Amazon Web Services 에서의 개발 및 테스트

2012 년 11 월

Carlos Conde, Attila Narin

(이 문서의 최신 버전은 <http://aws.amazon.com/whitepapers/>를 참조하십시오.)

목차

요약

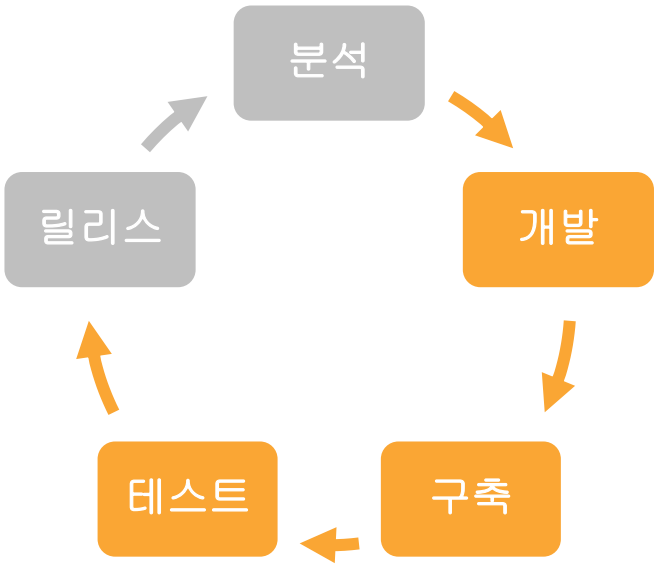
이 백서에는 Amazon Web Services(AWS)가 소프트웨어 개발 주기의 다양한 단계에서 가치를 추가하는 방법에 대해 개발과 테스트를 중심으로 설명되어 있습니다. 개발 단계에서는 버전 제어 관리에 AWS 를 사용하는 방법을 보여주고 프로젝트 관리 도구, 구축 프로세스 및 AWS 에서 호스팅되는 환경에 대해 설명하고, 모범 사례를 소개합니다. 테스트 단계에서는 테스트 환경을 관리하는 방법을 설명하고 로드 테스트, 승인 테스트, 내결함성 테스트 등 다양한 종류의 테스트를 실행합니다. AWS 는 이러한 각 시나리오와 단계에서 고유의 장점을 제공하기 때문에 소프트웨어 개발 프로젝트에 가장 적합한 시나리오와 단계를 선택할 수 있습니다. 이 백서의 대상은 프로젝트 관리자, 개발자, 테스터, 시스템 아키텍트 또는 소프트웨어 프로덕션 활동과 관련된 모든 사람입니다.

소개

소프트웨어 공급업체 조직은 기업의 핵심적인 요구 사항부터 소프트웨어 사용자 지정 또는 통합까지 다양한 이유로 소프트웨어를 제작합니다. 또한 조직은 웹 애플리케이션, 독립형 애플리케이션, 자동화된 에이전트 등 다양한 유형의 소프트웨어를 만듭니다. 그러한 경우 개발 팀은 출시 시간 또는 프로덕션 시간을 단축하기 위해 항상 높은 품질의 소프트웨어를 최대한 빨리 납품해야 하는 압박을 받고 있습니다.

이 문서에서 “개발 및 테스트”는 소프트웨어 생성 시 적용되는 다양한 도구와 정책을 가리킵니다. 개발할 소프트웨어의 유형과 관계없이, 올바른 개발 및 테스트 정책을 설정하는 것이 성공의 핵심입니다. 하지만 애플리케이션을 생성하는 작업에는 소프트웨어 엔지니어만 필요한 것이 아니라 시간, 비용, 전문 지식 등 제약을 받는 IT 리소스도 필요합니다.

소프트웨어 수명 주기는 일반적으로 다음과 같은 주요 요소로 구성됩니다.



