클라우드 설계 및 구축: 모범 사례

2010 년 1 월

*최근 업데이트 - **2011 년 1 월***

Jinesh Varia
jvaria@amazon.com

서론

소프트웨어 설계자들은 최근 몇 년 동안 확장성이 뛰어난 애플리케이션 구축을 위해 여러 가지 기술 개념 및 모범 사례를 개발하여 구현해 왔습니다. "테라바이트급 정보 시대" 를 맞이하여, 데이터세트의 지속적인 증가와 예측하기 어려운 트래픽 패턴, 응답 시간 단축 요구 등으로 인해 이러한 기술 개념의 필요성이 더욱 커지고 있습니다. 본 백서에서는 이러한 기존의 기술 개념 가운데 일부를 재해석, 보완하여, 클라우드 컴퓨팅 측면에서 발전시켜 나갈 수 있는 방안들을 소개하며 아울러, 클라우드가 갖추고 있는 동적인 특성으로 인해 새롭게 부상하고 있는 탄력성과 같은 새로운 개념에 대해서도 논의하고자 합니다.

본 백서는 엔터프라이즈급 애플리케이션을 고정적인 물리적 환경에서 가상 클라우드 환경으로 마이그레이션하고자 하는 *클라우드 설계자*들을 대상으로 작성되었습니다. 따라서 새로운 *클라우드 애플리케이션*을 개발하거나 기존의 애플리케이션을 클라우드로 마이그레이션하고자 할 때 필요한 기술 개념, 원칙, 모범 사례를 집중적으로 소개하고 있습니다.

배경

클라우드 설계자로서 클라우드 컴퓨팅의 장점을 이해하는 것은 매우 중요합니다. 본 절에서는 클라우드 컴퓨팅의 몇 가지 경영 및 기술적 측면의 장점과 아울러, 현재 이용 가능한 AWS 서비스에 대해 설명합니다.

클라우드 컴퓨팅이 기업 경영에 미치는 장점

기업이 클라우드에 애플리케이션을 구축할 경우 누릴 수 있는 혜택은 다음과 같습니다.

초기 인프라 투자 비용이 거의 없음: 대규모 시스템을 구축해야 하는 경우 부동산 구입비, 물리적 보안 유지 비용, 하드웨어(랙, 서버, 라우터, 백업 전원 공급 장치) 설치비에 하드웨어 관리(전원 관리, 냉각)와, 운영 직원의 인건비에 이르기까지 막대한 비용이 소요될 수 있습니다. 높은 초기 투자 비용 때문에 프로젝트를 시작하려면 여러 차례 경영진의 승인을 얻어야 하는 것이 일반적이었습니다. 하지만 이제는 공공시설 형태의 클라우드 컴퓨팅 덕분에 고정 비용이나 초기 투자비가 필요하지 않습니다.

수요에 따른 인프라 구축: 과거에는 고객 기업의애플리케이션 수요가 증가하는데, 인프라와 지원 시스템이 이 수요에 부응하지 못하는 경우, 성공을 거둘 수 없었습니다. 반면, 인프라와 시스템에 막대한 금액을 투자하였으나 기대만큼 수요가 발생하지 못하는 경우에도 역시 실패를 피할 수 없었습니다. 필요에 따라 자체 설정이 가능하도록 클라우드 내에 애플리케이션을 구축할 경우에는 대규모 시스템 용량을 미리 구입하여 구축할 필요가 없습니다. 이러한 솔루션은 *성장* 속도에 비례하는 확장과 사용량에 준하는 요금 지불 방식으로 탄력성을 높이며, 리스크 완화 및 운용 비용 절감에도 기여합니다.

보다 효율적인 리소스 활용: 일반적으로 시스템 관리자들은 (주어진 용량이 소진되었을 때) 하드웨어의 추가 구매 또는 (대기 용량이 과다한 상태일 때) 인프라의 이용률 향상 방안에 대해 고민합니다. 하지만, 클라우드를 적용할 경우, 애플리케이션의 필요에 따라 리소스를 추가로 요청하거나 취소하는 방법으로 보다 효과적이고 효율적으로 리소스를 관리할 수 있습니다.

사용량에 따른 비용 책정: 공공요금 형태의 요금 방식으로, 사용한 인프라에 대해서만 요금이 부과되며, 인프라를 할당받았으나 사용하지 않은 경우에는 요금을 지불하지 않습니다. 이를 통해 새로운 차원의 비용 절감 효과를 기대할 수 있습니다. 최적화 패치를 사용하여 기존에 보유하고 있는 클라우드 애플리케이션을 업데이트할 경우 즉시(때로는 익월 결제 시) 비용 절감 효과를 확인할 수 있습니다. 예를 들어, 캐싱 계층에서 사용자의 데이터 요구를 70%까지 줄일 수 있을 경우, 절감 효과가 즉시 발생하여 익월 결제 시 바로 요금이 줄어든 것을 확인할 수 있습니다. 더욱이, 플랫폼을 클라우드에서 구축하는 경우 이와 같이 유연하고 가변적인 사용량 기준의 요금 체제를 고객들에게도 확대 적용할 수 있습니다.

시장 출시 기간 단축: 병렬화 방식으로 처리 속도를 가장 크게 높일 수 있습니다. 병렬로 실행될 수 있는 컴퓨팅 또는 데이터 중심의 작업을 컴퓨터 한 대에서 처리할 때 500 시간이 걸린다면 클라우드 아키텍처[6]를 사용할 때에는 500 개 인스턴스를 생성하여 작업을 수행하므로 같은 양의 작업을 단 1 시간 내에 처리할 수 있습니다. 탄력적인 인프라 구축을 통해 경제적인 방법으로 병렬화를 수행하여 애플리케이션 출시 기간을 단축시킬 수 있습니다.

클라우드 컴퓨팅의 기술적 장점

클라우드 컴퓨팅의 기술적 장점은 다음과 같습니다.

자동화 – "스크립터블 인프라": (API 를 이용하여) 프로그램 가능한 인프라를 활용하여 반복 가능한 구축 및 배포 시스템을 만들 수 있습니다.

Auto-scaling: 운용자의 개입 없이 예기치 않은 수요에 맞게 애플리케이션 규모를 늘리거나 줄일 수 있습니다. Auto-scaling 기능을 통해 자동화를 촉진하고 효율성을 더욱 높입니다.

능동적 확장 및 축소: 고객의 트래픽 패턴을 고려해 적절한 계획 하에 예상 수요량에 맞게 애플리케이션 규모를 늘리거나 줄임으로써 이에 따른 비용을 낮게 유지시켜 줍니다.

보다 효율적인 개발 주기: 생산 시스템을 개발 및 테스트 환경에 사용할 수 있도록 쉽게 복제할 수 있습니다. 단계별 환경 구성을 통해 쉽게 생산을 진행할 수 있습니다.

시험 기능 개선: 테스트용 하드웨어를 항상 충분히 지원합니다. 개발 과정 중 각 단계 별로 자동으로 시험을 실시합니다. 시험이 진행되는 기간 동안에만 사전 설정한 환경으로 "임시 시험실"을 구축할 수 있습니다.

재해 복구 및 비즈니스 연속성: 클라우드는 저렴한 비용으로 일정 규모의 DR 서버와 데이터 스토리지를 유지할 수 있는 옵션을 제공합니다. 클라우드를 사용하여 지리적 분산 효과와 아울러 수 분 이내에 다른 지점으로 해당 환경을 복제할 수 있습니다.

과도한 트래픽을 클라우드로 라우팅: 단 몇 번의 클릭과 효과적인 로드 밸런싱 기법을 이용하여, 초과 트래픽을 클라우드로 라우팅하는 방법으로 애플리케이션의 폭주를 완벽하게 예방할 수 있습니다.

Amazon Web Services 클라우드 개요

Amazon Web Services (AWS) 클라우드는 높은 신뢰성과 확장성을 갖춘 인프라를 최소한의 지원 및 관리 비용으로 웹 규모의 솔루션 구축을 위해 제공하며, 개별 건물 또는 데이터센터 내에 설치되어 있는 고객 인프라에 비해 높은 유연성을 제공합니다.

AWS 는 현재 다양한 인프라 서비스를 제공합니다. 아래 그림에서는 AWS 관련 용어 및 고객 애플리케이션과 다양한 Amazon Web Services 연결 방법 및 서비스 간 연계 방법 등을 제시하고 있습니다.

Amazon Elastic Compute Cloud (Amazon EC2)[1]는 클라우드에서 컴퓨팅 용량을 자유 자재로 변경할 수있는 컴퓨팅 파워를 제공하는 웹 서비스입니다. 사용자가 운영 체제, 애플리케이션 소프트웨어 및 관련 구성 설정 값을 *Amazon 머신 이미지*(AMI)에 번들로 함께 구성할 수 있습니다. 이러한 AMI 를 사용하여 다수의 가상 *인스턴스*들을 제공할 수 있을 뿐 아니라 용량 요구량이 변경될 때마다 간단한 웹 서비스 호출을 통해 인스턴스를 해체하여 신속하게 용량 규모를 조정할 수도 있습니다. 시간 단위로 인스턴스 요금을 지불하는 *온 디맨드 인스턴스(On-Demand Instances)* 또는 저렴한 금액을 일시불로 지불하면 온 디맨드 인스턴스보다도 더 저렴하게 인스턴스를 실행할 수

[1] Amazon EC2 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/ec2)를 참조하십시오.

있는 *예약 인스턴스(Reserved Instances)*를 구입하거나 사용하지 않는 용량에 입찰하여 비용을 더욱 더 줄일 수 있는 *스팟 인스턴스(Spot Instances)*를 구입할 수 있습니다. 인스턴스를 한 개 이상의 *지역*에서 시작할 수 있습니다. 각 지역(region)에는 여러 개의 *가용 영역(Availability Zones)*이 있습니다. 이러한 가용 영역은 다른 가용 영역에 장애가 발생할 경우 이 장애 발생 영역에서 분리되도록 설계된 개별 지점들이며, 동일 지역 내에 있는 다른 가용 영역들에 저렴하고, 지연 시간이 짧은 네트워크 연결을 제공합니다.

