Building Fault-Tolerant Applications on AWS

October 2011

Jeff Barr, Attila Narin, and Jinesh Varia

Contents

Introduction

Software has become a vital aspect of everyday life in nearly every part of the world. No matter where we are, we interact with software–whether that is by using our mobile phone, withdrawing money from an automated bank machine, or even by just stopping at a traffic light.

Because software has become such an integral part of our daily lives, a great deal of work has to be done to ensure that this software remains operational and available.

Generally speaking, this area of study is known as *fault-tolerance*, the ability for a system to remain in operation even if some of the components used to build the system fail.

Although it's true that essential systems must be available at all times, we also expect a much wider range of software to always be available to us. For example, we may want to visit an e-commerce site to purchase a product. Whether it is at 9:00am on a Monday morning or 3:00am on a holiday, we expect that the site will be available and ready to accept our purchase. The cost of not meeting these expectations can be crippling to many businesses. Even with very conservative assumptions, it is estimated that a busy e-commerce site could lose thousands of dollars for every minute it is unavailable. This is just one example of why businesses and organizations strive to develop software systems that can survive faults.

Amazon Web Services (AWS) provides a platform that is ideally suited for building fault-tolerant software systems. However, this attribute is not unique to our platform. Given enough resources and time, one can build a fault-tolerant software system on almost any platform. The AWS platform is unique because it enables you to build fault-tolerant systems that operate with a minimal amount of human interaction and up-front financial investment.

Failures Shouldn't be THAT Interesting

When a server crashes or a hard disk runs out of room in an on-premises datacenter environment, administrators are notified immediately, because these are noteworthy events that require at least their attention — if not their intervention as well. The ideal state in a traditional, on-premises datacenter environment tends to be one where failure notifications are delivered reliably to a staff of administrators who are ready to spring into action in order to solve the problem. Many organizations are able to reach this state of IT nirvana – however, doing so typically requires extensive experience, up-front financial investment, and significant human resources.

This is not the case when using the platform provided by Amazon Web Services. Ideally, failures in an application built on our platform can be dealt with automatically by the system itself, and as a result, are fairly uninteresting events.

Amazon Web Services gives you access to a vast amount of IT infrastructure–computing, storage, and communications–that you can allocate automatically (or nearly automatically) to account for almost any kind of failure. You are only charged for resources that you actually use, so there is no up-front financial investment to be made.

Amazon Machine Images

Amazon Elastic Compute Cloud (Amazon EC2) is a web service within Amazon Web Services that provides computing resources – literally server instances – that you use to build and host your software systems. Amazon EC2 is a natural entry point to Amazon Web Services for your application development. You can build a highly reliable and fault-tolerant system using multiple EC2 instances—using the tools and ancillary services such as Auto Scaling and Elastic Load Balancing.

On the surface, Amazon EC2 instances are very similar to traditional hardware servers. Amazon EC2 instances use familiar operating systems like Linux, Windows, or OpenSolaris. As such, they can accommodate nearly any kind of software that can run on those operating systems. Amazon EC2 instances have IP addresses so the usual methods of interacting with a remote machine (e.g., SSH or RDP) can be used.

The template that you use to define your service instances is called an Amazon Machine Image (AMI). This template basically contains a software configuration (i.e., operating system, application server, and applications) and is applied to an *instance type*[1].

Instance types in Amazon EC2 are essentially hardware archetypes – you choose an instance type that matches the amount of memory (i.e., RAM) and computing power (i.e., number of CPUs) that you need for your application.

A single AMI can be used to create server resources of different instance types; this relationship is illustrated below.

