01

GS Neotek

ARCHITECTING FOR THE CLOUD

Best Practices

www.wisen.co.kr

WiseN

Wisely Combine the Network platforms

ARCHITECTING FOR THE CLOUD

Best Practices

Contents

소개

지난 몇 년 동안,
소프트웨어 설계자들은 확장성이 뛰어난 어플리케이션 구축하기 위해 여러 개념과 모범 사례를 발견하고 구현해 왔습니다. “테라의 시대(era of tera)”인 오늘 날, 끊임 없이 늘어나는 데이터와 예측할 수 없는 트래픽 패턴, 그리고 더욱 빠른 응답 시간에 대한 요구로 인해 확장성에 대한 이러한 개념들이 더욱 잘 적용될 수 있습니다. 이 문서는 기존의 개념들을 강화하여 클라우드 컴퓨팅으로 발전시킬 수 있는 방법을 설명합니다. 또한 클라우드의 특성으로 알려진 탄력성이라는 개념에 대해서도 설명합니다.

이 문서는
엔터프라이즈 어플리케이션을 고정된 물리적 환경에서 가상화된 클라우드 환경으로 이동하고자 하는 클라우드 아키텍트를 대상으로 삼고 있습니다. 또한 기존의 어플리케이션을 클라우드로 마이그레이션 하거나 새로운 클라우드 어플리케이션을 생성하기 위한 개념, 원리 및 모범 사례에 초점을 맞추고 있습니다.

배경

클라우드 아키텍트로서
클라우드 컴퓨팅의 장점을 이해하는 것은 중요합니다.
이 섹션에서는, 클라우드 컴퓨팅의 사업적, 기술적 장점과
다양한 AWS 서비스에 대해 배우게 될 것입니다.

클라우드 컴퓨팅의 사업적 장점

클라우드에서 어플리케이션을 구축하는 데에는 다음과 같은 명확한 사업적 장점이 있습니다.

거의 0에 가까운 초기 투자 비용

만약 대규모 시스템을 구축해야 하는 경우에는 운영 요원 및 장비 관리 (전원 관리, 냉각), 장비 (랙, 서버, 라우터, 백업 전원 공급장치), 물리적 보안, 부동산에 큰 비용을 투자 해야 할 수도 있습니다. 높은 초기 비용 때문에, 수 차례 경영진으로부터 허락를 받아야 할 것입니다. 하지만, 클라우드 컴퓨팅을 활요하면 고정 비용이나 초기 비용이 발생하지 않습니다.

Just-In-Time 인프라스트럭쳐

과거에는, 여러분의 어플리케이션이 인기를 끌었지만 인프라가 충분하지 않아 어플리케이션의 성공이 실패를 만드는 경우도 있고 반면에 충분한 인프라를 준비했지만 어플리케이션이 인기를 끌지 못해 실패를 만드는 경우도 있습니다. 하지만, 필요한 때에 자동 확장이 가능한 클라우드에 어플리케이션을 배포함으로써, 대규모 시스템을 위한 사전 예측을 하지 않아도 됩니다. 이러한 특징은 서비스 대응에 대한 민첩성은 증가시키는 반면, 사용량만큼 비용을 지불하기 때문에 리스크와 비용은 감소됩니다.

보다 효율적인 자원 활용

시스템 관리자들은 용량을 초과했을 때 장비를 구하는 것과 유휴 용량으로 인한 활용도에 대해 고민을 합니다. 클라우드 환경에서는 필요할 때 자원을 요청하고 반환하기 때문에 자원을 보다 효율적으로 관리할 수 있습니다.

사용량 기반 비용

클라우드 가격 정책에 따라 사용한 만큼의 비용이 청구됩니다. 자원을 할당하더라도 사용하지 않은 자원에 대해서는 비용을 지불하지 않습니다. 패치를 통해 어플리케이션을 업데이트 할때, 바로 다음 달 청구서를 통해 즉시 비용 절감을 확인할 수 있습니다. 예를 들어, 캐싱을 통해 데이터 요청을 70% 감소시킬 경우, 비용 절감이 즉시 발생되고 다음 달 청구서에서 확인할 수 있습니다. 클라우드 상에 플랫폼을 구축한다면, 고객에게 동일한 유연성과 다양한 사용량 기반의 비용 방법을 제공할 수 있습니다.

시장 출시 시간의 단축

병렬화는 프로세싱을 가속화하는데 좋은 방법 중 하나입니다. 만일 하나의 연산 집약적 작업이나 데이터 집약적 작업이 병렬적으로 하나의 시스템에서 프로세스 하는데에 500 시간 걸릴수 있다면, 클라우드 설계로는, 500 인스턴스를 실행 및 스폰하고 같은 작업을 1시간 안에 프로세스 하는 것이 가능할 수 있습니다.
탄력성 (Elasticity) 기반의 유효성을 가지고 있는 것은 시장 출시 기간을 줄이면서 비용 효과적인 방식으로 병렬 처리를 이용하는 기능을 가진 어플리케이션을 제공합니다.

클라우드 컴퓨팅의 기술적인 장점

클라우드 컴퓨팅이 기술적 장점 중 일부는 자동화.
“스크립트 방식의 인프라” 프로그램화를 할 수 있는 (API 기반) 인프라를 활용하여 반복적인 빌드 및 배포 시스템을 만들 수 있습니다.

자동 크기 조절 (Auto-Scaling)

관리자의 개입 없이 예상치 못한 수요에 맞게 어플리케이션을 확장하거나 축소할 수 있습니다. Auto-Scaling은 자동화를 통해 더 많은 효율성을 제공합니다.

사전 스케일링

트래픽 패턴 분석을 통해 수요를 예상하여 어플리케이션을 확장 또는 축소할 수 있습니다.

더 효율적인 개발 라이프 사이클

개발 시스템을 쉽게 개발 및 테스트 환경으로 사용하기 위해 복제 할 수 있습니다. 스테이징 환경은 개발 과정을 더욱 쉽게 할 수 있습니다.

개선된 시험가능성

테스트를 위해 하드웨어가 부족하지 않게 됩니다. 개발 과정의 모든 단계에서 테스트를 자동화 시켜줍니다. 테스트 단계 동안만 사전 구성된 환경과 함께 “인스턴트 테스트 랩”을 복제 구성할 수 있습니다.

재해 복구 및 비즈니스 연속성

클라우드는 DR 서버와 스토리지 시스템을 유지하기 위해 낮은 비용의 옵션을 제공합니다. 클라우드를 사용하면 지리적 분포와 다른 위치에 복제를 빠른 시간 내에 가능하게 하는 장점을 가질수 있습니다.

초과된 트래픽을 클라우드로 소화

초과하는 트래픽을 클라우드로 라우팅하여 완전한 오버 플로우 방지 어플리케이션을 만들 수 있습니다.