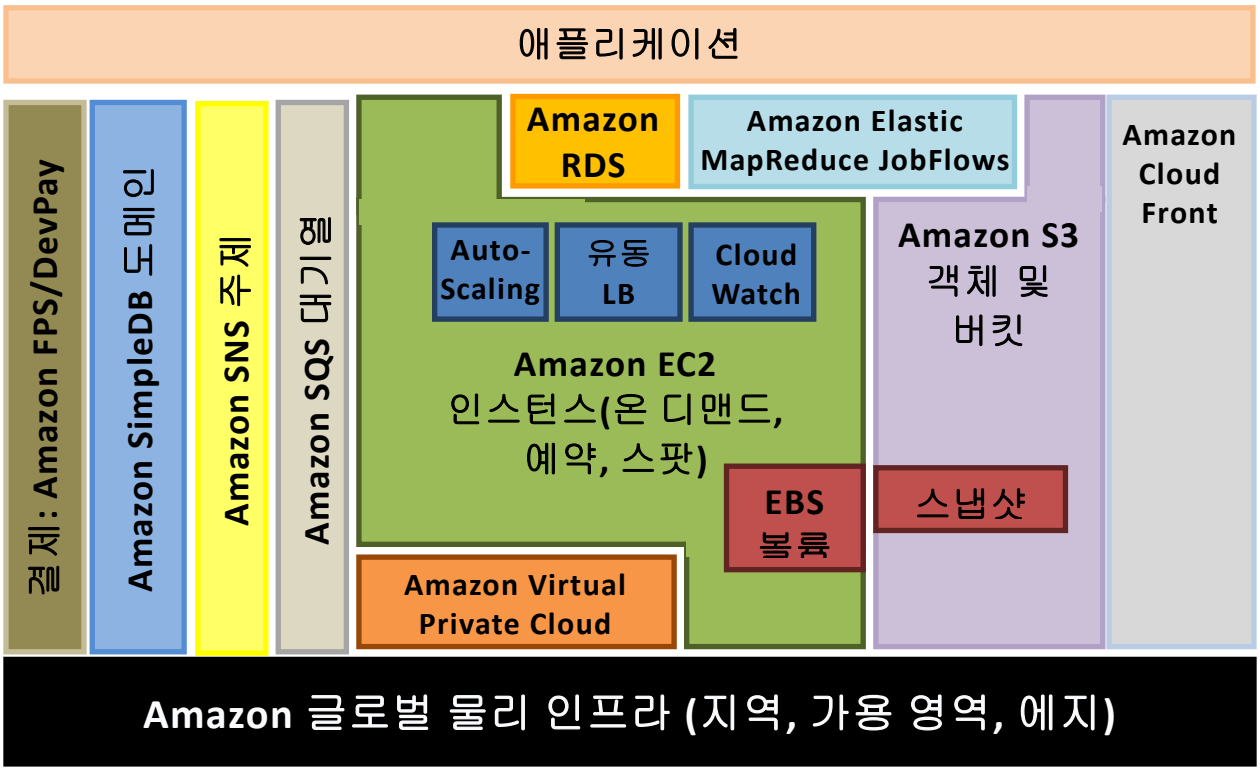

그림 1: Amazon Web Services

엘라스틱 IP 주소할당 기능을 이용하여 고정 IP 주소를 인스턴스마다 프로그래밍 방식으로 할당할 수 있습니다. Amazon CloudWatch[2] 를 사용하여 Amazon EC2 인스턴스에 대한 모니터링 기능을 통해 리소스 이용률, 운용 성능, 전반적인 수요 패턴(CPU 사용률, 디스크 읽기/쓰기, 네트워크 트래픽 등의 메트릭 포함)을 파악할 수 있습니다. Auto Scaling 기능[3]을 사용하여 *Auto Scaling 그룹*을 만들어 Amazon CloudWatch 가 수집하는 메트릭에 따라 특정 조건에서 용량을 자동으로 확장할 수 있습니다. Elastic Load Balancing[4] 서비스를 사용하여 *유연한 로드 밸런서를 생성하여 인입 트래픽을 분배할 수 있습니다. Amazon* Elastic Block Storage (EBS)[5] *볼륨*은 네트워크에 연결된 영구 스토리지를 Amazon EC2 인스턴스에 제공합니다. EBS 볼륨에 대해 일관성있는 PIT (Point in Time) *스냅샷*을 생성하여 Amazon Simple Storage Service (Amazon S3)[6] 에 저장할 수 있습니다.

---

[2] Amazon CloudWatch 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/cloudwatch/)를 참조하십시오.
[3] Auto Scaling 기능에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/auto-scaling)를 참조하십시오.
[4] Elastic Load Balancing 기능에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/elasticloadbalancing)를 참조하십시오.
[5] Elastic Block Store 에 대한 자세한 사항는 웹 사이트(http://aws.amazon.com/ebs)를 참조하십시오.
[6] Amazon S3 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/s3)를 참조하십시오.

Amazon S3 는 지속성이 뛰어난 분산형 데이터 저장장치입니다. 간편한 웹 서비스 인터페이스를 통해 웹에서 언제, 어디에서나 대량의 데이터를 표준 HTTP 동사를 사용하여 *버킷*(컨테이너) 내에 *객체*로 저장하거나 이를 검색할 수 있습니다. 객체의 복사본은 (정적 및 스트리밍 콘텐츠)콘텐츠 전송을 위한 웹 서비스인 Amazon CloudFront[7] 서비스를 통해 만들어진 *배포 (distribution)* 기능을 사용하여 전 세계 14 개의 *에지*에서 배포 및 캐싱할 수 있습니다. Amazon SimpleDB[8]는 복잡한 운용 방식 없이 실시간 데이터베이스 쿼리 기능 및 구조화된 데이터를 쉽게 쿼리할 수 있는 기능 등 데이터베이스의 핵심 기능을 제공하는 웹 서비스입니다. 데이터 세트를 *도메인*에 구성하여 특정 도메인에 저장된 모든 데이터를 조회할 수 있습니다. 도메인은 *속성-값 쌍*으로 정의되는 *항목*들의 모음입니다.

Amazon Relational Database Service[9] (Amazon RDS)는 관계형 데이터베이스를 클라우드에서 설정, 운용,확장할 수 있는 간단한 방법을 제공합니다. *DB 인스턴스*를 실행하여 다기능 MySQL 데이터베이스에 액세스할 수 있으며 백업, 패치 관리 등의 일반적인 데이터베이스 관리 작업들도 간단하게 수행할 수 있습니다.

Amazon Simple Queue Service (Amazon SQS)[10] 는 컴퓨터와 애플리케이션 구성 요소 간에 전송되는 *메시지*를 저장하기 위한 안정적이고 확장성이 뛰어난 호스팅된 분산 대기열을 제공합니다.

Amazon Simple Notifications Service (Amazon SNS)는 [11] *주제*를 생성하고 발행-구독 프로토콜을 사용하여 클라우드에서 애플리케이션 또는 사람에게 해당 주제에 대해 알릴 수 있는 간편한 방법을 제공합니다.

Amazon Elastic MapReduce[12] 는 Amazon Elastic Compute Cloud (Amazon EC2) 및 Amazon Simple Storage Service(Amazon S3)에서 웹 규모의 인프라에서 실행되는 호스팅된 Hadoop 프레임워크를 제공하며, 이를 통해 맞춤형 *JobFlows* 를 생성할 수 있습니다. JobFlow 는 일련의 MapReduce *단계*를 가리킵니다.

Amazon Virtual Private Cloud (Amazon VPC)[13] 를 이용하면 AWS 에 포함된 사설 클라우드로 기업 네트워크를 확장할 수 있습니다. Amazon VPC 는 IPSec 터널 모드를 사용하여 고객 데이터 센터 내에 있는 게이트웨이와 AWS 의 게이트웨이를 안전하게 연결합니다.

Amazon Route53 는 확장성이 뛰어난 DNS 서비스로서, 고객이 관리하고자 하는 모든 도메인에 *HostedZone* 을 생성하여 DNS 레코드를 관리할 수 있습니다.

AWS Identity and Access Management (IAM)[14] 을 이용하면 하나의 보안 자격 증명으로 다수의 사용자를 생성하며, 고객의 AWS 계정 내에서 이들 사용자 각각에 대한 액세스 허가 여부를 관리할 수 있습니다. IAM 은 기본적으로 AWS 서비스에 포함되어 있습니다. 어떤 서비스 API 도 IAM 을 지원하기 위해 변경되지 않았으며, AWS 서비스 API 를 이용해 구축된 기존의 애플리케이션 및 도구들은 IAM 을 사용하는 경우에도 계속 기능을 유지합니다.

AWS 는 또한 Amazon 의 결제 인프라를 활용하는 다양한 지불 및 요금 청구 서비스[15] 를 제공합니다.

---

[7] Amazon CloudFront 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/cloudfront)를 참조하십시오.
[8] Amazon SimpleDB 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/simpledb)를 참조하십시오.
[9] Amazon RDS 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/rds)를 참조하십시오.
[10] Amazon SQS 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/sqs)를 참조하십시오.
[11] Amazon SNS 에 대한 자세한 사항은 다음 주소에서 확인할 수 있습니다 http://aws.amazon.com/sns
[12] Amazon ElasticMapReduce 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/elasticmapreduce)를 참조하십시오.
[13] Amazon Virtual Private Cloud 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/vpc)를 참조하십시오.
[14] Amazon Identity and Access Management 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/iam)를 참조하십시오.
[15] Amazon Flexible Payments Service 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/fps)를 참조하십시오. Amazon DevPay 에 대한 사항은 웹 사이트(http://aws.amazon.com/devpay)를 참조하십시오.

모든 AWS 인프라 서비스는 장기 약정이나 계약이 필요 없는 공공 서비스 요금 형태의 요금제를 지원합니다. 예를 들어, Amazon EC2 에서는 인스턴스 사용 요금을 시간 단위로 지불하며, Amazon S3 에서는 기가바이트 단위로 스토리지 및 데이터 전송 요금을 지불합니다. 이러한 서비스 및 종량제 요금에 대한 자세한 사항은 AWS 웹 사이트에서 확인할 수 있습니다.

AWS 클라우드를 사용하는 경우에도 유연성과 아울러 다음과 같이 이미 사용자에게 익숙한 컨트롤 기능을 그대로 유지할 수 있습니다.

- 프로그래밍 모델, 언어 또는 운영 체제(Windows, OpenSolaris 또는 다양한 Linux 제품)를 원하는 대로 자유롭게 사용할 수 있습니다.
- 고객의 요구에 부합하는 AWS 제품을 자유롭게 선택할 수 있으며, 어떤 서비스든 개별적으로 또는 통합하여 사용할 수 있습니다.
- AWS 는 규모 조절이 가능한 리소스(스토리지, 대역폭 및 컴퓨팅)을 제공하므로, 원하는 만큼 자유롭게 소비할 수 있으며 소비한 만큼만 지불하면 됩니다.
- 과거에 사용한 시스템 관리 도구도 자유롭게 사용할 수 있으며, 데이터 센터를 클라우드로 확장할 수 있습니다.