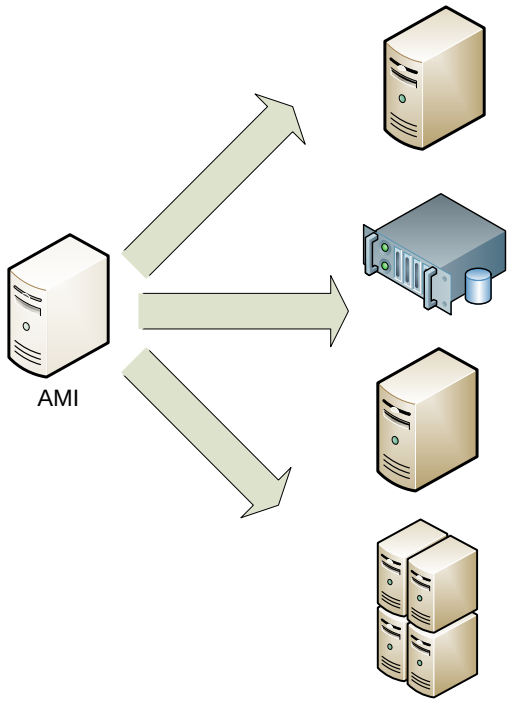

Figure 1 - Amazon Machine Image

[1] Instance Types - http://aws.amazon.com/ec2/instance-types/

Amazon publishes many AMIs that contain common software configurations. In addition, various members of the AWS developer community have also published their own custom AMIs. All of these AMIs can be found on the Amazon Machine Image resources page[2] on the AWS web site.

However, the first step towards building fault-tolerant applications on AWS is to create a library of your own AMIs. Your application should be comprised of at least one AMI that you have created. Starting your application then is simply a matter of launching the AMI.

For example, if your application is a web site or web service, your AMI should be configured with a web server (e.g., Apache or Microsoft Internet Information Server), the associated static content, and the code for all dynamic pages. Alternatively, you could configure your AMI to install all required software components and content itself by running a bootstrap script as soon as the instance is launched. As a result, after launching the AMI, your web server will start and your application can begin accepting requests.

Once you have created an AMI, replacing a failing instance is very simple; you can literally just launch a replacement instance that uses the same AMI as its template.

This can be done through an API invocation, through scriptable command-line tools, or through the AWS Management Console as illustrated below. Later in this document, we introduce the Auto Scaling service, which can replace failed or degraded instances with fresh ones automatically.

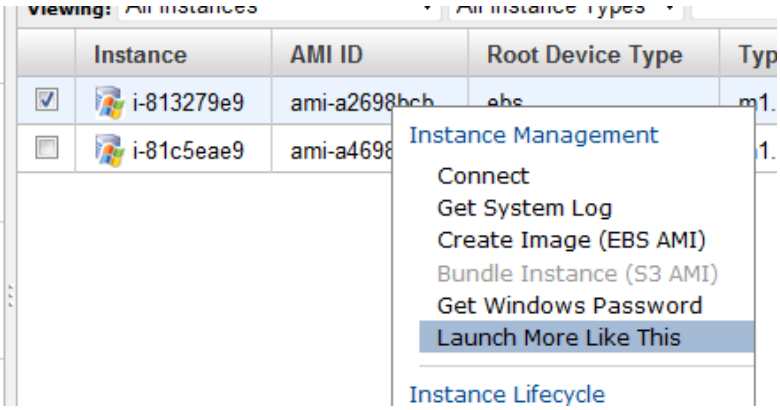

Figure 2 - Launching an Amazon EC2 Instance

This is really just the first step towards fault-tolerance. At this point, you are able to quickly recover from failures; if an instance fails, or is not behaving the way you want it to, you can simply launch another one based on the same template. To minimize downtime, you might even always keep a spare instance running – ready to take over in the event of a failure. This can be done efficiently using *elastic IP addresses*. You can easily fail over to a replacement instance or spare running instance by remapping your elastic IP address to the new instance. Elastic IP addresses are described in more detail later in the document.

Being able to quickly launch replacement instances based on an AMI that you define is a critical first step towards fault tolerance. The next step is storing persistent data that these server instances have access to.

[2] Amazon Machine Images Resources page - http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171

Elastic Block Store

Amazon Elastic Block Store (Amazon EBS) provides block level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes are off-instance storage that persists independently from the life of an instance.

Amazon EBS volumes are essentially hard disks that can be attached to a running Amazon EC2 instance. Amazon EBS is especially suited for applications that require a database, a file system, or access to raw block level storage. EBS volumes store data redundantly, making them more durable than a typical hard drive. The annual failure rate (AFR) for an EBS volume is 0.1% and 0.5%, compared to 4% for a commodity hard drive.

Amazon EBS and Amazon EC2 are often used in conjunction with one another when building a fault-tolerant application on the AWS platform. Any data that needs to persist should be stored on Amazon EBS volumes, not on the so-called "ephemeral storage" associated with each Amazon EC2 instance. If the Amazon EC2 instance fails and needs to be replaced, the Amazon EBS volume can simply be attached to the new Amazon EC2 instance. Since this new instance is essentially a duplicate of the original, there should be no loss of data or functionality.