아마존 웹 서비스 클라우드의 이해

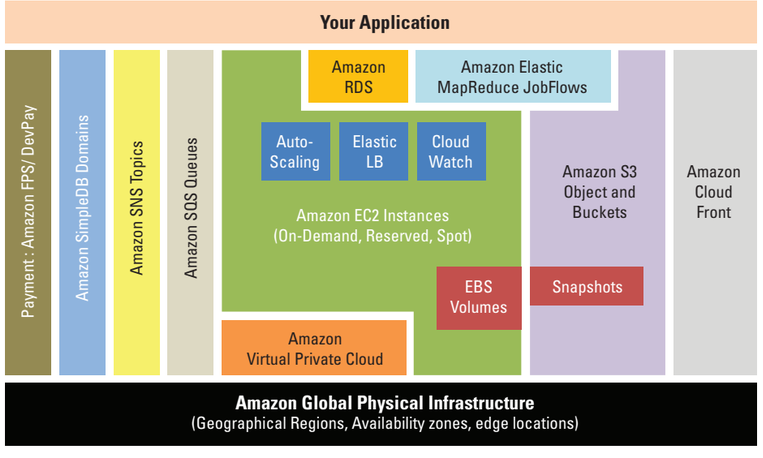
아마존 웹 서비스 (AWS) 클라우드는 사내 또는 데이터센터 시설 내에서 기존 구축된 인프라보다 최소한의 지원 및 관리 비용, 더 많은 유연성을 가지고 웹 스케일 솔루션을 배포하는 데 매우 안정적이고 확장 가능한 인프라를 제공합니다.

AWS는 다양한 형태의 인프라 서비스를 제공합니다. 아래의 그림에서는 AWS 용어를 소개하고, 어떻게 여러분의 어플리케이션이 각각의 아마존 웹 서비스와 작용하고, 다른 서비스들이 서로 상호 작용하는지, 그리고 각자 어떻게 상호 작용하는지를 이해하는데 도움이 될 것입니다.

아마존 엘라스틱 컴퓨트 클라우드 (아마존 EC2)는 클라우드 컴퓨팅 용량 크기 조정이 가능한 웹 서비스입니다. 아마존 머신 이미지 (AMI)로 운영 체제, 응용 소프트웨어 및 관련 구성 설정을 묶을 수 있습니다. 용량 변경에 따라, AMI를 사용하여 용량을 확장하거나 해제하여 축소할 수 있습니다.

EC2 서비스에는 시간당으로 지불하는 온 디맨드 인스턴스를 구입하거나, 초기에 일회 지불 방법으로 낮은 비용을 유지할 수 있는 예약 인스턴스를 구매할 수 있고, 이 예약 인스턴스는 온 디맨드 인스턴스나 사용하지 않는 자원을 입찰을 통해 구입할 수 있는 스팟 인스턴스보다 상대적으로 낮은 사용량을 통해 비용을 절감할 수 있습니다.

인스턴스는 하나 이상의 지역(Region)에서 실행 할 수 있습니다. 각 지역은 여러 가용 영역(AZ)을 가지고 있고, 가용 영역은 지리적으로 분리되어 있으며, 다른 가용 영역의 장애에 영향을 받지 않고 저렴하고 낮은 지연시간의 네트워크를 제공합니다.



클라우드의 개념

클라우드는 확장성이 뛰어난 인터넷 아키텍처를 구축하려는 기존의 개념을 강화하고 어플리케이션을 구축 및 배포하는 방식을 전체적으로 바꿀 수 있는 새로운 개념을 소개합니다. 개념에서 실제 구현으로 넘어갈 때, “모든 것이 바뀌었지만 달라진 것은 없군.”이라고 느낄 수 있습니다. 클라우드는 여러 프로세스, 패턴, 관행, 철학을 바꾸고, 그 동안 배웠던 몇몇 기존의 서비스 지향 아키텍처 원칙에 대해서는 예전보다 더욱 중요해진 만큼 더욱 강화시켜주게 됩니다. 이 섹션에서는 이러한 새로운 클라우드의 개념을 볼 것이고, SOA의 개념을 확인할 것입니다.

기존의 응용 프로그램들이 개발하는 경우에는, 개발되는 그 당시의 경제적인 또는 아키텍쳐적인 측면을 사전에 예상하여 만들었습니다. 하지만, 클라우드는 새로운 철학을 가져왔고, 아래 사항을 통해서 이해할 필요가 있습니다.

확장 가능한 아키텍처 구축

확장성이라는 장점을 활용하기 위해 확장성이 있는 아키텍처를 구축하는 것이 중요합니다.

클라우드는 개념적으로 무한한 확장성을 제공하도록 설계되었습니다. 하지만 아키텍처 자체가 확장성이 없는 경우에는, 클라우드에서 제공하는 다양한 확장성의 개념을 활용할 수 없습니다. 인프라와 아키텍쳐 둘 다 함께여야만 합니다. 기존 아키텍쳐의 획일적 구성 요소와 병목 현상을 식별하고, 아키텍처 온 디맨드 프로비저닝 기능을 활용할 수 없는 영역을 찾아내고, 확장 가능한 인프라를 활용 및 클라우드를 활용하기 위해 응용 프로그램을 리팩토링해야 합니다.

확장 가능한 어플리케이션의의 특징 :

- 리소스의 증가에 비례하여 성능을 증가시킬 수 있다.
- 확장 가능 서비스는 시스템간의 이질성을 처리 할 수 있다
- 확장 가능 서비스는 운영적인 측면에서 효율적이다.
- 확장 가능 서비스는 탄력적이다
- 확장 가능한 서비스는 인프라가 증가할 때 더욱 비용 효율적이 되어야 한다. (유닛의 수가 증가하면 단위 당 비용은 감소한다.)

위 사항들이 어플리케이션의 고유한 부분이 되어야 하고, 이러한 특성들을 고려하여 구조를 설계하면, 그 동안 고민하였던 아키텍처와 인프라의 확정성의 문제를 해결할 수 있을 것입니다.

탄력성(Elasticity)의 이해

아래의 그래프는 클라우드 아키텍트가 어플리케이션을 확장할 때 접근 가능한 방식들을 보여줍니다.

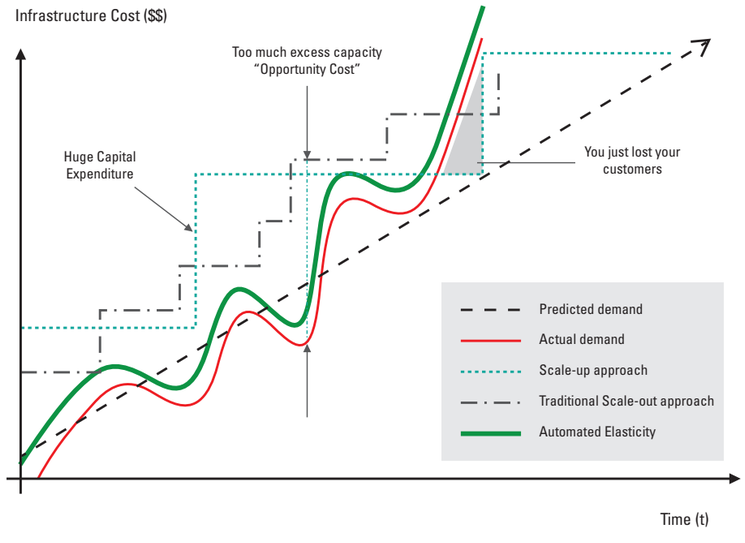
스케일 업 접근 방식

확장성을 고려하지 않고, 수요에 맞춰 더 많은 고성능 서버를 수직 확장하는 방법입니다. 일반적으로 어떠한 지점까지는 가능하지만 수요에 비해 과잉 투자를 하거나 신규 하드웨어가 구축되기 전에 용량이 초과될 수 있습니다.