클라우드 개념

클라우드는 확장성이 뛰어난 인터넷 아키텍처[13]를 구축하고자 하는 기존 개념을 보완 강화하는 한편, 애플리케이션들의 구축 및 배치 방법을 전혀 새롭게 바꾸고자 하는 새로운 개념을 도입합니다. 따라서, 개념을 실행에 옮기는 과정에서 마치 "모든 것이 바뀌었지만, 달라진 것이 아무것도 없다"는 느낌을 받게 될 것입니다. 클라우드는 여러 프로세스, 형식, 관행, 철학들을 바꾸고, 이미 알려진 기존의 서비스 지향형 아키텍처(SOA)의 원칙들 가운데 일부분들은 그 어느 때보다 중요한 부분이므로 이를 더욱 더 강화하였습니다. 이 절에서는 이와 같이 새로운 클라우드 개념과 아울러 다시 주목을 받게 된 SOA 개념 가운데 몇 가지를 살펴보고자 합니다.

기존의 애플리케이션들은 개발 당시 의도했던 경제성과 구조적인 측면들을 염두에 두고 구축되었습니다. 클라우드에는 고객이 반드시 이해해야 할 몇 가지 새로운 철학 개념들이 도입되었으며, 아래 절에서 그러한 내용을 설명하고자 합니다.

확장 가능한 아키텍처 구축

확장 가능한 인프라의 혜택을 누리기 위해서는 확장 가능한 아키텍처를 구축하는 것이 중요합니다.

클라우드는 이론상 무한 확장성이 가능하도록 설계되었습니다. 하지만, 고객이 보유하고 있는 아키텍처의 확장이 불가능한 경우에는 인프라 확장 혜택을 누릴 수가 없으므로, 이 두 가지 요소가 조화를 이루어야 합니다. 고객의 아키텍처에서 단일 구조의 구성 요소와 아키텍처의 병목 지점을 파악하고, 이 아키텍처에서 온 디맨드 설정 기능을 활용할 수 없는 분야를 파악하며, 확장 가능한 인프라를 이용하고 클라우드의 혜택을 누리기위해 고객의 애플리케이션을 *리팩토링*할 수 있어야 합니다.

진정한 의미의 확장형 애플리케이션이 갖는 특징은 다음과 같습니다.

- 리소스의 양을 증가시키면 성능도 이에 비례하여 증가합니다.
- 확장형 서비스는 이질성(heterogeneity) 문제를 해결할 수 있습니다.
- 확장형 서비스는 운용이 효율적입니다.
- 확장형 서비스는 탄력적입니다.
- 확장형 서비스는 서비스가 증가할 때 더 비용 효율적입니다(단위 비용이 단위 수 증가에 따라 감소함).

이와 같은 특징이 고객 애플리케이션에 내재되어야 하며, 위의 특성을 염두에 두고 아키텍처를 설계할 경우 아키텍처와 인프라가 조화를 이루어 고객이 원하는 확장성을 달성할 수 있을 것입니다.

탄력성의 이해

아래의 그래프에서는 클라우드 설계자가 수요 충족을 위해 애플리케이션의 규모를 조절하고자 할 때 취할 수 있는 몇 가지 방법을 보여주고 있습니다.

수직 확장형 접근법: 확장형 애플리케이션 아키텍처를 염두에 두지 않고 수요 충족을 위해 보다 더 크고, 강력한 컴퓨터(수직적 확장)에 크게 투자합니다. 이러한 방법은 일반적으로 어느 정도까지는 효과가 있으나 엄청난 비용이 발생하거나(아래 그림의 "막대한 자본금(huge capital expenditure)" 참조) 또는 새로 "거대한 설비"를 구축하기도 전에 수요가 지원 용량을 초과할 수 있습니다(아래 그림의 "고객 상실 부분(You just lost your customers)" 참조).

기존의 수평적 확장 방법: 수평으로 확장 가능한 아키텍처를 만들고 소량씩 투자합니다. 대부분의 기업 및 대규모 웹 애플리케이션은 이러한 방식에 따라 애플리케이션 구성 요소를 배분하며 데이터 세트를 연합하고 서비스 중심형 디자인을 채택합니다. 이러한 방법이 때로는 수직적 확장 방법보다 더 효율적입니다. 하지만, 그럼에도 불구하고, 정기적으로 수요를 예측하고 이러한 수요를 충족하기 위해 소규모 단위로 인프라를 구축해야 합니다. 이로 인해 종종 용량 초과("현금 낭비")나 지속적인 수동 모니터링("인력 낭비")이 발생하기도 합니다. 또한 수평적 확장 방법은 애플리케이션에 일시적으로 과부하('슬래시닷 효과'라고도 함)가 발생하는 경우에는 효과를 발휘할 수 없습니다[16].

참고: 두 방법 모두 초기 설치 비용이 있으며, 본질적으로 반응적 특성을 가집니다.

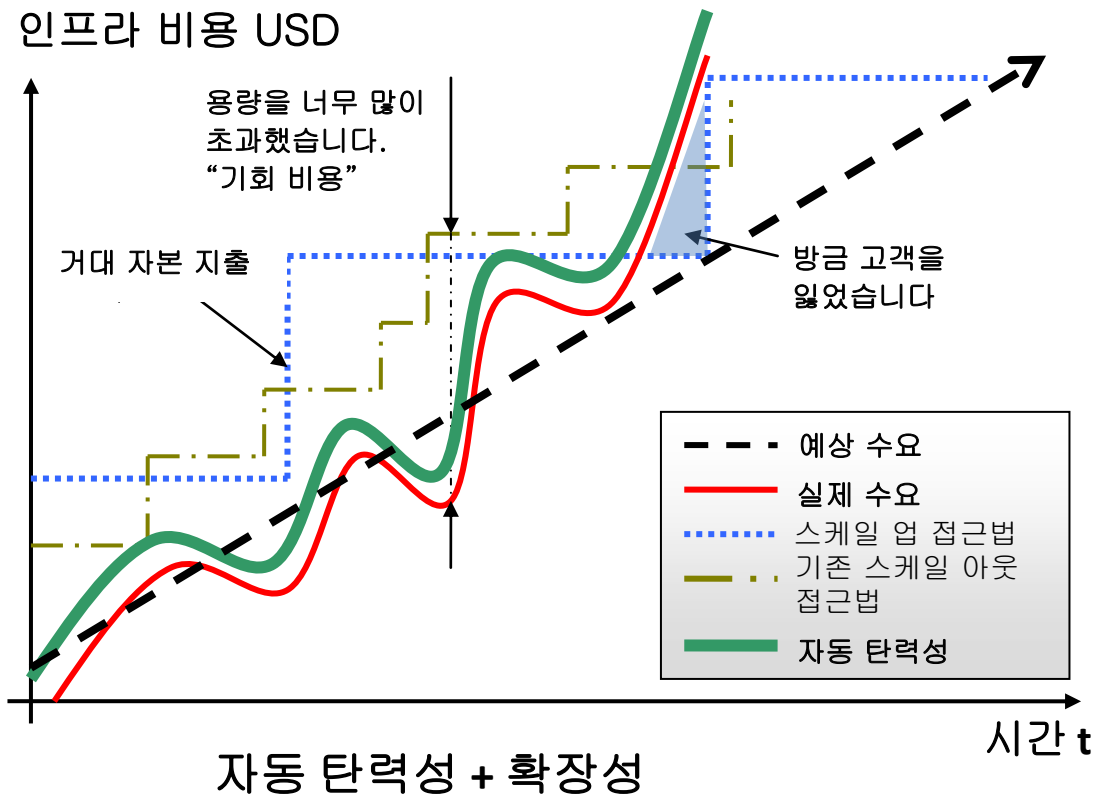

그림 2: 자동 탄력성 조절

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

기존의 인프라는 일반적으로 고객의 애플리케이션에서 차후 수 년간 사용할 컴퓨팅 리소스의 양을 예측해야 합니다. 리소스의 양을 너무작게 잡으면 애플리케이션에 예기치 않은 트래픽을 처리할 용량이 없어 잠재적으로 고객 불만을 초래하게 되며, 반대로 너무 높게 잡으면 과도한 리소스로 인해 비용이 낭비됩니다.

그러나 탄력적인 온 디맨드 *클라우드*(자동 탄력 조절)를 사용하면 인프라를 실제 수요량과 거의 일치하게 조정(확장 및 축소 가능)할 수 있으며, 이를 통해 전반적인 사용률은 높이고 비용은 줄일 수 있습니다.

탄력성은 클라우드가 제공하는 기본적인 속성 중 하나입니다. 탄력성은 마찰은 최소화하면서 컴퓨팅 리소스 규모를 쉽게 확장하거나 축소할 수 있는 기능입니다. 이러한 탄력성이야말로 궁극적으로 클라우드가 제공하는 장점들 중 가장 중요한 부분에 해당함을 이해하는 것이 중요합니다. 여러분은 클라우드를 설계할 때, 이러한 개념을 충분히 이해하고, 이를 애플리케이션 아키텍처 내에 구현하여 클라우드의 혜택을 극대화해야 합니다.

과거에는 애플리케이션을 고정적이고 경직되어 있으며, 설정이 미리 이루어진 인프라에 구축했습니다. 기업들은 매일 서버를 설정하거나 설치할 필요가 없었으며, 대부분의 소프트웨어 아키텍처는 신속한 배포 또는 하드웨어 절감 등의 문제를 고민할 필요가 없습니다. 새로운 리소스를 설치하거나 새로 투자하는 데에는 많은 시간과 비용이 소모되므로, 소프트웨어 설계자들은 하드웨어 이용률을 극대화하기 위해 시간과 리소스를 투자하지 않습니다. 애플리케이션을 구동하는 하드웨어의 활용도가 최대 용량에 못 미치는 경우에도 이를 불가피한 것으로 간주했습니다. 수 분 이내에 새로운 리소스를 구한다는 생각 자체가 불가능했으므로,

아키텍처 내에서 "탄력성"이라는 개념이 간과되었던 것입니다. 클라우드에서는 이러한 사고 방식을 바꿔야 합니다. 클라우드 컴퓨팅에서는 필요한 리소스 확보 과정을 합리화하여, 더 이상 사용하지 않는 하드웨어를 미리 주문하여 보관할 필요가 없습니다. 대신 클라우드 설계자들은 클라우드가 제공하는 광대한 규모와 빠른 응답 시간을 활용하여 필요한 리소스를 몇 분 내로 주문하거나 심지어 구매 과정을 자동화할 수 있습니다. 사용하지 않았거나 또는 사용하고 남은 리소스를 반납하는 데에도 동일한 효과가 적용될 수 있습니다.