Amazon EBS volumes are highly reliable, but to further mitigate the possibility of a failure, backups of these volumes can be created using a feature called *snapshots*. A robust backup strategy will include an interval (time between backups, generally daily but perhaps more frequently for certain applications), a retention period (dependent on the application and the business requirements for rollback), and a recovery plan. Snapshots are stored for high-durability in Amazon S3.

Snapshots can be used to create new Amazon EBS volumes, which are an exact replica of the original volume at the time the snapshot was taken. Because backups represent the on-disk state of the application, care must be taken to flush in-memory data to disk before initiating a snapshot.

These Amazon EBS operations can be performed through the API or from the AWS Management Console, as illustrated below.

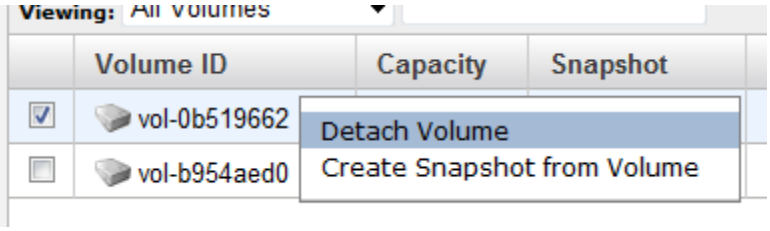

Figure 3 - Amazon EBS

Elastic IP Addresses

Elastic IP Addresses are public IP addresses that can be mapped (routed) to any EC2 instance within a particular EC2 Region. The addresses are associated with an AWS account, not to a specific instance or the lifetime of an instance, and are designed to aid in the construction of fault-tolerant applications. An elastic IP address can be detached from a failed instance and then mapped to a replacement instance within a very short time frame. As with Amazon EBS volumes (and for all other EC2 resources for that matter), all operations on elastic IP addresses can be performed programmatically through the API, or manually from the AWS Management Console:

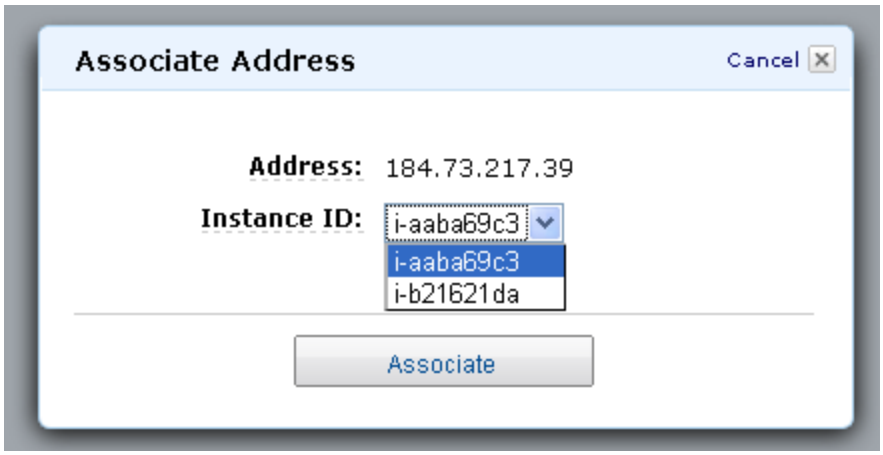

Figure 4 - Elastic IP addresses

Failures Can Be Useful

> *"I'm not a real programmer. I throw together things until it works then I move on. The real programmers will say 'yeah it works but you're leaking memory everywhere. Perhaps we should fix that.' I'll just restart Apache every 10 requests."*
>
> *Rasmus Lerdorf (creator of PHP)*

Though often not readily admitted, the reality is that most software systems will degrade over time. This is due in part to any or all of the following reasons:

1. Software will leak memory and/or resources. This includes software that you have written, as well as software that you depend on (e.g., application frameworks, operating systems, and device drivers).
2. File systems will fragment over time and impact performance.
3. Hardware (particularly storage) devices will physically degrade over time.

Disciplined software engineering can mitigate some of these problems, but ultimately even the most sophisticated software system is dependent on a number of components that are out of its control (e.g., operating system, firmware, and hardware). Eventually, some combination of hardware, system software, and your software will cause a failure and interrupt the availability of your application.

