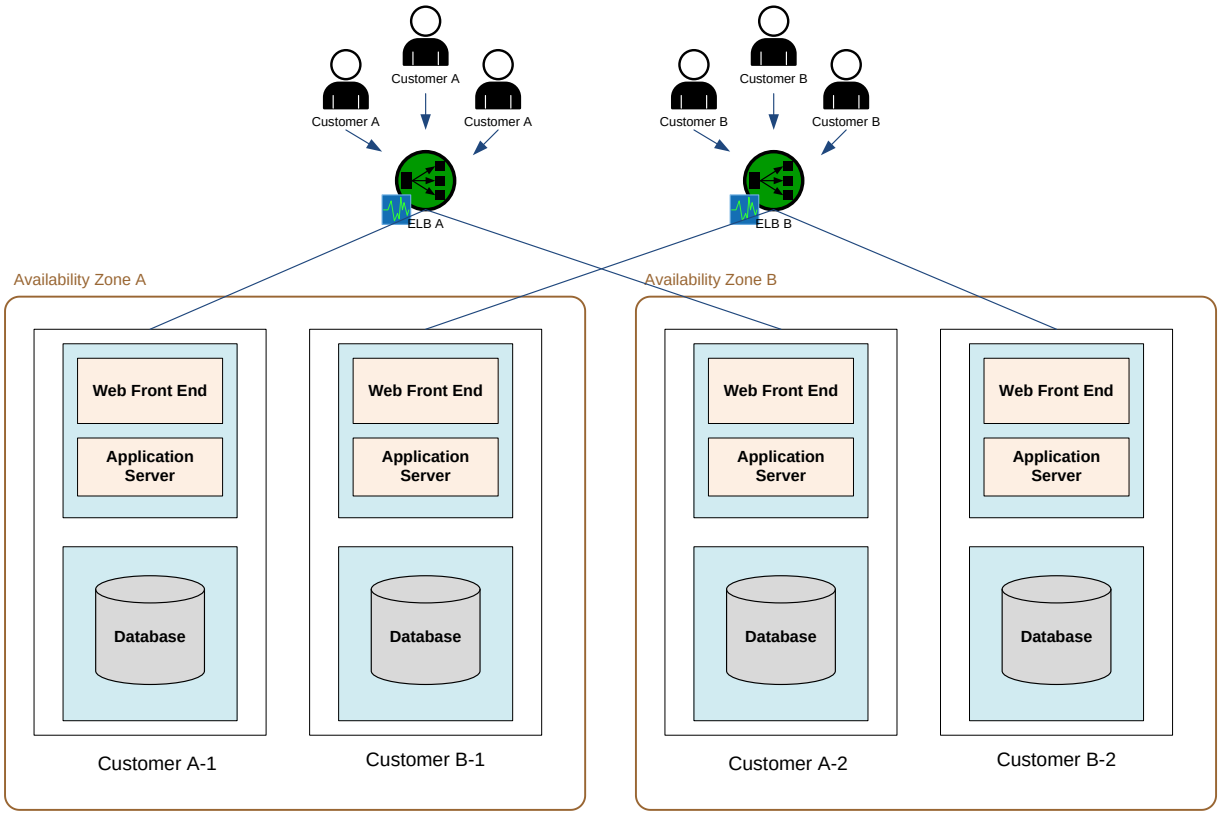
**Treat hosts as ephemeral and dynamic**

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 EBS volume) will be lost on failure. Additionally, when a new host is brought up there can be no assumptions about its IP address or location within an AZ. This forces 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.