고객의 애플리케이션 아키텍처에서 이러한 변화를 수용하지 못하고 탄력성을 발휘하지 못한다면 클라우드의 장점을 충분히 활용하지 못할 수도 있습니다. 클라우드 설계자라면 창의적으로 사고하며, 자신의 애플리케이션 내에 탄력성을 부여하는 방법에 대해서도 연구해야 합니다. 예를 들어, 매일 밤 야간 구축(build) 작업을 수행하고 새벽 2 시부터 2 시간 동안 리그레션 시험 및 단위 시험을 수행(종종 "QA/빌드 박스"라 함)하며 나머지 시간은 유휴 상태인 인프라가 있다고 가정할 때, 탄력적인 인프라를 이용하면, "살아있는(alive)" 박스에서 심야 구축 작업을 실시하고, 심야 2 시간 작업분에 대해서만 비용을 지불하면 됩니다. 마찬가지로, 하루 동안의 수요량을 충족하기 위해 항상 최대 용량(5 개 서버, 연중 무휴 상시 작동)으로 실행되던 내부 장애 티켓 발행용 웹 애플리케이션을 트래픽 패턴에 따라 온 디맨드로(오전 9 시부터 오후 5 시까지 서버 5 개, 오후 5 시에서 오전 9 시까지 서버 2 개)로 운용하도록 선택할 수 있습니다.

탄력적인 지능형 클라우드 아키텍처를 설계하여 인프라를 필요할 때에만 사용하는 것 자체가 놀라운 기술입니다. 탄력성이란 구조적인 설계 요건 또는 시스템 속성 중 하나여야 합니다. 그렇다면, 내 애플리케이션 아키텍처의 어떤 구성 요소 또는 계층에 탄력성을 부여할 수 있을까? 구성 요소를 *탄력적*으로 구성하는 데 시간은 얼마나 걸리나? 전체 시스템 아키텍처에 탄력성을 구현할 경우 어떠한 효과를 기대할 수 있을까?와 같은 질문들을 스스로에게 던져볼 수 있을 것입니다.

다음 절에서는 고객 애플리케이션에 탄력성을 구현하는 데 필요한 몇 가지 특별한 기법을 소개하고자 합니다. 클라우드의 장점을 효과적으로 발휘하려면 이러한 기법을 반드시 고려할 필요가 있습니다.

제약 사항들에 구애받지 않음

클라우드로 애플리케이션을 이동시켜 클라우드에서 이용 가능한 시스템에 고객 시스템 사양을 매핑할 때 클라우드가 고객이 보유하고 있는 리소스의 규격과 정확하게 일치하지 않을 수 있음을 염두에 두어야 합니다. 예를 들어, "클라우드가 서버에서 원하는 만큼의 RAM 용량을 제공하지 않는다"거나 "내 데이터베이스가 인스턴스 한 개에서 얻을 수 있는 것보다 더 많은 IOPS 를 필요로 하는" 경우가 있을 수 있습니다.

클라우드가 *가상 리소스*를 제공하며, 온 디맨드 설정 모델과 결합해야만 이 리소스가 영향력을 발휘한다는 사실을 이해해야 합니다. 클라우드 환경에서 고객의 하드웨어 사양과 동일한 사양을 구하지는 못하더라도, 이보다 더 많은 리소스를 클라우드에서 활용할 수 있으므로, 클라우드 리소스를 사용할 때 우려하거나 제약을 느낄 필요가 없습니다.

예를 들어, 클라우드의 서버에서 원하는 RAM 용량 이상을 제공하지 못할 경우, *맴 캐시드 (memcached)*와 같은 분산형 캐시를 사용하거나[17] 여러 대의 서버로 데이터를 분할할 수 있습니다. 데이터베이스가 더 많은 IOPS 를 필요로하거나, 클라우드에 직접 매핑하지 않을 경우, 고객의 데이터 유형과 사용 방법에 따라 몇 가지 방법을 선택할 수 있습니다. 읽기 중심의 애플리케이션인 경우, 동기화된 슬레이브들에 읽기 부하를 분산할 수 있습니다. 또는 *샤딩(sharding)* [10] 알고리즘을 사용하여 필요할 때 데이터를 라우팅하거나 다양한 데이터베이스 클러스터링 솔루션을 사용할 수도 있습니다.

요컨대, 유연성과 온 디맨드 설정 기능을 결합시킬 경우 실질적으로 제약조건이 해소되어 궁극적으로 시스템의 확장성 및 전체 성능을 개선할 수 있다는 사실을 깨닫게 될 것입니다.

가상 관리

클라우드의 출현으로 시스템 관리자의 역할이 "가상 시스템 관리자"로 바뀌었습니다. 이는 가상 시스템 관리자가 애플리케이션에 대해 보다 상세하게 이해하고, 업무 차원에서 최선의 방법을 선택할 수 있게 됨에 따라 이들이 수행하는 일상적인 작업이 더욱 큰 의미를 가지게 되었음을 의미합니다. 시스템 관리자는 더 이상 서버를 설정하고, 소프트웨어를 설치하며, 네트워크 장치를 연결할 필요가 없습니다. 이러한 모든 번거로운 작업이 이제는 단 몇 번의 클릭과 명령 실행으로 해결되기 때문입니다. 클라우드에서는 이러한 인프라를 프로그래밍하여 자동화할 수 있습니다. 시스템 관리자는 기술력을 한 단계 높여, 스크립트를 사용하여 가상 클라우드 리소스를 관리하는 방법을 익혀야 합니다.

마찬가지로, 데이터베이스 관리자의 역할이 "가상 데이터베이스 관리자"로 바뀌었습니다. 이들은 웹 기반 콘솔을 통해 리소스를 관리하고, 데이터베이스 하드웨어의 용량이 초과될 경우 스크립트를 실행해 새로운 용량을 프로그래밍 방식으로 추가하고 일일 프로세스를 자동화합니다. 가상 데이터베이스 관리자(DBA)는 이제 새로운 배포 방법(가상 머신 이미지)을 익히고, 새로운 모델(쿼리 병렬화, 지리적 중복성, 비동기 복제[11])를 수용하고, 데이터에 대한 구조적 접근법(샤딩[9], 수평적 분할[13], 페더레이션[14])을 수정하고, 다양한 유형의 데이터 세트별로 클라우드에서 이용 가능한 여러 가지 스토리지 옵션을 활용할 수 있어야 합니다.

기존의 기업에서는 애플리케이션 개발자가 네트워크 관리자와 긴밀하게 협력하지 않으며, 네트워크 관리자는 애플리케이션에 대해 전혀 이해하지 못하는 경우가 있었으며, 그로 인해 네트워크 계층과 애플리케이션 아키텍처 계층에 가능한 최적화 기능이 제대로 이루어지지 못하고 있었습니다. 그러나 클라우드를 사용할 경우 위의 두 역할을 어느 정도 하나로 통합할 수 있습니다. 기업은 미래의 애플리케이션 설계 시, 두 역할 간에 정보가 활발히 공유되도록 하고 두 역할의 통합을 고려해야 합니다.

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

모범적인 클라우드 구현 사례

본 절에서는 클라우드에서 애플리케이션을 구축할 때 도움이 될 수 있는 구현 사례를 소개하겠습니다.

장애 복구형 설계와 장애에 대비하지 않는 설계

경험 법칙: 클라우드에서 아키텍처를 설계할 때는 비관주의자가 되어야 합니다. 즉, 모든 일에 항상 실패가 따를 것을 전제로 해야 합니다. 따라서, 장애를 자동으로 복구할 수 있도록 아키텍처를 설계, 구축, 구현해야 합니다.

특히 언젠가는 하드웨어에 장애 또는 고장이 *발생*할 수 있으며, 중단이 *발생한다고* 가정합니다. 애플리케이션에 재해가 *발생*할 수도 있고, 예상보다 훨씬 더 많은 초당 요청 수로 인해 *피해*를 입을 수도 있음을 전제해야 합니다. 시간이 지남에 따라 애플리케이션 소프트웨어에도 결국 장애가 발생한다고 믿어야 합니다. 비관론자가 되어 설계할 때 장애 복구 전략을 수립하여 전반적으로 더 나은 시스템을 설계할 수 있는 것입니다.

시간이 지날수록 장애가 발생한다는 점을 인식하고, 이러한 사고를 아키텍처에 반영하여, 재해가 발생하기 전에 이러한 장애를 처리할 메커니즘을 구축해 인프라 확장에 대비함으로써, 클라우드에 최적화된 내결함성 (fault-tolerant) 아키텍처를 만들 수 있습니다.

시스템 노드 하나에 장애가 발생하면 어떤 일이 발생할 것이며, 이러한 장애를 감지하는 방법은 무엇인지? 또, 이 노드는 어떻게 교체하며, 이를 위해 어떠한 시나리오를 마련해야 하는지? 장애 발생 시 전체 시스템을 다운시킬 수 있는 지점(single point of failure)은 어디인지? 또, 애플리케이션 서버 열 앞에 로드 밸런서가 있는데, 이 장치가 고장나면 어떻게 해야 하는지?, 아키텍처 내에 마스터와 슬래이브 노드가 있는데, 이 가운데 마스터가 장애를 일으킨다면, 장애 조치가 어떻게 이루어질 수 있는지? 슬래이브 장치를 어떻게 새로 투입하며, 마스터와 어떤 방식으로 동기화를 유지할 것인지? 등등의 질문을 스스로에게 던져 보아야 합니다.

하드웨어 장애에 대비해야 하는 것과 마찬가지로 설계 시 소프트웨어 장애에도 대비해야 합니다. 관련 서비스에서 인터페이스가 변경되면 고객이 보유하고 있는 애플리케이션에는 어떤 일이 발생할 것인지? 다운스트림 서비스가 타임아웃되어 예외값을 리턴할 경우는 어떻게 해야 하는지, 캐시 키가 인스턴스의 메모리 용량을 초과하여 계속 증가하는 경우는 어떻게 해야 하는지? 등등의 질문도 스스로에게 해 보아야 합니다.

이와 같은 장애를 처리할 수 있는 메커니즘을 구축해야 합니다. 예를 들어, 다음과 같은 전략이 도움을 줄 수 있습니다.