In a traditional IT environment, hardware can be regularly maintained and serviced, but there are practical and financial limits to how aggressively this can be done. However, with Amazon EC2, you can terminate and recreate the resources you need at will.

An application that takes full advantage of the AWS platform can be *refreshed* periodically with new server instances. This ensures that any potential degradation does not adversely affect your system as a whole. In a sense, you are using what would be considered a failure (e.g., a server termination) as a forcing function to refresh this resource.

Using this approach, an AWS application is more accurately defined as the service it provides to its clients, rather than the server instance(s) it is comprised of. With this mindset, the server instances themselves become immaterial and even disposable.

Auto Scaling

The concept of automatically provisioning and scaling compute resources is a crucial aspect of any well-engineered, fault-tolerant application running on the Amazon Web Services platform. Auto Scaling[3] is a powerful option that you can very easily apply to your application.

Auto Scaling enables you to automatically scale your Amazon EC2 capacity up or down. You can define rules that determine when more (or fewer) server instances are needed, such as:

1. When the number of functioning server instances is above (or below) a certain number, launch (or terminate) server instances
2. When the resource utilization (i.e. CPU, network or disk) of the server instance fleet is above (or below) a certain threshold, launch (or terminate) server instances. Such metrics will be collected from the Amazon CloudWatch service, which monitors Amazon EC2 instances.

Auto Scaling enables you to terminate server instances at will, knowing that replacement instances will be automatically launched. Auto Scaling also enables you to add more instances in response to an increasing load; and when those instances are no longer needed, they will be automatically terminated.

These rules enable you to implement a number of traditional redundancy patterns very easily.

For example, 'N + 1 redundancy[4]' is a very popular strategy for ensuring a resource (e.g., a database) is always available. 'N + 1' dictates that there should be N+1 resources operational, when N resources are sufficient to handle the anticipated load.

This approach is ideal for Auto Scaling. To implement N + 1 with Auto Scaling, you simply define a rule that there should always be at least 2 instances of a given AMI available. When used in conjunction with Elastic Load Balancing, each instance would handle a fraction of the incoming load, with sufficient headroom (unused capacity) on each instance to handle the entire load if necessary. If one instance fails, Auto Scaling will immediately launch a replacement, since the minimum threshold of 2 instances was breeched. Auto Scaling will always ensure that you have 2 healthy server instances available.

Since Auto Scaling will automatically detect failures and launch replacement instances, if an instance is not behaving as expected (e.g., it is running with poor performance), you can simply terminate that instance and a new one will be launched.

By using Auto Scaling, you can (and should) regularly turn your instances over to ensure that any leaks or degradation do not impact your application – you can literally set expiry dates on your server instances to ensure they remain 'fresh.'

With an 'N+1' approach, you can also have the additional server accept requests – this enables your application to transition seamlessly in case the primary server fails. The Elastic Load Balancing feature in Amazon EC2 is an ideal way to balance the load amongst your servers.

[3] Auto Scaling is applicable in a number of scenarios; this document will examine how to it specifically towards achieving fault-tolerance.
[4] http://en.wikipedia.org/wiki/N%2B1_redundancy

Elastic Load Balancing

Elastic Load Balancing is an AWS product that distributes incoming traffic to your application across several Amazon EC2 instances. When you use Elastic Load Balancing, you are given a DNS host name – any requests sent to this host name are delegated to a pool of Amazon EC2 instances.

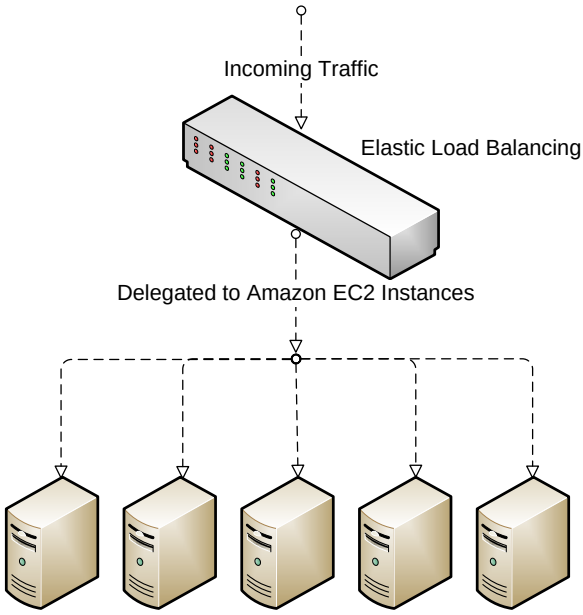

Figure 5 - Elastic Load Balancing

Elastic Load Balancing detects unhealthy instances within its pool of Amazon EC2 instances and automatically reroutes traffic to healthy instances, until the unhealthy instances have been restored.