기존의 스케일 아웃 (scale-out) 접근 방식

수평으로 작은 덩어리 단위로, 인프라에 투자하며 확장 아키텍처를 구성합니다. 대부분의 기업과 대규모 웹 애플리케이션은 애플리케이션 컴포넌트를 분산하며 그들의 데이터 세트 및 서비스 지향 설계를 이용함으로써 이러한 패턴을 따릅니다. 이 접근 방법은 스케일 업 방식보다 더 효과적입니다. 그러나, 이것은 여전히 정기적 수요를 예측하고 수요를 충족하기 위해 적당한 규모를 모아 인프라 구축이 이루어집니다. 이것은 초과 비용(“Burning cash”) 및 수동으로 지속적인 모니터링 (“Burning human cycles”)으로 결과가 나타납니다. 또한 어플리케이션도 작은 사이트의 링크가 갑작스럽게 인기를 끌어 폭주하게 되는 현상인 “Viral Fire”의 희생자가 될 때에는, 일시적으로 사이트가 동작하지 않을 것입니다. (Slashdot Effect라고도 함)

참고: 두 가지 접근법은 초기 비용이 투자되고, 두 방식 모두 현실 상황에 대비하여 반응합니다.



[자동화된 유연성]

전통적인 인프라는 일반적으로 응용 프로그램이 몇 년의 기간 동안 사용하는 컴퓨팅 자원의 양이 얼마인지 예측하는 일이 필요합니다. 만일 과소 추정하는 경우, 어플리케이션이 예기치 않은 트래픽을 처리하기 위해 가용량이 부족하다면 잠재적으로 고객 불만의 결과로 나타날수 있습니다. 만일 과다 추정할 경우에는, 불필요한 자원과 돈을 낭비하게 됩니다.

하지만 클라우드 방식의 특성인 온 디맨드와 탄력성은 (자동화된 유연성) 실제 수요를 인프라와 가깝게 맞출 수 있게 해주고 (확장하고 축소하면서), 따라서 전체적인 효율을 증가시키고 비용을 절감하게 해 줍니다.

탄력성은 클라우드의 기본 속성 중 하나입니다. 탄력성은 컴퓨팅 리소스를 위아래로 쉽게 확장할 수 있게 해줍니다. 궁극적으로 클라우드의 장점의 대부분과 연관된 것이 탄력성이라는 점을 이해하는 것이 중요합니다. 클라우드 아키텍트로서, 이 개념을 생각하면서 클라우드의 최대의 이익을 위해 이것을 어플리케이션 아키텍처로 녹여내야 합니다.

전통적으로, 어플리케이션은 고정적이고, 융통성이 없이 사전 구축된 인프라 위에 구축되었습니다. 회사는 일단위로 서버를 설치하고 제공할 필요가 없었기 때문에, 대부분의 소프트웨어 아키텍처들은 하드웨어의 빠른 확장 또는 축소를 해결할 수 없었습니다. 프로비저닝 시간과 새로운 자원을 획득하기 위한 선행 투자가 너무 높았기 때문에, 소프트웨어 아키텍트들은 하드웨어 사용률 최적화에 시간과 자원을 투자하지 않았습니다. 만약 응용 프로그램이 실행되는 하드웨어가 비효율적이었더라도 받아들였습니다. 빠른 시간내에 새로운 자원을 가지게 된다는 아이디어가 불가능했기 때문에 아키텍처 내에서 “탄력성”의 개념은 간과되었습니다.

클라우드와 함께 이 사고 방식을 바꿔야 합니다. 클라우드 컴퓨팅은 필요한 자원을 획득하는 과정을 간소화시킵니다. 미리 주문을 하거나 사용되지 않는 하드웨어들을 더 이상 수용하고 있을 필요가 없습니다. 대신, 클라우드 아키텍트들은 광대한 규모와 클라우드의 빠른 응답 시간을 활용하여, 수요가 일어나기 전, 미리 계획된 프로세스를 자동화하기 전에 그들이 원하는 것을 단순히 빠른 시간 내에 필요 시에 요청할 수 있습니다. 자원이 더이상 필요하지 않는 경우에 불필요한 리소스를 해제 하는 것도 적용할 수 있습니다.

이러한 변화를 수용하지 않고 어플리케이션 아키텍처에 탄력성을 구현할 수 없는 경우, 클라우드를 최대한 활용하지 못할 수 있습니다. 클라우드 아키텍트로서, 여러 분들은 창조적으로 생각해야하며, 어플리케이션에서 탄력성을 구현할 수있는 방법에 대해 생각해야 합니다. 예를 들면, 매일 밤 실행 구축하고 2 시간 동안 오전 2시에 회귀 및 단위 테스트를 수행하는 데 사용되는 인프라 (자주 불리우는 이름은 “QA/Build box”)는 나머지 시간에는 사용되지 않았습니다. 탄력적인 인프로 구조에서 우리는 밤에만 실행할 수 있는 “살아있는”, 단 2 시간 동안만 지급되는 박스를 만들 수 있습니다. 마찬가지로, 하루 동안 수요를 충족하기 위해 항상 최대 용량을 실행하는 데 사용되는 내부 장애 티켓팅 웹 응용 프로그램 (5 서버 연중 무휴 24 시간)은 온 디맨드로, 트래픽 패턴 (5서버 오전 9시에서 오후 5시까지 2 서버 오후 9시에서 오전 9시까지)에 따라 프로비저닝을 실행할 수 있습니다.

단지 필요로 할때 인프라가 실행되도록, 탄력적인 클라우드 아키텍처를 설계하는 것은 그 자체가 예술입니다. 탄력성은 디자인 요구 사항이나 시스템 특성 중 하나이어야 합니다.

우리가 물어봐야 할 질문은:

내 어플리케이션 아키텍처의 어떤 구성 요소 또는 레이어가 유연하게 될 수 있을까? 그 구성 요소를 탄력적으로 만들기 위해 무엇을 해야하나? 내 전반적인 시스템 아키텍처에 탄력성 구현의 영향은 어떻게 될 것인가?

다음 섹션에서는 어플리케이션에 탄력성을 구현하기 위한 특정 기술들을 볼 수 있습니다. 효과적으로 클라우드의 이점을 활용하려면, 아키텍트가 이 사고 방식을 가지는 것이 중요합니다.

제약 조건을 두려워하지 않음

클라우드에 어플리케이션을 마이그레이션하고 클라우드에서 사용할 수 있는 사용자 시스템 사양을 매핑하기로 결정했을 때, 여러분은 클라우드에 기존 자원과 동일한 사양이 없을 수도 있다는 것을 알 수 있습니다. 예를 들어, “클라우드는 서버에서는 RAM의 X용량을 제공하지 않습니다”또는 “내 데이터베이스에는 단일 인스턴스에서 얻을 수 있는 것보다 더 많은 IOPS가 있어야합니다.”

클라우드가 추상적인 자원을 제공하고 온 디맨드 프로비저닝 모델을 결합 할 때 그 자원들이 강력해 진다는 것을 이해해야 합니다. 클라우드 환경에서 하드웨어의 정확한 사양을 얻을 수 없는 경우에도, 그것을 극복하기 위해 클라우드에서 자원을 더 많이 얻을 수있는 능력을 가지고 있음을 이해하는 것이 중요하기 때문에, 클라우드 자원을 사용할 때 두려워하고 제한하지 않는 것이 필요합니다.