1. 데이터에 대해 일관성있는 백업 및 복구 전략을 수립하여 이를 자동화합니다.
2. 재부팅 시 복구할 수 있는 프로세스 스레드를 구축합니다.
3. 대기열에서 메시지를 재로딩하여 시스템 상태를 다시 동기화합니다.
4. 사전 구성 및 최적화를 거친 가상 이미지를 저장하여 시작/부팅 시 (2)와 (3)을 지원하도록 합니다.
5. 메모리 내에 세션 또는 상태 저장(stateful) 사용자 컨텍스트를 유지하지 말고, 이들을 데이터 스토리지로 이동합니다.

우수한 클라우드 아키텍처라면 재부팅 및 재시작 시 손상되지 않아야 합니다. GrepTheWeb (클라우드 아키텍처 관련 자료[6]참조)에서는 Amazon SQS 와 Amazon SimpleDB 를 함께 사용함으로써 전체 컨트롤러 아키텍처가 본 절에 제시된 장애 유형에 대해 뛰어난 회복력을 보입니다. 예를 들어, 컨트롤러 스레드를 실행 중인 인스턴스가 종료되는 경우, 다시 불러와 아무 일도 없었던 것처럼 이전 상태로 다시 시작할 수 있습니다. 이를 위해서는 사전 구성된 Amazon 머신 이미지를 생성해야 합니다. Amazon 머신 이미지가 실행되면 Amazon SQS 대기열에 있는 모든 메시지를 제거합니다. 메시지 상태는 재부팅 시 Amazon SimpleDB 도메인에서 읽을 수 있습니다.

기본 하드웨어에 장애가 발생할 수 있음을 전제로 한 설계방식을 통해 실제로 장애가 발생할 수 있는 미래의 상황에 대비할 수 있습니다.

이러한 설계 원칙은 Hamilton 의 백서[11]에도 중점적으로 설명되어 있는 바와 같이 운용하기 편리한 애플리케이션들을 설계하는 데 도움이 됩니다. 이러한 원칙을 부하를 능동적으로 측정하여 동적으로 균형을 맞추는 데까지 확대하여 적용할 경우, 한 개의 요소에서 여러 요구들을 만족시켜야 하는 클라우드의 특성 (multi-tenant nature) 으로 인해 발생하는 네트워크 및 디스크 성능 변화를 해결할 수 있을 것입니다.

이상에서 제시한 모범적인 방안들을 적용하기 위한 AWS 만의 전략:

1. 엘라스틱 IP 를 사용한 점진적인 장애 조치: 엘라스틱 IP 는 동적으로 재매핑할 수 있는 고정 IP 입니다. 신속하게 재매핑하고 다른 서버 집합으로 장애 조치하여 트래픽을 새 서버로 라우팅할 수 있습니다. 이 방법은 하드웨어를 새 버전으로 업그레이드하거나 하드웨어 장애 발생 시 효과적입니다.
2. 여러 개의 가용 영역(Availability Zones) 활용: 가용 영역의 개념은 논리 데이터 센터와 유사합니다. 고객 아키텍처를 여러 개의가용 영역에 구축하여 가용도를 높일 수 있습니다. Amazon RDS 다중 AZ[21] 배포 기능을 활용하여 데이터베이스 업데이트를 자동으로 여러 가용 구역에 복제합니다.
3. Amazon 머신 이미지를 보유하여 다른 가용 영역에 매우 손쉽게 환경을 저장하거나 복제할 수 있습니다. 즉, 여러 가용 영역에 여러 개의 데이터베이스 슬레이브들을 유지하며, 복제물을 바로 설정할 수 있습니다.
4. Amazon CloudWatch(또는 다양한 실시간 오픈 소스 모니터링 도구)를 활용하여 하드웨어 고장 또는 성능 저하 발생 시 감시 능력을 높여 적절한 조치를 취할 수 있습니다. Auto Scaling 그룹을 설정하여 인스턴스 규모를 일정하게 유지함으로써 이상이 있는 Amazon EC2 인스턴스를 새 것으로 교체합니다.
5. Amazon EBS 를 활용하여 cron 작업을 설정하면 점진적으로 더 많은 스냅샷이 자동으로 Amazon S3 에 업로드되고 데이터는 인스턴스와 독립적으로 유지됩니다.
6. Amazon RDS 를 활용하여 백업 보존 기간을 설정하면 Amazon RDS 가 자동 백업을 수행합니다.

고객 구성 요소들의 해체

클라우드는 *시스템 구성 요소들을 더 느슨하게 결합할 수록, 이들 요소의 확장성이 더 향상된다*는 SOA 설계 원칙을 재확인시켜 줍니다.

이 개념의 핵심은 상호 연관성이 높지 않은 구성 요소들을 구축하여 만약 구성 요소 중 하나가 다운되거나(고장나거나), 수면 상태이거나(응답 없음) 또는 연결 중(응답 느림)일 때, 시스템 내 다른 요소들은 장애가 발생되지 않은 것처럼 동작을 유지할 수 있도록 하는 데 있습니다. 본질적으로 결합이 느슨하면 애플리케이션의 다양한 계층과 구성 요소들을 상호 격리하여 각 구성 요소가 서로 비동기적으로 동작하며, 상대 요소들을 "블랙 박스"로 취급합니다. 예를 들어, 웹 애플리케이션 아키텍처에서는 앱 서버를 웹 서버 및 데이터베이스에서 격리시킬 수 있습니다. 이 앱 서버와 웹 서버가 서로에 대해 인식하지 못하므로, 이들 계층 간 결합이 해체되며, 코드 및 기능면에서 연관성이 사라집니다. 일괄 처리형(batch-processing) 아키텍처에서는 상호 독립적인 *비동기* 구성 요소들을 만들 수 있습니다.

그렇다면, 기존의 단면적인 애플리케이션에서 어떤 비즈니스 요소 또는 기능을 격리하여 단독으로 실행할 수 있는지? 기존 시스템을 파괴하지 않고 이 요소에 인스턴스를 추가하여 더 많은 사용자들을 지원할 수 있는 방법은? 이 요소를 캡슐화하여 다른 요소들과 비동기로 상호 작용할 수 있도록 하는 데는 어떠한 노력이 필요한지?와 같은 질문들을 할 필요가 있습니다.

구성 요소들의 결합을 해체하여 *비동기* 시스템을 구축하고, 수평적으로 확장하는 것은 클라우드의 개념과 관련하여 매우 중요합니다. 이를 통해 동일 요소에 인스턴스들을 추가할 수 있을 뿐 아니라 혁신적인 하이브리드 모델을 설계하여 일부 요소들은 사내에서 실행하고 다른 요소들은 클라우드 규모를 활용하여 컴퓨팅 능력과 대역을 추가로 이용할 수 있게 할 수 있습니다. 이러한 방식으로, 별다른 수고 없이도 스마트한 로드 밸런싱 기술을 구사하여 과도한 트래픽을 클라우드로 전환할 수 있습니다.

*메시징 대기열*을 사용하여 결합이 느슨한 시스템을 구축할 수 있습니다. 두 개의 구성 요소를 연결하는 데 대기열/버퍼를 사용할 경우 동시성, 고가용성 및 부하 스파이크를 지원할 수 있습니다. 그 결과 구성 요소 중 일부를 일시적으로 사용할 수 없게 되더라도 전체 시스템은 계속 작동합니다. 하나의 구성 요소가 다운되거나 일시적으로 사용할 수 없게 되면 시스템은 메시지를 버퍼링한 뒤 구성 요소가 복구되면 처리하도록 합니다.

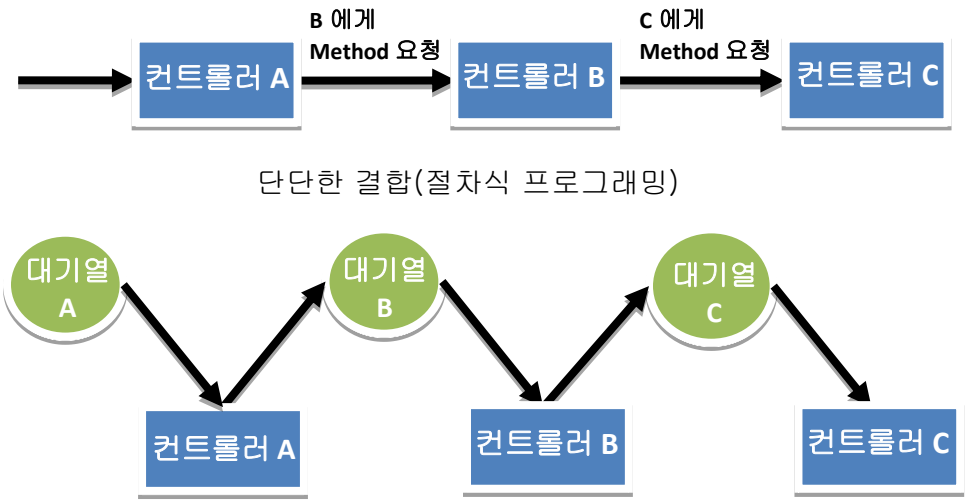

그림 3: 대기열을 사용한 구성 요소 결합 해체

클라우드 아키텍처 백서[6]에는 GrepTheWeb 아키텍처에서 대기열의 활발한 사용 예를 보여주고 있습니다. GrepTheWeb 에서 수많은 요청들이 갑자기 서버에 도달하거나(인터넷으로 인한 과부하 상황) 일상적인 표현들의 처리에 평균(구성 요소의 느린 응답 속도) 이상의 시간이 걸리면 Amazon SQS 대기열이 요청을 지속적으로 버퍼링하여 이러한 지연이 다른 구성 요소에 영향을 주지 않도록 합니다.

이와 같은 모범 사례를 구현하기 위한 AWS 만의 전략은 다음과 같습니다.

1. Amazon SQS 를 사용하여 구성 요소들을 격리합니다.[18]
2. Amazon SQS 를 구성 요소 간 버퍼로 사용합니다.[18]
3. 모든 구성 요소에서 서비스 인터페이스를 공개하고 해당 범위 내에서 자체 확장을 책임지며, 다른 구성 요소와 비동기적으로 상호 작용하도록 설계합니다.
4. 구성 요소의 논리 구조를 Amazon 머신 이미지와 번들로 결합시켜 보다 자주 구현할 수 있도록 합니다.
5. 애플리케이션을 가능한 한 상태 비저장 (stateless)으로 만듭니다. 세션 상태를 구성 요소 외부에 저장합니다 (가능한 경우 Amazon SimpleDB 에 저장).