Auto Scaling and Elastic Load Balancing are an ideal combination – Elastic Load Balancing gives you a single DNS name for addressing and Auto Scaling ensures there is always the right number of healthy Amazon EC2 instances to accept requests.

Regions and Availability Zones

Another key element to achieving greater fault tolerance is to distribute your application geographically. If a single Amazon Web Services datacenter fails for any reason, you can protect your application by running it simultaneously in a geographically distant datacenter.

Amazon Web Services are available in geographic *Regions*. When you use AWS, you can specify the Region in which your data will be stored, instances run, queues started, and databases instantiated. For most AWS infrastructure services, including Amazon EC2, there are five Regions: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore) and Asia Pacific (Japan). Amazon S3 has a slightly different region structure: US Standard, which encompasses datacenters throughout the United States, US West (Northern California), EU (Ireland), Asia Pacific (Singapore) and Asia Pacific (Japan).

Within each Region are *Availability Zones (AZs)*. 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. By launching instances in separate Availability Zones, you can protect your applications from a failure (unlikely as it might be) that affects an entire zone.

Regions consist of one or more Availability Zones, are geographically dispersed, and are in separate geographic areas or countries. The Amazon EC2 service level agreement commitment is 99.95% availability for each Amazon EC2 Region.

Building Multi-AZ Architectures to Achieve High Availability

You can achieve High Availability by deploying your application that spans across multiple Availability Zones. Redundant instances for each tier (e.g. web, application, and database) of an application could be placed in distinct Availability Zones thereby creating a multi-site solution. The desired goal is to have an independent copy of each application stack in two or more Availability Zones.

To achieve even more fault tolerance with less manual intervention, you can use Elastic Load Balancing. You get improved fault tolerance by placing your compute instances behind an Elastic Load Balancer, as it can automatically balance traffic across multiple instances and multiple Availability Zones and ensure that only healthy Amazon EC2 instances receive traffic. You can set up an Elastic Load Balancer to balance incoming application traffic across Amazon EC2 instances in a single Availability Zone or multiple Availability Zones. Elastic Load Balancing can detect the health of Amazon EC2 instances. When it detects unhealthy Amazon EC2 instances, it no longer routes traffic to those unhealthy instances. Instead, it spreads the load across the remaining healthy instances. If all of your Amazon EC2 instances in a particular Availability Zone are unhealthy, but you have set up instances in multiple Availability Zones, Elastic Load Balancing will route traffic to your healthy Amazon EC2 instances in those other zones. It will resume load balancing to the original Amazon EC2 instances when they have been restored to a healthy state.

This multi-site solution is highly available, and by design will cope with individual component or even Availability Zone failures.

The figure below illustrates a highly available system on AWS, which spans two Availability Zones (AZs).

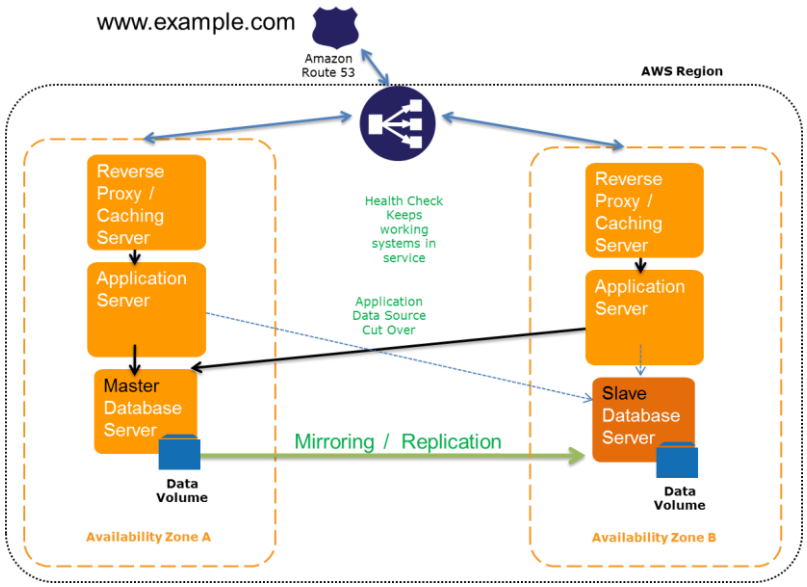

Figure 6: Leverage Elastic Load Balancers and Multi-Availability Zones

Elastic IP Addresses play a critical role in the design of a fault-tolerant application spanning multiple Availability Zones. The failover mechanism can easily re-route the IP address (and hence the incoming traffic) away from a failed instance or zone to a replacement instance.