예를 들어, 클라우드 서버에 RAM에 대해서 정확하지 않거나, 그 이상의 용량을 제공하지 않는 경우, memcache 같은 분산 캐시를 사용하거나 여러 서버에 데이터를 분산해 보려고 해보셔야 합니다. 만약 데이터베이스가 더 많은 IOPS를 필요로 하고 직접 클라우드로 매핑되지 않는 경우 여러분들의 유형에 맞는 데이터 및 사용 사례에 따라 선택할 수있는 몇 가지 권장 사항이 있습니다. 읽기가 많은 응용 프로그램인 경우에는 동기화된 슬레이브 시스템으로 걸쳐 읽기 부하를 분산할 수 있습니다. 또는, 데이터 위치를 알 수 있도록, 데이터를 라우팅을 해주는 샤딩 알고리즘을 사용하거나 다양한 데이터베이스 클러스터링 솔루션을 사용할 수 있습니다.

생각해 보면, 탄력성과 온 디맨드 프로비저닝 기능을 결합할 때, 제약은 실제로 확장성 및 시스템의 전반적인 성능을 향상시킬 방법으로 나누어 질 수 있음을 알게 될 것입니다.

가상 관리

클라우드의 출현은 “가상 시스템 관리자”를 시스템 관리자의 역할로 변경시켰습니다. 이는 관리자가 어플리케이션에 대해 더 알고 사업적으로 최선을 결정을 하게 됨으로써 관리자들이 일상적으로 수행하는 작업들이 더 흥미롭게 되는 것을 의미합니다. 시스템 관리자는 더 이상 서버를 제공 할 필요가 없고 모두 몇 번의 클릭과 명령행 호출로 대체되었기 때문에 소프트웨어 및 네트워크 장치를 설치하는 고된 작업들을 실행 할 필요가 없습니다. 인프라를 프로그래밍 할 수 있기 때문에 클라우드는 자동화를 권장합니다. 시스템 관리자는 기술 스택을 늘리고 스크립트를 사용하여 추상적인 클라우드 자원을 관리하는 방법을 배울 필요가 있습니다.

마찬가지로, 데이터베이스 관리자의 역할이 웹 기반 콘솔을 통해 자원을 관리하는 “가상 데이터베이스 관리자”로 변경됨으로서, 데이터베이스 하드웨어 용량이 부족 경우 프로그램의 새로운 용량을 추가할 스크립트를 실행하고 일상적인 프로세스들을 자동화 시킵니다. 가상 DBA는 이제 새로운 배포 방법을 배우고(가상 머신 이미지), 새로운 모델을 받아들여야 하고(쿼리 병렬 처리, 지리적 중복 및 비동기 복제), 데이터에 대한 구조적인 접근 방법(샤딩, 수평 분할, 페더레이션)과 클라우드에서 사용 가능한 스토리지 옵션(서로 다른 데이터 집합을 위한)에 대해 다시 생각해야 합니다.

전통적인 기업의 회사에서, 어플리케이션 개발자는 네트워크 관리자들과 가깝게 일하지 않을 수 있으며 네트워크 관리자들은 어플리케이션에 관하여 아무것도 모를 수도 있습니다. 결과적으로, 네트워크 계층 및 응용 계층 구조에서 몇 가지 최적화가 간과됩니다. 클라우드는 이 역할을 어느 정도 하나로 병합했습니다. 미래의 어플리케이션을 설계할 때, 기업은 두 가지 역할 사이의 지식의 더 많은 교차 교류를 장려하고 각 포지션이 결합되는 것을 알아야 할 필요가 있습니다.

클라우드의 모범사례

이 섹션에서,
여러분은 클라우드에서 어플리케이션을 구축 할 수 있도록 도와주는 모범 사례에 대하여 배울 것입니다.

“실패를 위해서 디자인 하면 아무것도 실패 하지 않을 것이다.”

경험 법칙:

클라우드 아키텍처를 설계할 때 비관적이 되어라. 일이 실패할 것이라고 가정합니다. 즉, 항상 설계 하고, 장애 자동 복구에 대한 구현 및 배포합니다.

특히, 하드웨어가 장애가 발생한다고 가정해야 합니다. 정전이 발생할 것으로 가정해야 합니다. 일부 재해가 어플리케이션에 문제를 일으킬 것이라고 가정해야 합니다. 언젠가는 당신이 기대하는 초당 요청의 예상보다 더 많은 요청으로 셧다운될 것을 가정해야 합니다. 시간이 지나면 어플리케이션 소프트웨어가 장애가 발생할 것을 가정해야 합니다. 설계 기간 동안 복구 전략에 대해 고민하는 것은 결국 전체 시스템을 더 좋게 설계하도록 할 것입니다.

여러분이 확장성을 가진 인프라를 위해, 장애를 처리할 수 있는 매커니즘을 적용한 아키텍처를 생각한다면 클라우드에 최적화된 fault-tolerant 아키텍처를 만들 수 있을 것입니다.

이 모범 사례를 구현하기위한
특정 AWS 전술 :

1. 페일 오버를 위한 Elastic IP를 사용:
Elastic IP는 동적으로 다시 매핑 할 수있는 고정 IP입니다. 귀하의 트래픽이 새 서버로 라우팅되도록 신속하게 서버의 또 다른 세트에 매핑 및 장애 조치 할 수 있습니다. 하드웨어 오류의 경우 또는 이전에서 새 버전으로 업그레이드 할 때 훌륭하게 동작합니다.

2. 여러 가용 영역을 활용:
가용 영역은 논리 데이터 센터와 같은 개념입니다. 여러 가용 영역에 대한 아키텍처를 배포하여 높은 가용성을 보장 할 수 있습니다. 자동으로 여러 가용 영역에 걸쳐 데이터베이스 업데이트를 복제하는 배포 기능을 아마존 RDS Multi-AZ를 통해 활용합니다.

3. 아마존 머신 이미지를 유지, 다른 가용 영역에서 아주 쉽게 복원 및 복제, 환경 구성을 할 수 있도록 가용 영역 및 설정. 실시간 복제를 통해 여러 데이터베이스 슬레이브를 유지합니다.

4. 아마존 CloudWatch를 활용하면 성능 저하나 하드웨어 장애가 발생한 경우 적절한 조치를 할 수 있습니다. 고정된 크기를 유지하도록 오토 스케일 그룹을 설정함으로써 문제가 있는 EC2 인스턴스를 새롭게 대체합니다.

5. 아마존 EBS 활용 및 Cron jobs를 설정하여 증분 스냅 샷이 자동으로 아마존 S3에 업로드되고 데이터 인스턴스가 독립적으로 유지될 수 있도록 합니다.

6. 아마존 RDS를 활용하고 자동 백업을 수행할 수 있도록 백업의 보관 기간을 설정합니다.

필수적인 질문:

시스템의 노드에 장애가 발생하면 어떤 일이 발생할까요?
어떻게 그 실패를 인지할까요?
어떻게 그 노드를 대체할까요? 어떤 종류의 시나리오를 그 피해에 계획할까요?
단일 장애점 (Single Point of Failure)은 무엇일까요?
로드 밸런서가 어플리케이션 서버군 앞에 있을 때, 만약 그 로드 밸런서가 장애가 난다면?
아키텍처에서 마스터와 슬레이브가 있는 경우, 만약 마스터 노드에 장애가 난다면?
어떻게 장애 조치가 발생하고 그리고 어떤 방법으로 새로운 슬레이브 시스템이 인스턴스화되며 마스터와 동기화하게 될까요?

하드웨어 장애에 대한 설계처럼,
또한 소프트웨어 오류에 대한 설계도 해야합니다.

필수적인 질문:

종속 서비스가 인터페이스를 변경하는 경우 내 응용 프로그램은 어떻게 될까요?
만약 다운 스트림 서비스가 시간이 초과하거나 예외를 반환하는 경우는?
만약 캐시 키가 인스턴스의 메모리 한도를 초과 할 경우는?