### Approaches for Handling Multi-Tenancy

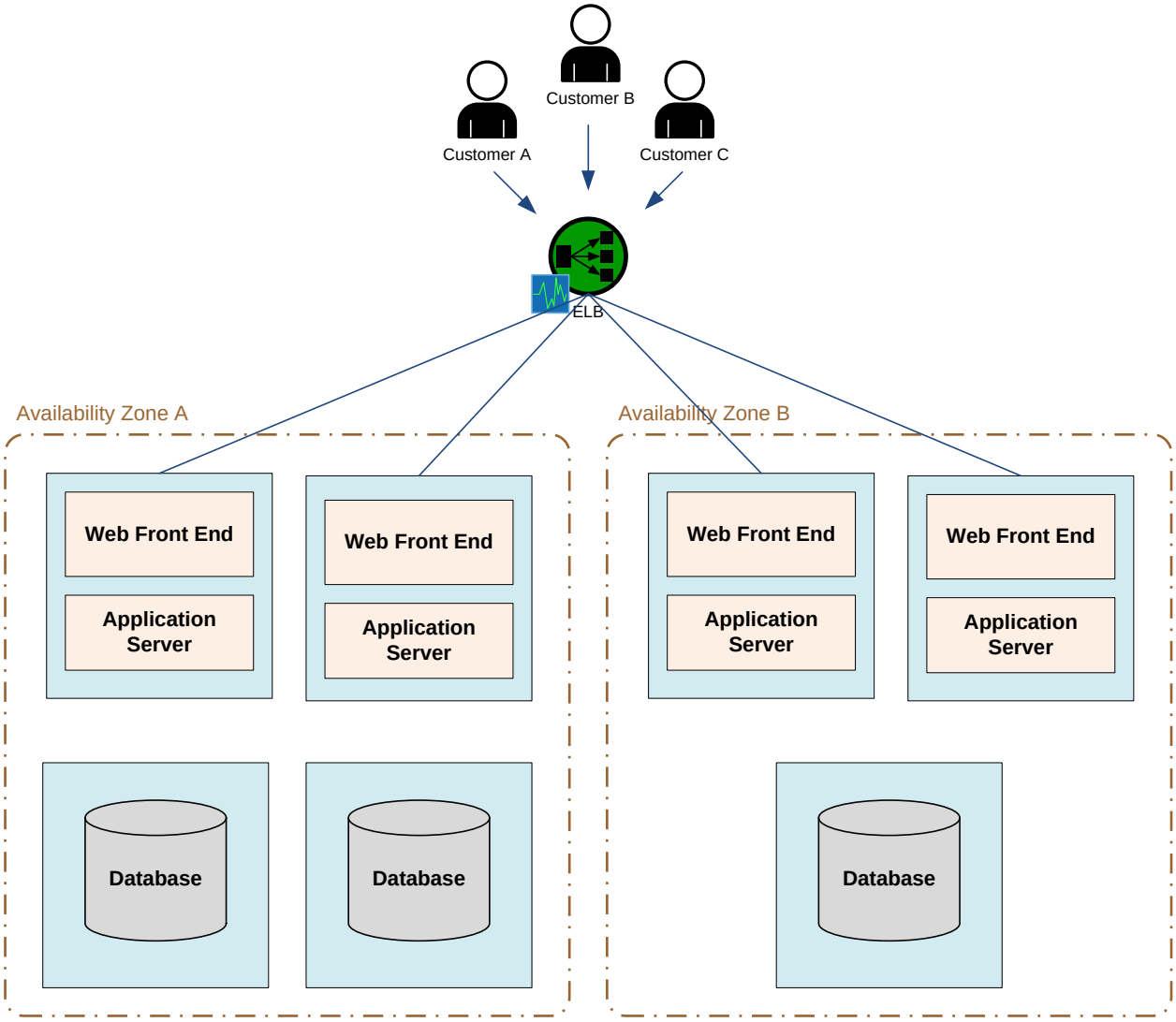
The fundamental architectural consideration when designing a SaaS application is deciding how to handle the issue of multi-tenancy. One way of handling multi-tenancy is to rely on virtualization as the mechanism for keeping application modules separated for each customer. Another option is to have a single application that has logical separation of data built and enforced in each architectural layer.

The following diagram depicts the strategy of provisioning a separate SaaS environment for each customer that leverages the best practices highlighted in this document:




This approach will make sense for a SaaS application that will be constructed from subsystems of an existing single-tenant application or packaged software. It is easy to use AWS virtualization to create separate SaaS applications that are provisioned per customer. Depending on the size and requirements of each customer this could be condensed to a single EC2 micro-instance that hosts the entire application or a grouping of many EC2 instances for each customer.

If the SaaS application is being built from the ground up, it may make more sense to create a single application that has multi-tenancy baked into every layer of the architecture. This approach is likely to make more efficient use of available resources and allow for more flexibility when operating the infrastructure. That said, it is likely going to be significantly more difficult to design and implement from a software development perspective. The following diagram depicts this approach:




## Summary

AWS offers a compelling alternative for partners who would like to focus their energy on building a differentiated software offering as opposed to building, operating and maintaining infrastructure. The dynamic and elastic nature of AWS enables partners to pay only for what is used by customers, and use by customers implies revenue generated through subscription. This means cost and revenue scale up and down together, providing a more stable and predictable balance sheet.