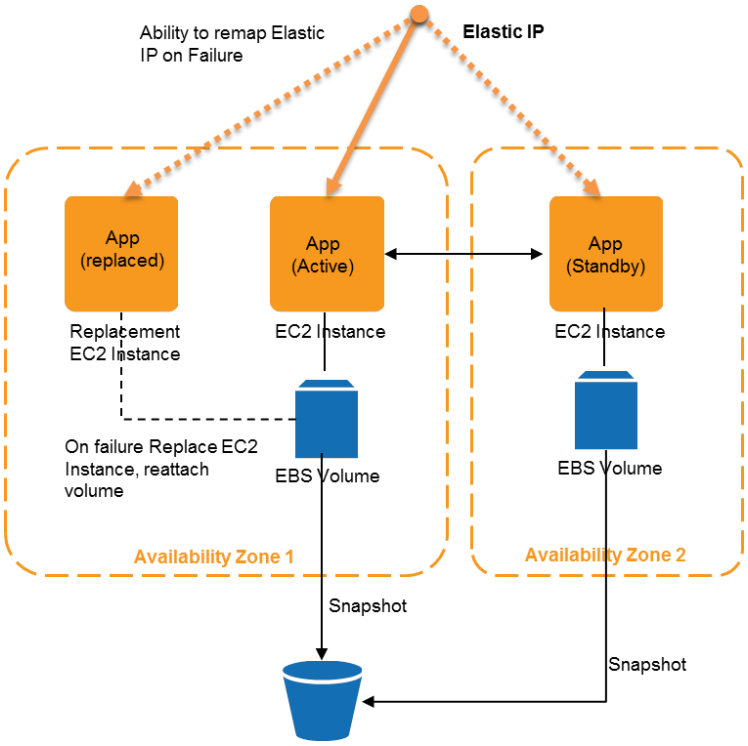

Figure 7: Leverage Elastic IPs and Multi-Availability Zones

Auto Scaling can work across multiple Availability Zones in an AWS Region, making it easier to automate increasing and decreasing of capacity. AWS database offerings, like SimpleDB and Amazon Relational Database Service (Amazon RDS) can help to reduce the cost and complexity of operating a multi-site system. Please refer to the Fault-Tolerant Building Blocks section for more details.

Reserved Instances

All of the techniques examined so far have relied on the assumption that you will be able to procure Amazon EC2 instances whenever you need them.

Amazon Web Services has massive hardware resources at its disposal, but like any cloud computing provider, those resources are finite. The best way for users to maximize their access to these resources is by reserving a portion of the computing capacity that they require. This can be done through a feature called Reserved Instances.

With Reserved Instances, you literally reserve computing capacity in the Amazon Web Services cloud. Doing so enables you to take advantage of a lower price, but more importantly in the context of fault tolerance, it will maximize your chances of getting the computing capacity you need.

Fault-Tolerant Building Blocks

Amazon EC2 and its related features provide a powerful, yet economic platform to deploy and build your applications upon. However, they are just one aspect of Amazon Web Services as a whole.

Amazon Web Services offers a number of other products that can be incorporated into your application development. These web services are implicitly fault-tolerant, so by using them, you will be increasing the fault tolerance of your own applications.

Amazon Simple Queue Service

Amazon Simple Queue Service (SQS) is a highly reliable distributed messaging system that can serve as the backbone of your fault-tolerant application.

Messages are stored in queues that you create – each queue is defined as a URL, so it can be accessed by any server that has access to the Internet, subject to the Access Control List (ACL) of the queue. You can use Amazon SQS to help you ensure that your queue is always available; any messages that you send to a queue are retained for up to four days (or until they are read and deleted by your application).

A canonical system architecture using Amazon SQS is illustrated below.