그 실패를 처리하는 메커니즘을 구축해야 합니다.
예를 들어, 다음의 전략은 실패의 경우에 도움이 될 수 있습니다.

1. 데이터를 위한 일관된 백업과 복원 전략을 만들고, 자동화 시킬 것
2. 부팅시 그대로 지속되는 프로세스 스레드를 만들 것
3. 큐에서 메시지를 다시 로드 함으로 재동기화되는 시스템의 상태를 허용할 것
4. 런치/부트시에 2번과 3번 항목을 지원하는 미리 구성되고 미리 최적화된 가상 이미지를 유지할 것
5. 인 메모리 (in-memory) 세션이나 연결 지향적인 유저 컨텍스트를 피하고, 그 데이터를 데이터 저장소로 이동할 것

좋은 클라우드 아키텍처는 재부팅하고 재런치 시에 영향을 받지 않아야 한다. GrepTheWeb (클라우드 아키텍처 문서에서 설명)에서 아마존 SQS와 아마존 SimpleDB와의 조합을 사용한, 전체적인 컨트롤러 아키텍처는 이 섹션에서 나열된 오류의 유형에 매우 탄력적입니다. 예를 들어, 인스턴스가있는 컨트롤러 스레드의 실행이 멈추었을 경우, 미리 구성된 아마존 머신 이미지(AMI)를 통해 이전 상태 그대로 재가동될 수 있습니다.

하드웨어가 장애를 일으킬 것이라는 추측을 가지고 디자인을 하는 것이 미래에 실제로 장애가 발생했을 때 문제 해결에 대한 준비를 시켜줄 것입니다. 이 디자인 원칙은 운영 친화적인 어플리케이션을 설계하는 데 도움이 됩니다. 사전에 측정하여 동적으로 부하를 분산할 수 있다면 클라우드의 멀티 테넌트 특성으로 인한 네트워크와 디스크 성능의 차이를 처리할 수 있습니다.

컴포넌트 분리

클라우드는 더 느슨하게 컴포넌트들을 결합하고,
더 크고, 더 좋게 확장할 수 있는 SOA 디자인 원칙을 추구합니다.

이 모범 사례를 구현하기위한
AWS 구체적인 전술 :

1. 컴포넌트를 분리하기 위해 아마존 SQS를 사용 하라.

2. 아마존 SQS를 컴포넌트의 버퍼로 사용하라.

3. 모든 컴포넌트가 서비스 인터페이스를 가지고 자체적으로 확장성을 가지며, 다른 컴포넌트와 비동기적으로 동작하도록 설계하라.

4. 자주 배포될 수 있도록 아마존 머신 이미지(AMI)에 컴포넌트의 논리적 구조를 묶어라.

5. 가능한한 비상태적으로 어플리케이션을 만들고, 컴포넌트의 외부 에 상태 세션을 저장하라.

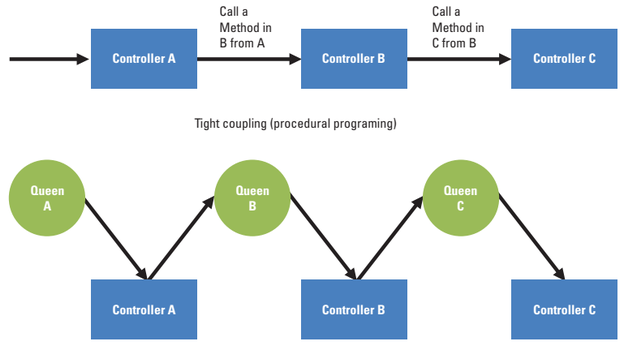
핵심은 각각의 컴포넌트가 밀결합되지 않도록 구축하는 것입니다. 밀결합되지 않는다는 것은 하나의 컴포넌트가 죽거나 응답이 없거나 응답이 느린 상태에 있더라도 다른 컴포넌트들은 영향을 받지 않고 정상 동작하는 것을 의미합니다. 본질적으로, 약결합은 다양한 계층과 어플리케이션의 컴포넌트들을 분리시킵니다. 이로 인해, 각 컴포넌트들은 비동기적으로 상호작용하게 됩니다. 예를 들어, 웹 애플리케이션 아키텍처의 경우, 웹 서버 및 데이터베이스로부터 애플리케이션 서버를 분리 할 수 있습니다. 어플리케이션 서버는 웹 서버에 대해서 모르고 반대로도 동일하게 적용이 됩니다, 이것은 각 계층사이에 분리 요소를 주고 프로그램 코드나 기능적 측면에는 종속성이 없어집니다. 일괄 처리 아키텍처의 경우, 서로 독립적일 수 있도록, 비동기적인 컴포넌트를 생성 할 수 있습니다.

필수 적인 질문:

어떤 비지니스 컴포넌트 또는 기능을 밀결합된 어플리케이션으로부터 분리시키고 독립적으로 실행되도록 할 수 있을까요? 그리고 어떻게 시스템의 중단 없이 인스턴스를 추가할 수 있을까요? 어느 정도의 노력으로 비동기적으로 다른 구성 요소와 상호 작용 할 수 있도록 구성 요소를 캡슐화 할 수 있을까요?

구성 요소를 분리하는 비동기 시스템을 구축하고자 하는 것과 수평적인 확장은 클라우드에서 매우 중요합니다. 동일한 컴포넌트를 가진 인스턴스를 수평 확장하는것 뿐만 아니라 컴포넌트의 일부는 온-프레미스(on-premise)에서 계속 실행하고 추가적인 컴퓨팅 용량과 대역폭은 클라우드를 사용함으로써 하이브리드 모델을 설계할 수도 있습니다. 최소한의 노력으로 스마트 로드밸런싱을 구현함으로써 초과되는 트래픽을 클라우드에서 처리할 수 있습니다.

메시지 큐를 사용하여 약결합된 시스템을 구축 할 수 있습니다. 큐 / 버퍼가 임의의 두 컴포넌트를 연결하는 데 사용되는 경우, 동시성, 고가용성 및 로드 스파이크 처리를 지원할 수 있습니다. 이것은 컴포넌트의 일부가 일시적으로 사용 가능하지 않더라도 전체 시스템은 지속적으로 실행되는 것을 의미합니다. 하나의 구성 요소가 죽거나 일시적으로 사용할 수 없는 경우, 시스템은 메세지를 버퍼링하고 컴포넌트가 살아나면 처리합니다.



[큐를 이용하여 컴포넌트를 분리]

클라우드 아키텍처 논문에서 주로 언급되는 GrepTheWeb 아키텍처에서 큐의 높은 사용빈도를 볼 수 있습니다. 갑자기 요청이 많아지는 과부하 상황이나 컴포넌트중 하나가 응답속도가 느려져서 정규표현식의 처리가 오래 걸리는 경우에, 다른 컴포넌트에 영향을 주지 않도록 아마존 SQS는 요청을 버퍼링합니다.

탄력성의 구현

클라우드는 어플리케이션에 탄력성이라는 새로운 개념을 제공합니다. 탄력성은 세 가지 방식으로 구현 될 수 있습니다.