탄력성 구현

클라우드는 애플리케이션에 탄력성이라는 새로운 개념을 도입했는데, 이러한 탄력성은 다음의 세 가지 방법으로 구현할 수 있습니다.

1. 능동적 주기적 확장: 일정한 간격(매일, 주간, 월간, 분기별)으로 이루어지는 정기적인 확장
2. 이벤트에 따른 능동적 확장: 예정된 비즈니스 이벤트(신상품 출시, 영업 활동 등)로 인한 트래픽 급증 전망에 따른 확장
3. 필요에 따른 자동 확장: 모니터링 시스템을 사용하여 적절한 조치를 취하는 데 필요한 트리거를 전송하여 메트릭(인스턴스의 서버 또는 네트워크 I/O 사용량)에 따라 규모를 늘리거나 줄일 수 있습니다.

"탄력성"을 적용하기 위해서는 우선 배포 과정을 자동화하고 구성 및 구축 과정을 간소화해야 합니다. 그럼으로써, 인력 개입 없이도 시스템에서 자동으로 규모를 늘리거나 줄일 수 있습니다.

이용률이 낮은 유휴 상태의 서버보다는 수요에 맞추어 리소스를 할당함으로써 전반적인 이용률을 높여 즉각적인 비용 효율을 높일 수 있습니다.

인프라 자동화

클라우드 환경을 이용할 때 가장 중요한 장점은 클라우드의 API 를 사용하여 리소스 배포 과정을 자동화할 수 있다는 것입니다. 마이그레이션 과정이 완료될 때까지 기다리기 보다는 초기에 자동 배포 프로세스를 마련하는 것이 바람직합니다. 자동화되고 반복 가능한 배포 과정을 만들어 오류를 줄이고 효율적이고 확장 가능한 업데이트 과정을 촉진할 수 있습니다.

배포 프로세스를 자동화하려면

- 작고 자주 사용하는 설치 및 구성용 스크립트에 대해 "레시피" 라이브러리를 만듭니다.
- AMI 내에 함께 제공되는 에이전트를 사용하여 구성 및 배포 과정을 관리합니다.
- 인스턴스를 부트스트래핑합니다.

인스턴스를 부트스트래핑하는 방법

부팅 시 인스턴스가 이름과 역할에 대해 질문을 하도록 합니다. 모든 인스턴스에 주어진 환경에서 수행해야 할 역할을 배분합니다(웹 애플리케이션의 경우 "DB 서버", "앱 서버", "슬레이브 서버" 등). 이 역할은 인스턴스 시작 시 인수 (argument) 형태로 전달되어, 부팅 후 취해야 하는 단계들을 인스턴스화 (instantiate)해야 할 때를 AMI 에게 알립니다. 부팅 시, 인스턴스는 해당 역할에 따라 필요한 리소스(코드, 스크립트, 구성)를 확보하여, 자신을 클러스터에 "연결"하여 기능을 수행합니다 인스턴스 부트스트래핑의 장점은 다음과 같습니다.

1. 단 몇 번의 클릭과 최소한의 노력만으로 (개발, 준비, 생산) 환경을 다시 만들 수 있습니다.
2. 가상 클라우드 기반 리소스에 대한 통제력을 높여줍니다.
3. 사람이 실수로 인한 배포 또는 배치 상의 실수를 줄일 수 있습니다.
4. 하드웨어 고장 시 회복력이 더 뛰어난 자기 치유 및 자기 검색 가능 환경을 만들 수 있습니다.

고객의 인프라를 자동화하기 위한 AWS 만의 기법

1. Amazon EC2 의 Auto Scaling 기능을 사용하여 서로 다른 클러스터에 대해 Auto Scaling 그룹을 지정합니다.
2. 시스템 메트릭(CPU, 메모리, 디스크 I/O, 네트워크 I/O)을 Amazon Cloudwatch 를 사용하여 모니터링하고, 적절한 조치(Auto Scaling 서비스를 사용하여 새로운 AMI 를 동적으로 시작)를 취하거나 통지합니다.
3. 컴퓨터 구성 정보를 동적으로 저장 및 검색합니다. 즉, Amazon SimplDB 를 사용하여 인스턴스가 부팅되는 동안 구성 데이터를 가져옵니다(예: 데이터베이스 연결 문자열). SimpleDB 는 또한 IP 주소, 컴퓨터 이름 및 역할 등 인스턴스에 대한 정보를 저장하는 데 사용할 수도 있습니다.
4. 구축 프로세스를 설계하여 Amazon S3 의 버킷에 최신 구축물 (builds)을 덤핑하고, 시스템 시작 중에 애플리케이션의 최신 버전을 다운로드합니다.
5. 리소스 관리 도구(자동화 스크립트, 사전 구성된 이미지)를 구축하는 데 투자하거나, Chef[18], Puppet[19], CFEngine[20] 또는 Genome[21]과 같은 오픈 소스 구성 관리 도구를 사용합니다.
6. Just Enough Operating System (JeOS[22])과 고객의 소프트웨어 종속 요소를 Amazon 머신 이미지에 번들시키기 때문에 유지관리가 용이합니다. 시작 시 구성 파일 또는 매개 변수 (parameter)를 전달하고 시작 후에는 사용자 데이터[23]와 인스턴스 메타데이터를 검색합니다.
7. Amazon EBS 볼륨으로 부팅하고[24] 여러 Amazon EBS 볼륨들을 인스턴스 하나에 연결함으로써 번들 처리 및 시작 시간을 줄입니다. 가능한 경우 언제든지 공용 볼륨의 스냅샷을 생성하고 스냅샷[25]을 공유합니다.
8. 애플리케이션 구성 요소는 실행 중인 하드웨어의 상태나 위치에 구애받지 않아야 합니다. 예를 들어, 새 노드의 IP 주소를 클러스터에 동적으로 할당합니다. 장애 시 자동으로 장애 조치하여 새로운 복제 노드를 시작시킵니다.

병렬화에 관하여

클라우드를 이용하면 손쉽게 병렬화를 구현할 수 있습니다. 클라우드에서 데이터를 요청할 때, 데이터를 클라우드에 저장할 때, 클라우드에서 데이터를 처리(또는 작업 실행)할 때 등 클라우드 설계자는 클라우드에 아키텍처를 설계할 때 병렬화의 개념을 이해해야 합니다. 클라우드를 이용하면 반복 가능한 과정을 매번 쉽게 처리할 수 있으므로, 가능한 경우 병렬화를 구현하고 자동화하는 것이 좋습니다.

클라우드는 대량의 데이터 액세스(검색 및 저장) 작업을 병렬로 처리할 수 있도록 설계되었습니다. 최대 성능 및 처리량을 얻으려면 *요청 병렬화 (request parallelization)* 방법을 사용해야 합니다. 다수의 동시 스레드를 사용하여 요청을 다중 스레딩하면 데이터를 순차적으로 요청할 때 보다 더 빠르게 데이터를 저장하거나 가져올 수 있습니다. 따라서 가능한 경우 클라우드 애플리케이션의 프로세스에서 무공유 원칙과 다중 스레드를 통해 스레드를 안전하게 관리할 수 있어야 합니다.

---

[18] Chef 에 대한 자세한 사항은 웹 사이트(http://wiki.opscode.com/display/chef/Home)를 참조하십시오.
[19] Puppet 에 대한 자세한 사항은 웹 사이트(http://reductivelabs.com/trac/puppet/)를 참조하십시오.
[20] CFEngine 에 대한 자세한 사항은 웹 사이트(http://www.cfengine.org/)를 참조하십시오.
[21] Genome 에 대한 자세한 사항은 웹 사이트(http://genome.et.redhat.com/)를 참조하십시오.
[22] http://en.wikipedia.org/wiki/Just_enough_operating_system
[23] 인스턴스 메타데이터와 사용자 데이터 사항은 웹 사이트(http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/index.html?AESDG-chapter-instancedata.html)를 참조하십시오.
[24] Amazon EBS 로 부팅 기능에 대한 자세한 사항은 웹 사이트(http://developer.amazonwebservices.com/connect/entry.jspa?externalID=3121)를 참조하십시오.
[25] 스냅샷 공유 방법은 웹 사이트(http://aws.amazon.com/ebs/)를 참조하십시오.

병렬화는 클라우드에서 요청을 처리하거나 실행할 때 훨씬 더 중요합니다. 웹 애플리케이션에서 가장 좋은 방법은 로드 밸런서를 사용하여 여러 개의 비동기 웹 서버로 진입하는 요청들을 분산하는 것입니다. 일괄 처리 애플리케이션에서는 마스터 노드를 사용하여 병렬로 작업을 처리하도록 여러 개의 슬레이브 작업 노드를 생성할 수도 있습니다(Hadoop 과 같은 분산 처리 프레임워크와 유사함[26]).

탄력성과 병렬화를 조합하면 클라우드의 우수성이 빛을 발하게 됩니다. 고객의 클라우드 애플리케이션은 수 분 이내에 단지 몇 개의 API 호출만으로 설정된 컴퓨팅 인스턴스로 클러스터를 만들어 병렬로 작업을 수행하고, 그 결과를 저장한 다음 모든 인스턴스를 종료합니다. [6]에서 설명한 GrepTheWeb 애플리케이션이 이러한 애플리케이션의 한 예입니다.

AWS 만의 병렬화 전략:

1. 모범 사례 백서[2]에 자세하게 설명한 바와 같이 Amazon S3 요청들을 멀티스레딩합니다.
2. Amazon SimpleDB GET 및 BATCHPUT 요청[3][4] [5]을 멀티스레딩 합니다.
3. Amazon Elastic MapReduce Service 를 사용하여 각 일일 일괄 처리 프로세스(인덱싱, 로그 분석 등)에 대한 JobFlow 를 생성하면 작업을 병렬로 처리하여 시간이 절약됩니다.
4. Elastic Load Balancing 서비스를 사용하여 부하를 *동적으로* 여러 웹 앱 서버에 분산합니다.

동적 데이터는 컴퓨팅에 더 가깝게, 정적 데이터는 최종 사용자에 더 가깝게

일반적으로 데이터를 가능한 한 컴퓨팅 또는 처리 요소에 가깝게 유지하여 지연 시간을 줄이는 것이 바람직합니다. 클라우드에서는 인터넷의 지연 처리 문제를 해결해야 하기 때문에 이러한 원칙이 더욱 의미가 있고 중요합니다. 더욱이 클라우드에서는 데이터 전송량(GB)에 따라 클라우드 안팎의 대역폭 사용료를 지불해야 하는데 요금이 매우 빨리 증가할 수 있습니다.