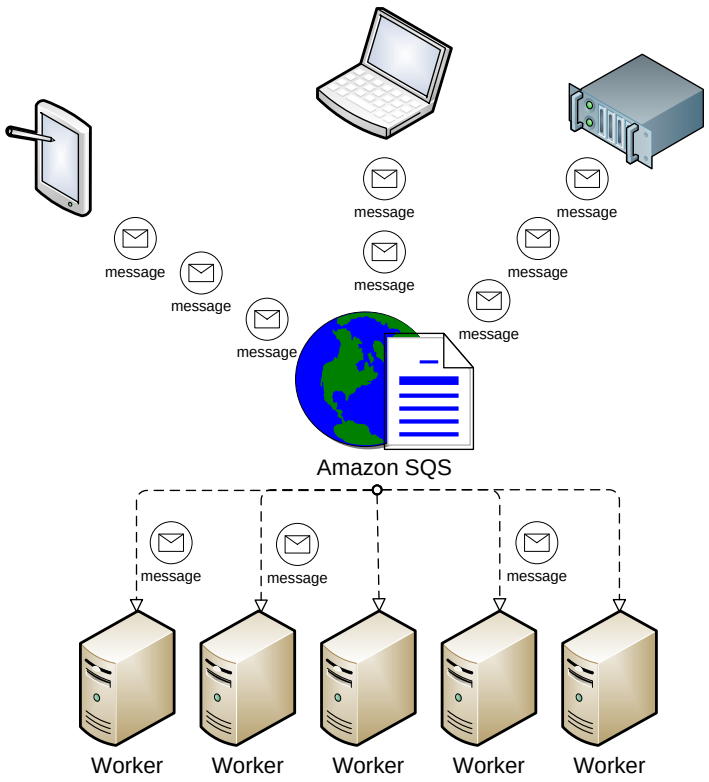

Figure 8 - Amazon SQS System Architecture

In this example, an Amazon SQS queue is used to accept requests. A number of Amazon EC2 instances constantly poll that queue, looking for requests. When a request is received, one of these Amazon EC2 instances will pick up that request and process it. When that instance is done processing the request, it goes back to polling.

If the number of messages in a queue starts to grow or if the average time to process a message becomes too high, you can scale upwards by simply adding more workers on additional Amazon EC2 instances.

It is common to incorporate Auto Scaling to manage these Amazon EC2 instances to ensure that there is an adequate supply of EC2 instances that run 'workers' consuming messages from the queue. Even in an extreme case where all of the worker processes have failed, Amazon SQS will simply store the messages that it receives. Messages are stored for up to four days, so you have plenty of time to launch replacement Amazon EC2 instances.

Once a message has been pulled from an SQS queue, it becomes invisible to other consumers for a configurable time interval known as a *visibility timeout*. After the consumer has processed the message, it must delete the message from the queue. If the time interval specified by the visibility timeout has passed, but the message isn't deleted, it is once again visible in the queue and another consumer will be able to pull and process it. This two-phase model ensures that no queue items are lost if the consuming application fails while it is processing a message.

Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3) is a deceptively simple web service that provides highly durable, fault-tolerant data storage. Amazon Web Services is responsible for maintaining availability and fault-tolerance; you simply pay for the storage that you use.

Behind the scenes, Amazon S3 stores objects redundantly on multiple devices across multiple facilities in an Amazon S3 Region – so even in the case of a failure in an Amazon Web Service data center, you will still have access to your data.

Amazon S3 is ideal for any kind of object data storage requirements that your application might have. Amazon S3 is accessed by URL like Amazon SQS, so any computing resource that has access to the Internet can use it.

Amazon S3's Versioning feature allows you to retain prior versions of objects stored in S3 and also protects against accidental deletions initiated by a misbehaving application. Versioning can be enabled for any of your S3 buckets.

By using Amazon S3, you can delegate the responsibility of one critical aspect of fault-tolerance – data storage – to Amazon Web Services.

Amazon SimpleDB

Amazon SimpleDB is a fault-tolerant and durable structured data storage solution. With Amazon SimpleDB, you can decorate your data with attributes, and query for that data based on the values of those attributes. In many scenarios, Amazon SimpleDB can be used to augment or even replace your use of traditional relational databases such as MySQL or Microsoft SQL Server.

Amazon SimpleDB is highly available for your use, just like Amazon S3 and the other services. By using Amazon SimpleDB, you can take advantage of a scalable service that has been designed for high-availability and fault tolerance. Data stored in Amazon SimpleDB is stored redundantly without single points of failures.