1. 사전 순환 스케일링:
고정된 간격으로 발생하는 주기적인 스케일링 (일별, 주별, 월별, 분기 별)

2. 사전 이벤트 기반 스케일링:
트래픽 요청의 큰 폭주가 예상되는 경우로 인해
예약된 비즈니스 이벤트에만 스케일링 (신제품 출시, 마케팅 캠페인)

3. 수요에 따른 자동 스케일링.
모니터링 서비스를 사용함으로서, 네트워크 또는 서버의 사용률에 대한 메트릭에 기초하여 확장되거나 축소되도록 적절한 조치를 취할 수 있는 트리거를 보낼 수 있음

"탄력성"을 구현하기 위해, 먼저 배포 프로세스를 자동화하고 구성과 구축 프로세스를 간소화해야 합니다. 이 시스템은 사용자의 개입없이 확장 될 수 있음을 보장합니다. 이 결과는 전체 사용률이 비효율적으로 돌아가는 서버들보다 실질적인 수요에 가깝게 맞추는 것을 보장함으로써 전체적인 효율성이 증가하게 되며, 이로 인해 즉각적인 비용 절감 효과가 발생합니다.

인프라 자동화

클라우드 환경을 사용하는 중요한 장점 중 하나는, 배치 프로세스를 자동화하는 클라우드의 API를 사용할 수 있다는 점입니다. 마이그레이션 프로세스가 끝나기를 기다리는 것보다 마이그레이션 프로세스에 자동화된 배포 프로세스를 생성하는 것이 좋습니다. 자동 반복 배포 프로세스를 생성하는 것은 오류를 줄이고 효율적이고 확장 가능한 업데이트 프로세스를 만드는 것을 가능하게 합니다.

배포 프로세스를 자동화하려면

- "Recipes"의 라이브러리를 생성합니다.
  - 작고 자주 사용되는 설치 및 구성을 위한 스크립트
- AMI 내부에 번들 되어 있는 에이전트를 사용하여 구성 및 배치 프로세스 관리합니다.
- 인스턴스를 부트스트랩 합니다.

인스턴스를 부트스트랩 하는 방법

인스턴스가 부팅시 "나는 무엇이고 내 역할을 무엇입니까?"라는 질문을 물어 보게 합니다. 모든 인스턴스들은 "DB 서버", "어플리케이션 서버", "슬레이브 서버" 와 같은 역할이 있어야 합니다.
이 역할은 부팅 후에 인스턴스화 되는 단계에서 AMI 를 실행하는데 전달될 수 있습니다. 부팅시, 인스턴스는 역할에 따른 자원(코드, 스크립트, 구성)을 가져야 하고 클러스터에 "연결"시켜야 합니다.

인스턴스를 부트 스트랩을 구성할 때의 장점들은:
1. 몇 번의 클릭과 최소한의 노력으로 (개발, 준비, 생산) 환경을 재작성
2. 추상적인 클라우드 기반 자원을 보다 효율적으로 제어
3. 인력에 의한 배포 오류를 줄임
4. 하드웨어 장애에 더 탄력적인 자동 치유와 자동 감지 환경을 구축

인프라를 자동화 할 수 있는 AWS특정 전술 :

1. 아마존 EC2에서 아마존 자동 확장 기능을 사용하여 서로 다른 클러스터를 위한 자동 스케일링 그룹을 정의합니다.

2. 아마존 CloudWatch를 사용하여 시스템 메트릭 (CPU, 메모리, 디스크 I / O, 네트워크 I / O) 모니터링 및 또는 알림을 보낼 수 있게 하거나 자동 확장을 통해 동적으로 새로운 인스턴스를 생성하도록 합니다.

3. 저장 및 시스템 구성 정보를 동적으로 검색: 인스턴스의 부팅 시간 동안 구성 데이터를 가져 오기 위해 아마존 SimpleDB를 활용. SimpleDB는 IP 주소, 컴퓨터 이름과 같은 역할 인스턴스에 대한 정보를 저장하기 위해 사용될 수 있습니다.

4. 아마존 S3 버킷에 최신 빌드를 떨어 뜨리고, 시스템 시작 동안 어플리케이션의 최신 빌드를 다운로드하는 빌드 프로세스를 디자인하세요.

5. 리소스 관리 툴을 구축하는데 시간을 투자하거나 스마트한 오픈 소스 설정 관리 툴인 Check, Puppet, CFEngine, Genome 등을 사용하세요.

6. 딱 맞는 적합한 운영 체제와 아마존 머신 이미지에 소프트웨어 종속성을 묶으면 관리와 유지하기가 더 쉬워집니다. 시작 시간에 설정 파일이나 파라미터를 넘기고, 시작 후에 유저 데이터와 인스턴스 메타 데이터를 검색합니다.

7. 아마존 EBS 볼륨으로 부팅함으로써 번들링(bundling)과 런치(launch) 시간을 줄입니다. 일반 볼륨 스냅샷과 계정 간의 공유 스냅샷을 적당한 곳에 생성하세요.

8. 어플리케이션 컴포넌트는 실행되고 있는 하드웨어의 상태와 위치를 가정하지 않아야 합니다. 예를 들어, 클러스터에 새로운 노드의 IP를 동적으로 붙입니다. 장애 시에는 자동적으로 페일오버하고 새로운 클론을 시작하세요.

병렬적으로 생각하기

클라우드는 병렬화를 큰 노력없이 이루어낼 수 있습니다.
클라우드 아키텍트로서 클라우드 아키텍처를 설계할 때, 데이터 요청, 저장, 처리에 대해 병렬화를 고려해야 합니다. 가능한 곳에 병렬화를 구현하고, 자동화하는 것은 서비스에 도움이 됩니다. 그 이유는 클라우드는 모든 프로세스를 쉽게 반복 생성 할 수 있기 때문입니다.

데이터를 액세스(검색 및 저장)할 때에, 클라우드는 대규모 병렬 연산을 처리하도록 설계되었습니다. 최대의 성능과 처리량을 달성하기 위해, 당신은 요청 병렬화를 활용해야 합니다. 여러 개의 동시 스레드를 사용하여 요청을 멀티 스레딩하는 것은 순차적인 요청보다 저장 또는 데이터를 더 빠르게 가져옵니다. 따라서, 가능한 한 클라우드 응용 프로그램의 프로세스는 스레드 세이프하도록 만들어 져야 합니다.

클라우드에서 요청을 실행하거나 처리하는 경우, 훨씬 더 중요하게 병렬화를 활용 하게 됩니다. 웹 어플리케이션의 경우, 가장 좋은 방법은 로드 밸런서를 사용하여 여러 개의 비동기 서버로 요청을 분산하는 것입니다. 배치 프로세싱 애플리케이션의 경우, 마스터 노드는 병렬로 작업을 처리하는 여러 개의 슬레이브 워커 노드를 생성할 수 있습니다.

탄력성과 병렬화가 결합될 때 클라우드의 아름다움은 빛이 납니다. 여러분의 어플리케이션은 병렬적으로 작업을 수행할 수 있는 인스턴스를 몇개의 API로 수 분 내에 배치할 수 있고 작업이 완료되면 결과를 저장하고 모든 인스턴스를 종료할 수 있습니다.

AWS 병렬화 특정 전술 :

1. 아마존 S3 요청을 멀티스레드 합니다.

2. 아마존 SimpleDB GET 과 BATCHPUT 요청을 멀티스레드 합니다.

3. 일단위 배치 프로세스에 아마존 Elastic MapReduce를 이용, JobFlow를 생성하여 병렬적으로 작업을 수행하고 시간을 절약할 수 있습니다.