처리해야 할 대량의 데이터가 클라우드 외부에 존재할 경우, 데이터를 우선 클라우드로 "전송"한 다음 컴퓨팅을 수행하는 것이 비용도 저렴하고 속도도 빠릅니다. 예를 들어, 데이터 보관 애플리케이션의 경우 데이터 세트를 클라우드로 이동한 다음 데이터 세트에 대한 병렬 조회를 수행하는 것이 좋습니다. 관계형 데이터베이스로부터 데이터를 저장하고 검색하는 웹 애플리케이션의 경우, 앱 서버와 데이터베이스를 한 번에 클라우드로 이동하는 것이 좋습니다.

데이터가 클라우드에서 생성되는 경우, 이 데이터를 소비하는 애플리케이션 역시 클라우드 내에 구현되어 클라우드 내 무료 데이터 전송 및 지연 단축의 장점을 활용할 수 있습니다. 예를 들어, 로그와 클릭스트림 데이터를 생성하는 전자 상거래 웹 애플리케이션의 경우, 클라우드에서 로그 분석기와 보고 엔진을 실행하는 것이 좋습니다.

반대로, 데이터가 정적이며 자주 변경 하지 않을 경우(예를 들어, 이미지, 비디오, 오디오, PDF, JS, CSS 파일), 최종 사용자(요청자)에 더 가까운 에지에서 정적 데이터를 캐싱하여 액세스 지연 시간을 줄일 수 있도록 콘텐츠 전송 서비스를 이용하는 것이 좋습니다. 콘텐츠 전송 서비스를 이용하면 캐싱 덕분에 자주 요청되는 객체에 더 빨리 액세스할 수 있습니다.

[26] http://hadoop.apache.org/

이 모범 사례를 구현하기 위한 AWS 만의 전략:

1. 데이터 드라이브를 Import/Export 서비스[27]를 사용하여 Amazon 에 장착합니다. 인터넷을 통해 업로드하는 것보다 sneakernet[28]을 통해 대량의 데이터를 이동하는 것이 비용도 저렴하고 속도도 빠릅니다.
2. 동일한 가용 영역에서 여러 장치들을 시작합니다.
3. Amazon S3 버킷의 배분을 시작하여 Amazon CloudFront 이 전 세계 14 개 에지에서 해당 버킷의 콘텐츠를 캐싱할 수 있도록 합니다.

보안 관련 모범 사례

멀티 테넌트 환경인 경우 클라우드 설계자가 보안에 대해 염려하는 경우가 있습니다. *보안은 클라우드 애플리케이션 아키텍처의 모든 계층에 구현되어야 합니다.* 물리적 보안은 일반적으로 서비스 공급자가 제공하게 되는데(보안 백서[7]), 이러한 특성이 클라우드 사용과 관련하여 부수적으로 얻을 수 있는 장점입니다. 네트워크 및 애플리케이션 수준의 보안은 고객의 책임사항이며, 고객의 비즈니스에 부합하는 모범 사례로 구현해야 합니다. 본 절에서는 AWS 환경에서 클라우드 애플리케이션을 안전하게 보호하는 데 필요한 몇 가지 구체적인 도구, 기능 및 지침들에 대해 알아봅니다. 여기서 언급하는 이러한 도구와 기능을 사용하여 기본적인 보안을 구현하고 난 뒤 적절하거나 적합하다고 생각되는 표준 방법을 사용하여 추가적인 보안 기법을 구현하는 것이 좋습니다.

전송 중 데이터 보호

브라우저와 웹 서버 간에 민감한 기밀인 정보를 교환해야 할 때, 고객 서버 인스턴스에 SSL 을 구성합니다. VeriSign[29] 또는 Entrust[30]와 같은 외부 인증 기관이 발행하는 인증서도 필요합니다. 인증서에 포함된 공개 키가 브라우저에 대해 서버를 인증하고 양방향에서 데이터를 암호화하는 데 사용되는 공유 세션 키를 만드는 기반 역할을 합니다.

몇 가지 명령문 호출(Amazon VPC 사용)을 통해 Virtual Private Cloud 를 하나 생성할 수 있습니다. 이렇게 하면 AWS 클라우드 내에 논리적으로 격리된 리소스를 사용할 수 있으며, 이러한 리소스를 업계 표준인 암호화된 IPSec VPN 연결을 통해 데이터 센터에 직접 연결할 수 있습니다.

또한 Amazon EC2 인스턴스에 [15] OpenVPN 서버를 설정하고 모든 사용자 PC 에 OpenVPN 클라이언트를 설치할 수도 있습니다.

보관 중인 데이터 보호

민감한 기밀 데이터를 클라우드에 저장할 때 보안이 염려된다면 데이터(개별 파일)를 암호화한 후에 클라우드에 업로드하면 됩니다. 예를 들어, Amazon S3 객체로 저장하기 전에 오픈 소스 또는 [31]상용[32] PGP 기반 도구를 사용하여 데이터를 암호화하고, 다운로드한 후에 복호화합니다. 이것은 PHI (Protected Health Information)를 저장해야 하는 HIPPA 호환 애플리케이션[8]을 구축할 때 좋은 방법입니다.

---

[27] Amazon Import Export Services 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/importexport)를 참조하십시오.
[28] http://en.wikipedia.org/wiki/Sneakernet
[29] http://www.verisign.com/ssl/
[30] http://www.entrust.net/ssl-products.htm
[31] http://www.gnupg.org
[32] http://www.pgp.com/

Amazon EC2 에서는 파일 암호화가 운영 체제에 따라 달라집니다. Windows 를 구동하는 Amazon EC2 인스턴스는 내장형 EFS (Encrypting File System) 기능을 사용할 수 있습니다[16]. 이 기능은 파일과 폴더의 암호화 및 복호화를 자동으로 처리하며, 사용자가 처리 과정을 투명하게 볼 수 있게 합니다[19]. 그러나 그 이름과 달리 EFS 는 전체 파일 시스템을 암호화하지 않고, 대신 개별 파일을 암호화합니다. 전체 암호화된 볼륨이 필요하면 오픈 소스 TrueCrypt[33] 제품을 사용하는 것이 좋습니다. 이 제품은 NTFS 형식의 EBS 볼륨과 잘 호환됩니다. Linux 구동 Amazon EC2 인스턴스는 다양한 방법(EncFS[34], Loop-AES[35], dm-crypt[36], TrueCrypt[37])으로 암호화된 파일 시스템을 사용하여 EBS 볼륨을 마운트할 수 있습니다. 마찬가지로 OpenSolaris 를 구동하는 Amazon EC2 인스턴스는 ZFS[38] Encryption Support[20]를 사용합니다. 어떤 방법을 사용하든 Amazon EC2 에서 파일 및 볼륨을 암호화하면 파일과 로그 데이터가 보호되기 때문에 서버 내에 있는 사용자와 프로세스는 데이터를 분명한 텍스트 형태로 볼 수 있지만, 서버 밖에서는 데이터가 암호화된 형태로 보입니다.

어떤 운영 체제 또는 기술을 선택하든지 보관 중인 데이터를 암호화하는 경우에는 데이터 암호화에 사용되는 키를 관리해야 하는 문제에 직면하게 됩니다. 키를 잃어버리면 데이터를 영원히 잃게 되고, 키가 손상되면 데이터가 위험에 노출될 수 있습니다. 따라서 사용하는 제품의 키 관리 기능을 숙지하여 키를 잃어버릴 위험성을 최소화하도록 필요한 절차를 마련해야 합니다.

데이터를 도청으로부터 보호하는 것뿐만 아니라 재해에서 보호하는 방법도 고려해야 합니다. 정기적으로 Amazon EBS 볼륨의 스냅샷을 만들어 높은 지속성과 가용성을 유지해야 합니다. 스냅샷은 특성상 숫자가 증가되어 Amazon S3(별도의 지리적 위치)에 보관되며, 단 몇 번의 클릭이나 명령만으로 다시 복구 가능합니다.

고객의 AWS 자격 증명 보호

AWS 는 AWS 액세스 키와 X.509 인증서 등 2 가지 종류의 자격 증명을 제공합니다. AWS 액세스 키는 *액세스 키 ID* 와 *보안 액세스 키*의 두 부분으로 구성됩니다. REST 또는 Query API 를 사용할 때 고객의 비밀 액세스 키를 사용하여 인증 요청에 포함시킬 서명 값을 계산해야 합니다. '전송 중 도청'을 방지하려면 모든 요청을 HTTPS 를 통해 전송해야 합니다.

Amazon 머신 이미지(AMI)가 다른 AWS 웹 서비스와 통신하기 위해 (Amazon SQS 대기열 폴링 또는 Amazon S3 에서 객체 읽기 등) 프로세스를 실행할 경우, 범하기 쉬운 한 가지 설계상 실수는 AWS 자격 증명을 AMI 에 포함시키는 것입니다. 자격 증명을 AMI 에 포함시키는 대신 시작 시 인수 형태(arguments)로 전달하고 유선으로 [17]전송하기 전에 먼저 암호화해야 합니다.

보안 액세스 키가 손상되면 신규 액세스 키 ID 로 교체[39]하여 새 키를 받아야 합니다. 가장 권하고 싶은 방법은 고객의 애플리케이션 아키텍처에 키 교체 방식을 통합하여 정기적으로 또는 필요에 따라(불만을 품은 직원이 퇴사하는 경우) 이를 사용함으로써 손상된 키를 계속 사용하지 않도록 하는 것입니다.

또는 특정 AWS 서비스에 대한 인증을 위해 X.509 인증서를 사용할 수도 있습니다. 이 인증서 파일에는 base64 로 암호화된 DER 인증서 본문에 공개 키가 포함되어 있습니다. 별도의 파일에는 상응하는 base64 로 암호화된 PKCS#8 개인 키가 포함되어 있습니다.

---

[33] http://www.truecrypt.org/
[34] http://www.arg0.net/encfs
[35] http://loop-aes.sourceforge.net/loop-AES.README
[36] http://www.saout.de/misc/dm-crypt/
[37] http://www.truecrypt.org/
[38] http://www.opensolaris.org/os/community/zfs/
[39] http://aws.amazon.com/about-aws/whats-new/2009/08/31/seamlessly-rotate-your-access-credentials/

AWS 는 다중 요소 인증을 지원하는데, [40]이는 aws.amazon.com 및 AWS Management Console 에 있는 계정 정보를 이용하여 작업할 때 추가적인 보호 기능 역할을 합니다[41].