Amazon Relational Database Service

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easy to run relational databases in the cloud. In the context of building fault-tolerant and highly available applications, Amazon RDS offers several features to enhance the reliability of critical databases.

Automated backups of your database enable point-in-time recovery for your database instance. Amazon RDS will back up your database and transaction logs and store both for a user-specified retention period. This feature is enabled by default.

Similar to Amazon EBS snapshots, with Amazon RDS, you can initiate snapshots of your DB Instance. These full database backups will be stored by Amazon RDS until you explicitly delete them. You can create a new DB Instance from a DB Snapshot whenever you desire. This can help you to recover from higher-level faults such as unintentional data modification, either by operator error or by bugs in the application.

Amazon RDS also supports a *Multi-AZ* deployment feature. If this is enabled, a synchronous standby replica of your database is provisioned in a different Availability Zone. Updates to your DB Instance are synchronously replicated across Availability Zones to the standby in order to keep both databases in sync. In case of a failover scenario, the standby is promoted to be the primary and will handle your database operations. Running your DB Instance as a Multi-AZ deployment safeguards your data in the unlikely event of a DB Instance component failure or service health disruption in one Availability Zone.

Conclusion

Amazon EC2 is a natural entry point for your application development; its server instances are conceptually very similar to traditional servers; this greatly reduces the learning curve for developing applications for the cloud. However, using Amazon EC2 server instances in the same manner as traditional hardware server instances is only a starting point – doing so will not greatly improve your fault-tolerance, performance, or even your overall cost.

The complete benefits of the Amazon Web Services platform are realized when you incorporate more features of Amazon EC2, as well as other Amazon Web Services products.

In order to build fault-tolerant applications on Amazon EC2, it's important to follow best practices such as quickly being able to commission replacement instances, using Amazon EBS for persistent storage, and taking advantage of multiple Availability Zones and elastic IP addresses.

Using Auto Scaling enables you to greatly reduce the amount of time and resources you need to monitor your servers – if a failure occurs, a replacement will be automatically launched for you. Diagnosing an unhealthy server can be as simple as terminating it and letting Auto Scaling launch a new one for you.

Elastic Load Balancing enables you to publish a single, well-known end point for your application. The ebb and flow of Amazon EC2 instances launching, failing, being terminated and being re-launched will be hidden from your users.

Amazon SQS, Amazon S3, and Amazon SimpleDB are higher-level building blocks that you can incorporate into your application. These services are excellent examples of how to achieve fault-tolerance, and they in turn increase the fault-tolerance of your application. With Amazon RDS you have easy access to features that enable fault-tolerant database deployments, including automatic backups, snapshots, and Multi-AZ deployments.

Above all, the pricing model of Amazon Web Services gives you the option to experiment – there is no upfront investment, you simply pay for what you use. If a particular aspect of the Amazon Web Services platform turns out not to be suitable for your application, your investment is complete as soon as you stop using it.

The power, sophistication, and economic transparency offered by Amazon Web Services provide you with an unparalleled platform upon which to build your fault-tolerant software.

Further Reading

1. **Best Practices for using Elastic IPs and Availability Zones** - http://support.rightscale.com/09-Clouds/AWS/02-Amazon_EC2/Designing_Failover_Architectures_on_EC2/00-Best_Practices_for_using_Elastic_IPs_%28EIP%29_and_Availability_Zones

2. **Setting up Fault-tolerant site using Amazon's Availability Zones** - http://blog.rightscale.com/2008/03/26/setting-up-a-fault-tolerant-site-using-amazons-availability-zones/

3. **Scalr** - https://scalr.net/login.php

4. **Creating a virtual data center with Scalr and Amazon Web Services** - http://scottmartin.net/2009/07/11/creating-a-virtual-datacenter-with-scalr-and-amazon-web-services/

5. **Amazon Elastic Load Balancing** - http://aws.amazon.com/elasticloadbalancing/

6. **Auto** Scaling Service – http://aws.amazon.com/autoscaling

7. **Instance Types** - http://aws.amazon.com/ec2/instance-types/

8. **Elastic Block Store** - http://aws.amazon.com/ebs/

9. **Amazon Machine Images Resources** - http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171

10. **Amazon Relational Database Service** - http://aws.amazon.com/rds/