4. ELB (Elastic Load Balancer)를 사용하여 동적으로 여러 웹 어플리케이션 서버에 걸쳐 부하를 분산합니다.

컴퓨팅에 가까운 동적 데이터와 최종 사용자에 가까운 정적 데이터 유지

일반적으로 연산 및 처리에 대한 지연시간을 줄이기 위해 최대한 가깝게 데이터를 유지하는 것은 좋은 방법입니다. 클라우드에서는 인터넷의 지연에 대해 처리해야 하기 때문에 이 모범 사례는 훨씬 더 적절하고 중요합니다. 클라우드에서는 기가바이트 단위로 인바운드 및 아웃바운드 전송량에 대해 비용을 청구하기 때문에 빠르게 비용이 올라갈 수 있습니다.

프로세스가 필요한 많은 양의 데이터가 클라우드 외부에 있을 경우, 연산을 수행하기 전에 일단 클라우드로 "배송(Ship)" 및 데이터 전송을 하는 것이 더욱 저렴할 수 있습니다. 예를 들어, 데이터웨어 하우징 애플리케이션의 경우에는, 클라우드에 먼저 데이터 세트를 이동하고 데이터 세트에 대해 병렬 쿼리를 수행하는 것이 바람직합니다. 관계형 데이터베이스로부터 데이터를 저장 및 검색하는 웹 어플리케이션의 경우, 한번에 클라우드로 데이터베이스뿐만 아니라 애플리케이션 서버를 이동하는 것이 바람직합니다.

데이터가 클라우드에 생성되는 경우, 클라우드내 무료 데이터 전송 및 낮은 대기 시간의 장점을 취할 수 있도록, 데이터를 소비하는 어플리케이션 또한 클라우드내에 배치되어야 합니다. 예를 들어, 로그 및 클릭 스트림 데이터를 생성하는 전자 상거래 웹 애플리케이션의 경우에는, 클라우드 로그 분석기 및 레포팅엔진을 클라우드내에 실행하는 것이 바람직 합니다.

반대로, 데이터가 정적이고, 자주 변경되지 않을 경우, (예를 들어, 이미지, 비디오, 오디오, PDF 파일, JS, CSS 파일의 경우), 액세스 지연시간을 낮추기 위해 정적 데이터가 최종 사용자 (요청자)에 가까운 에지 로케이션에 캐시될 수 있도록 콘텐츠 딜리버리 서비스(CDN)를 활용하는 것이 바람직 합니다. 캐싱 때문에, 콘텐츠 딜리버리 서비스(CDN)는 인기있는 개체에 대한 빠른 액세스를 제공합니다.

이 모범 사례를 구현하기위한 AWS 특정 전술 :

1. AWS Import/Export서비스를 사용하여 아마존에 데이터 드라이브를 배송. 이는 인터넷을 통해 업로드하는 것보다 외장 저장 장치로 대량의 사용자 데이터를 저렴하고 빠르게 이동 할 수 있습니다.

2. 시스템의 클러스터를 실행하기 위해 동일한 가용 영역을 활용.

3. 아마존 S3 버킷의 디스트리뷰션을 생성하고, 전세계 여러 엣지 로케이션에 버킷내 컨텐츠를 클라우드 프론트가 캐시할 수 있도록 합니다.

보안 모범 사례

멀티 테넌트 (multi-tenant) 환경에서 클라우드 아키텍트들은 보안에 대한 우려를 표합니다. 보안은 클라우드 애플리케이션 아키텍처의 모든 계층에서 구현되어야 합니다. 물리적 보안은 일반적으로 서비스 제공 업체에 의해 처리됩니다. 이것은 클라우드를 사용하는 또 다른 이점입니다. 네트워크 및 애플리케이션 레벨 보안은 사용자의 책임이기 때문에, 비즈니스에 적용 가능한 모범 사례를 구현해야합니다. 이 섹션에서는, AWS 환경에서 클라우드 어플리케이션을 보호하는 방법에 대한 몇 가지 특정 도구, 기능 및 지침을 배웁니다. 이러한 도구와 기본 보안을 구현하기 위해 언급한 기능을 활용하고 적절한 표준 방법을 사용하여 추가 보안 모범 사례를 구현하는 것을 권장 합니다.

전송중인 데이터를 보호

만약 브라우저와 웹 서버 사이에 민감하거나 기밀 정보를 교환 할 필요가있을 경우, 서버 인스턴스에 SSL을 구성합니다. VeriSign 또는 Entrust 같은 외부 인증 기관에서 발급한 인증서가 필요 합니다. 인증서에 포함된 공개 키는 브라우저에 서버를 인증하고 양방향으로 데이터를 암호화하는 데 사용되는 공유된 세션 키를 생성하기 위해 사용됩니다.

명령행 호출로 가상 사설 클라우드 (VPC)를 만듭니다. 이는 AWS 내에 논리적으로 독립된 자원을 사용할 수 있도록 해주고 IPSec VPN 연결을 사용하여 여러분의 데이터 센터로 연결할 수 있습니다. OpenVPN 서버를 아마존 EC2 인스턴스에 설치하고 모든 사용자 PC에서 OpenVPN의 클라이언트를 설치할 수 있습니다.

저장된 데이터의 보호

민감하고 기밀이 필요한 데이터를 클라우드에 저장하는 것을 고려하고 있다면, 당신은 클라우드에 업로드하기 전에 데이터 (개별 파일)를 암호화 해야 합니다. 예를 들어, 아마존 S3 개체로 저장하기 전에 모든 오픈 소스나 상용 PGP-기반 도구를 사용하여 데이터를 암호화하고 다운로드 한 후 해독하면 됩니다. 이것은 HIPAA 호환 응용 프로그램을 구축 할 시에 프로텍트 헬스 인포메이션(PHI)을 저장할 필요가 있으므로, 좋은 사례가 될 수 있습니다.

아마존 EC2에서 파일 암호화는 운영 체제에 따라 달라집니다. Windows를 실행하는 Amazon EC2 인스턴스는 내장 된 파일 시스템 암호화 (EFS) 기능을 사용할 수 있습니다. 이 기능은 자동으로 파일 및 폴더의 암호화 및 암호 해독을 처리하고 과정을 투명하게 만들 것입니다. 그러나, EFS는 파일 시스템 전체를 암호화하지 않습니다. 대신, 개별 파일을 암호화합니다. 전체 암호화 된 볼륨이 필요한 경우, 오픈 소스 TrueCrypt 제품을 사용하는 것이 좋습니다; 이것은 NTFS로 포맷 된EBS 볼륨과 매우 잘 통합될 것입니다. Linux를 실행하는 Amazon EC2 인스턴스는 다양한 접근 방법 (EncFS34, 루프 AES35는 dm-crypt36, TrueCrypt37)을 사용하여 암호화 된 파일 시스템을 사용하여 EBS 볼륨을 마운트 할 수 있습니다. 마찬가지로, 오픈 솔라리스를 실행하는 Amazon EC2 인스턴스는 ZFS38 암호화 지원을 이용할 수 있습니다. 어떠한 접근 방식을 택하더라도 아마존 EC2에 파일과 볼륨을 암호화 하면 파일과 로그 데이터를 보호하는데에 도움이 됩니다. 그래서 서버내에 있는 사용자와 프로세스는 클리어 텍스트로 데이터를 볼 수 있는 반면에 서버 외부에 있는 시스템이나 사람은 암호화된 데이터 만을 보게 됩니다.

어떠한 운영체제 또는 기술을 선택하더라도 데이터 암호화는 도전 요소입니다.
데이터를 암호화하는데 사용되는 키를 관리 해야하고, 키를 분실 한 경우, 영원히 데이터를 잃게됩니다. 키가 노출 될 경우 데이터가 위험에 노출 될 수 있습니다. 따라서, 선택한 상품의 키 관리 기법을 연구해야하며 키를 분실했을 때의 위험을 최소화 하는 절차를 확립해야 합니다.