IAM 을 이용한 다수 사용자 및 사용자 권한 관리

AWS Identity and Access Management (IAM)[42]를 이용하면 본인의 AWS 계정을 이용하여 여러 사용자를 생성하고, 각 사용자의 권한을 관리할 수 있습니다. 사용자란 AWS 서비스 액세스 시 사용되는 고유한 보안 자격 증명을 포함하는일종의 아이덴티티(고객 AWS 계정에 포함)를 의미합니다. IAM 을 이용하면 암호나 액세스 키를 공유할 필요가 없어져 적절한 방법으로 각 사용자의 액세스를 손쉽게 허용하거나 거부할 수 있습니다.

IAM 을 사용하면 모범적인 보안 사례를 구현할 수 있습니다. 예를 들어, 고객의 AWS 계정 내에 있는 모든 사용자에게 고유의 자격 증명을 부여하여 사용자에게 작업을 수행하는 데 필요한 AWS 서비스와 리소스 액세스 권한만 주는 것입니다. IAM 은 기본값(default)으로 보안이 설정되어 있습니다. 신규 사용자에게 구체적인 권한이 부여되기 전까지는 AWS 에 액세스할 수 없습니다.

IAM 은 기본적으로 대부분의 AWS 서비스 내에 포함되어 있습니다. 어떠한 서비스 API 도 IAM 을 지원하기 위해 변경할 필요가 없으며, AWS 서비스 API 를 이용해 구축된 애플리케이션과 도구들은 IAM 을 사용하는 경우에도 계속 동작을 유지합니다. 따라서 신규 사용자를 위해 생성된 액세스 키를 사용하여 애플리케이션을 시작하기만 하면 됩니다.

AWS 서비스를 사용할 때는 되도록 고객 AWS 계정의 자격 증명 사용을 최소화하고, AWS 서비스와 리소스에 액세스할 때 IAM 사용자 자격 증명을 사용해야 합니다.

고객 애플리케이션 보호

모든 Amazon EC2 인스턴스는 하나 이상의 *보안 그룹*[43]으로 보호됩니다. 보안 그룹이란 고객 인스턴스에 전달할 인그레스 (인입) 네트워크 트래픽을 지정하는 규칙 집합을 의미합니다. TCP/UDP 포트, ICMP 유형과 코드 및 소스 주소를 지정할 수 있습니다. 보안 그룹은 인스턴스를 실행하기 위해 방화벽과 같은 기본적인 보호 방법을 제공합니다. 예를 들어, 웹 애플리케이션에 속한 인스턴스는 다음과 같은 보안 그룹 설정이 가능합니다.

---

[40] Multi-factor Authentication 에 대한 자세한 사항은 웹 사이트(http://aws.amazon.com/ko/mfa/)를 참조하십시오.
[41] AWS Management Console(http://aws.amazon.com/console/)
[42] 자세한 사항은 웹 사이트(http://aws.amazon.com.com/iam)를 참조하십시오.
[43] 보안 그룹에 대한 자세한 사항은 웹 사이트(http://docs.amazonwebservices.com/AWSEC2/2009-07-15/UserGuide/index.html?using-network-security.html)를 참조하십시오.

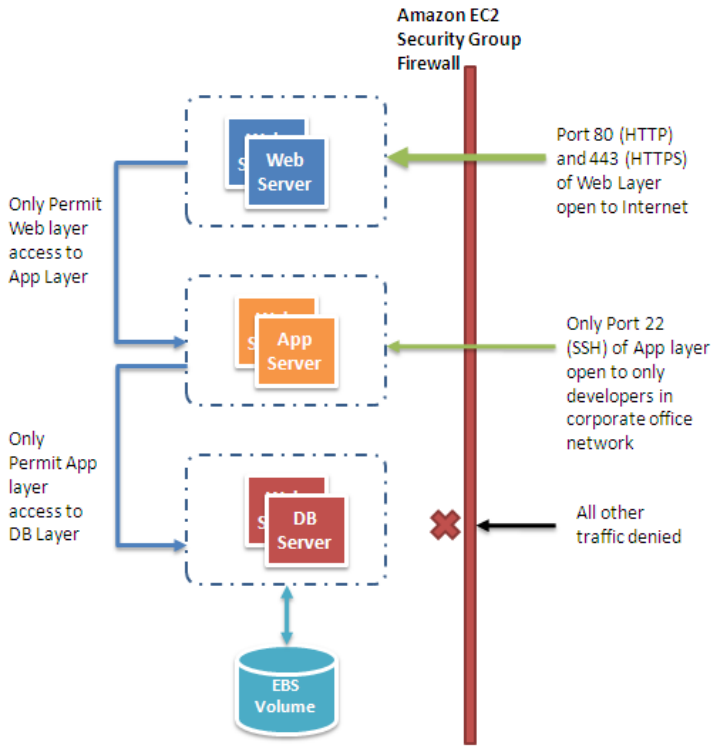

그림 4: Amazon EC2 보안 그룹을 사용한 웹 애플리케이션 보호

인입 트래픽을 제한하는 또 다른 방법은 고객 인스턴스에 방화벽 소프트웨어를 구성하는 것입니다. Windows 인스턴스는 내장된 방화벽[44]을 사용할 수 있으며, Linux 인스턴스는 *netfilter*[45] 및 *iptables* 를 사용할 수 있습니다.

시간이 지남에 따라 소프트웨어의 오류를 발견하게 되고, 수정을 위해 패치 작업을 해야 합니다. 애플리케이션의 보안을 극대화하기 위해서는 다음의 기본 지침에 따라야 합니다.

- 공급업체의 웹 사이트에서 정기적으로 패치를 다운로드하고 AMI 를 업데이트합니다.
- 새 AMI 에서 인스턴스를 다시 배포하고 패치로 인해 손상된 것이 없는지 확인하기 위해 고객의 애플리케이션을 시험합니다. 최신 AMI 가 *모든* 인스턴스에 배포되었는지 확인합니다.
- 테스트 스크립트에 투자하여 정기적인 보안 점검을 실행하고 프로세스를 자동화합니다.
- 타사 소프트웨어를 가장 안전한 설정으로 구성되도록 합니다.
- 반드시 필요한 경우가 아니라면 프로세스를 *루트* 또는 *관리자* 로그인으로 실행하지 않습니다.

좋은 코딩 방법을 사용하거나, 중요한 데이터를 격리하는 것과 같은 클라우드 이전에 적용되었던 모든 표준 보안 관행은 여전히 적용 가능하며 구현되어야 합니다.

요컨대, 고객이 필요로 하는 복잡한 물리적 보안 기능을 요약 정리하여, 고객의 애플리케이션을 안전하게 유지할 수 있도록 도구 및 기능들을 이용한 컨트롤 기능을 제공합니다.

---

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

미래 연구 방향

애플리케이션이 물리적 하드웨어 형태에 구애받지 않게 될 날이 머지 않았습니다. 전자레인지에 전원을 연결할 때 전기에 대한 지식까지 필요하지 않은 것처럼 애플리케이션을 클라우드에 *플러그인*하면 전기를 사용하듯이 애플리케이션을 실행하는 데 필요한 동력을 얻을 수 있게 될 것입니다. 설계자는 물리적 서버 대신 가상 컴퓨팅, 스토리지 및 네트워크 리소스를 관리하게 될 것입니다. 기본 물리적 하드웨어에 장애가 발생하거나 하드웨어를 제거 또는 교체하더라도 애플리케이션은 동작 상태를 계속 유지할 것입니다. 변화하는 수요 패턴에 맞춰 애플리케이션이 리소스를 *즉시* 자동으로 배분하여 사용률 수준을 언제나 최상으로 유지할 수 있게 될 것입니다. 애플리케이션 아키텍처에서 확장성, 보안, 고가용성, 내결함성, 테스트 용이성, 탄력성 등의 속성을 설정할 수 있게 될 것입니다. 또한, 플랫폼에서 이러한 속성을 자동으로 구성할 수 있게 될 것이며, 이러한 속성이 플랫폼의 본질적인 부분이 될 것입니다.

하지만 아직은 이러한 단계까지 이르지는 못했습니다. 현재는 본 백서에서 소개한 모범 사례를 구현하는 방법으로 이러한 특징 가운데 일부를 갖춘 클라우드 내에 애플리케이션을 구축할 수 있습니다. 클라우드 컴퓨팅 아키텍처의 모범 사례는 계속 발전할 것입니다. 연구원으로서 우리는 클라우드를 개선할 뿐 아니라 도구, 기술 및 프로세스를 구축하는 데 주력하여 개발자와 설계자가 클라우드에서 애플리케이션을 쉽게 사용할 수 있도록 할 것입니다.

결론

본 백서는 효율적인 클라우드 애플리케이션 설계를 위해 필요한 규범적 지침을 클라우드 설계자에게 제공할 목적으로 작성했습니다.

기본 개념과 모범 사례(장애에 대비하기 위한 설계, 애플리케이션 구성 요소들의 해체, 탄력성의 이해와 구현, 병렬화, 애플리케이션 아키텍처의 모든 측면에 보안 구현 방법)을 중점적으로 다루고 있어 클라우드 설계자들이 확장성 뛰어난 클라우드 애플리케이션을 구축하는 데 필요한 사항을 설계 시 고려할 수 있도록 했습니다.

AWS 클라우드는 사용량만큼 요금을 부과하는 매우 안정적인 인프라 서비스를 제공합니다. 본 백서에서 집중적으로설명한 AWS 중심 전략들은 고객 여러분이 이 서비스를 사용하여 클라우드 애플리케이션을 설계하는 데 도움을 드릴 수 있을 것입니다. 본 자료를 작성한 연구자로서 본인은 고객 여러분들이 AWS 와 같은 상용 서비스뿐 아니라 다른 제품들도 익히는 가운데, 이를 바탕으로 더 나은 클라우드 컴퓨팅 기술을 보완, 발전시켜 나갈 것을 권해 드립니다.

감사의 말

본 백서의 초안 작성 시 의견을 제시해 준 Jeff Barr, Steve Riley, Paul Horvath, Prashant Sridharan 및 Scot Marvin 에게 진심으로 감사를 드립니다. 귀중한 의견을 제공해 준 Matt Tavis 에게 특별히 감사를 전합니다. 그분의 도움이 없었다면 본 백서 작성은 불가능했을 것입니다.

본 백서 내용의 일부는 위 저자가 저술한 'Cloud Computing: Paradigms and Patterns'에서 인용한 것입니다.
Copyright © 2010 John Wiley & Sons, Inc.

참고 문헌

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

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

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