도청으로부터 데이터를 보호하는 것 외에도, 재해로부터 보호하는 방법을 고려하십시오. 아마존 EBS 볼륨에 대해 주기적으로 스냅샷을 만들면 높은 내구성 및 사용성을 보장할 수 있습니다. 스냅 샷은 아마존 S3에 자연 증분되어 자장되고 몇 번의 클릭 또는 명령행 호출로 인해 다시 복원 할 수 있습니다.

당신의 AWS 자격 증명을 보호

AWS는 보안 자격 증명을 위해 두 가지 유형(AWS 액세스 키와 X.509 인증서)을 제공합니다. 당신의 AWS 액세스 키는 두 부분으로 구성되어 있습니다. 액세스 키 ID와 비밀 액세스 키. REST 또는 쿼리 API를 사용하는 경우, 인증 인증을 위해 비밀 액세스 키를 사용해야 합니다. 데이터를 중간에 가로 채는 행위를 방지하기 위해 모든 요청을 HTTPS를 통해 전송해야 합니다.

아마존 머신 이미지(AMI)는 다른 AWS 웹 서비스와 통신하는데 필요한 프로세스를 실행하는 경우, 설계상의 공통적인 실수는 AMI내에 AWS 자격증명을 포함하는 것입니다. 자격 증명을 포함 하는 대신, 실행시 인수로 전달 하고 인터넷으로 보내기 전에 암호화 되어야 합니다.

비밀 액세스 키가 노출 될 경우, 새로운 액세스 키와 비밀 액세스 키를 생성해야 합니다. 노출된 키를 계속 사용할 수 없도록 어플리케이션 아키텍처에 키 순환 매커니즘을 적용하는 것이 좋습니다.

또는, 특정 AWS 서비스에 대한 인증을 위해 X.509 인증서를 사용할 수 있습니다. 인증서 파일은 base64로 인코딩 된 DER 내부에 공개 키가 포함되어 있습니다. 별도의 파일은 해당 base64로 인코딩 된 PKCS # 8 개인 키가 포함되어 있습니다.

AWS는 aws.amazon.com 및 AWS 관리 Console에 계정 정보와 함께 작업을 위한 추가 보호 장치로 멀티 팩터 인증을 지원합니다.

IAM을 통한 다중 사용자 및 권한 관리

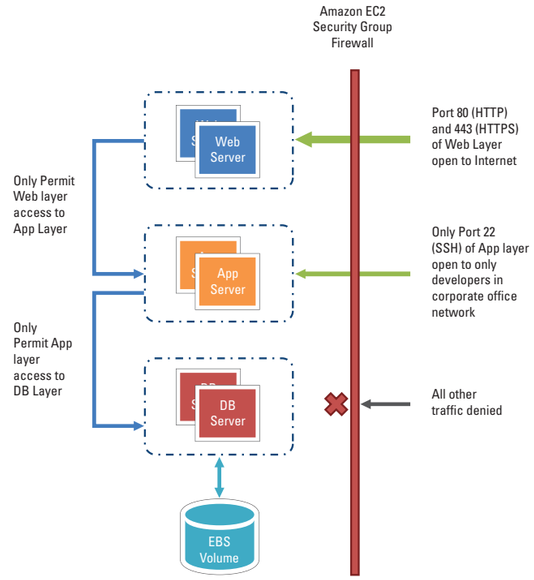
AWS ID 및 액세스 관리(IAM)는 여러 사용자를 생성하고 AWS 계정 내에서 이러한 사용자 각각에 대한 사용 권한을 관리 할 수 있습니다.사용자는 AWS 서비스에 액세스하는 데 사용할 수있는 고유한 자격 증명으로 식별합니다. IAM은 암호 또는 액세스 키를 공유 할 필요가 없고, 사용자의 액세스를 쉽게 활성화 하거나 비활성화 할 수 있습니다.

IAM은 AWS 계정내의 모든 사용자에게 최소 권한과 같은 고유 자격증명을 부여하거나, Job을 수행하는 유저에게 필요한 AWS 서비스와 자원에 대해 권한을 부여하는 것과 같은 보안 모범 사례를 구현할 수 있습니다. IAM은 기본적으로 안전합니다; 권한이 명시 적으로 부여 될 때까지 새로운 사용자는 AWS에 액세스 할 수 없습니다.

IAM은 기본적으로 대부분의 AWS 서비스에 통합되어 있습니다. 어떠한 API 서비스도 IAM을 지원하도록 변경하지 않았고 AWS 서비스API 위에 구축된 어플리케이션 및 도구는 IAM을 사용할 때 계속 작동합니다. 어플리케이션은 새로운 사용자에 대해 생성된 액세스 키만을 사용하여 시작해야 합니다.
AWS 서비스들을 사용할 때, 가능한 AWS 계정 인증의 사용을 최소화하고 IAM 사용자 인증의 장점을 활용해야 합니다.

어플리케이션 보안

모든 아마존 EC2 인스턴스는 하나 이상의 보안 그룹에 의해 보호됩니다.
TCP 및 UDP 포트, ICMP 유형과 코드 및 소스 주소를 지정할 수 있습니다. 보안 그룹은 방화벽과 같은 보호 기능을 제공합니다. 예를 들어, 웹 애플리케이션에 속하는 경우에는 다음과 같은 보안 그룹 설정을 가질 수 있습니다.



[아마존 EC2 보안 그룹을 사용하여 웹 어플리케이션 보안 하기]

들어오는 트래픽을 제한하는 또 다른 방법은 소프트웨어 기반의 방화벽을 구성하는 것입니다. 윈도우 인스턴스는 내장 firewall44를 사용할 수 있습니다. 리눅스 인스턴스는 netfilter iptables를 사용할 수 있습니다.

시간이 지남에 따라, 소프트웨어의 오류가 발견되고 해결하기 위한 패치가 필요합니다. 어플리케이션의 보안을 극대화하기 위해 다음과 같은 기본 지침을 확인해야합니다.

- 정기적으로 공급 업체의 웹 사이트에서 패치 다운로드 및 AMI업데이트.
- 새로운 AMI에서 인스턴스를 다시 배포하고 그 패치가 응용 프로그램에 어떠한 영향도 미치지 않도록 테스트 해야 합니다. 모든 인스턴스에 최신 AMI가 배포되어 있는지 확인 합니다.
- 정기적으로 보안 검사를 실행할 수 있도록 테스트 스크립트에 투자하고 프로세스를 자동화 합니다.
- 타사 소프트웨어가 가장 안전한 설정으로 구성되어 있는지 확인 합니다.
- 절대적으로 필요한 경우가 아니면 루트 또는 관리자 로그인을 통해 프로세스를 실행하지 마십시오.

클라우드 이전 시대의 표준 보안 방법인, 좋은 코딩과 민감한 데이터를 고립시키는 방법은 여전히 적용, 구현되어야 합니다.
클라우드는 물리적 보안의 복잡성을 추상화하고 어플리케이션을 안전하게 하도록 도구와 기능을 제공합니다.

www.wisen.co.kr

WiseN

Wisely Combine the Network platforms

ARCHITECTING FOR THE CLOUD

Best Practices

GS Neotek

서울특별시 구로구 경인로 576 (구로동) [TEL] 02-2630-5795 [FAX] 02-2630-5255