이 백서는 다양한 개발, 구축 및 테스트 단계 분야를 다룹니다. 각 단계에 필요한 IT 인프라의 유형은 차이가 있습니다. 이 부분에서 AWS 가 소프트웨어 개발 팀에게 많은 이점을 제공합니다. AWS 는 다양한 클라우드

인프라 서비스에 대한 온디맨드 방식의 액세스를 제공하기 때문에 사용하는 리소스에 대해서만 요금이 발생합니다. AWS 는 비용이 많이 드는 하드웨어에 대한 필요와 하드웨어 소유 및 운영에 수반되는 관리상의 문제를 해결하는 데 도움이 됩니다. 하드웨어와 IT 인프라를 소유하게 되면 일반적으로 3–5 년간 자본 지출이 발생하는데, 이 기간 동안 대부분의 개발 및 테스트 팀은 몇 시간, 며칠, 몇 주, 심지어 몇 달까지 컴퓨팅과 스토리지가 필요하게 됩니다. 이렇게 소요 시간에 차이가 발생하면 IT 운영 팀도 나름대로 고정된 리소스의 제약을 받고 있어 여러 프로젝트 팀이 동시에 요청을 할 때 이를 다 충족시키기가 어렵기 때문에 마찰이 생길 수 있습니다. 결과적으로 프로젝트 팀은 리소스의 근거 설명, 소싱, 보류하는 데 많은 시간을 들이게 됩니다. 시간을 절약해 당면한 주요 작업에 집중하는 데 사용할 수 있는 시간이었습니다.

개발 단계 또는 테스트 실행 기간에 필요한 리소스만 프로비저닝함으로써[1] 회사는 기존 하드웨어에 선행 투자하는 것에 비해 꼭 필요한 비용을 절감할 수 있습니다. 적절한 수준의 확장성을 통해 각 프로젝트의 요구 사항과 예산에 따라 리소스를 할당할 수 있습니다. 그러한 경제적 이점 이외에, AWS 는 몇 주나 몇 달이 걸리던 개발 및 테스트 인프라 구축을 몇 분 만에 완료할 수 있고 IT 리소스를 필요할 때만 제공할 수 있도록 용량을 늘리거나 낮추는 확장성 등 상당한 운영상의 장점도 제공합니다.

이 문서는 AWS 에서의 개발 및 테스트와 관련된 모범 사례와 권장 사항 몇 가지를 주요 내용으로 합니다. 예를 들어 개발 단계에서 버전 제어, 공동 작업 환경, 자동화된 구축 프로세스 등 도구 및 프로세스를 안전성과 내구성을 고려해 설정하는 방법을 다루고, 테스트 단계에서 자동화된 방식으로 테스트 환경을 설정하는 방법과 항목별 테스트, 로드 테스트, 스트레스 테스트, 복원성 테스트 등 다양한 유형의 테스트를 실행하는 방법을 다룹니다.

개발 단계

팀의 규모, 개발 중인 소프트웨어의 유형 또는 프로젝트 기간에 관계없이, 프로세스 합리화, 작업 조율 및 프로덕션 중앙 집중화를 위해서는 개발 도구가 필수적입니다. 모든 IT 시스템과 마찬가지로, 개발 도구도 올바른 관리 및 유지 관리가 필요합니다. AWS 에서 그러한 도구를 운영하면 개발 팀이 네트워크 구성, 하드웨어 설정 등의 하위 수준 시스템의 유지 관리 작업을 할 필요가 없게 되며 더 복잡한 작업의 실행도 지원받을 수 있게 됩니다. 다음 섹션에서는 AWS 에서 개발 도구의 주요 구성 요소를 운영하는 방법을 설명합니다.

소스 코드 리포지토리

소스 코드 리포지토리는 개발 팀에 가장 중요한 도구입니다. 따라서 가용성이 확보되어야 하고 여기에 포함된 데이터(버전 제어를 받는 소스 파일)를 올바른 백업 정책으로 내구성을 고려해 저장해야 합니다. 이 두 가지 특징, 즉 가용성과 내구성을 보장하려면 리소스, 전문 기술 및 시간 투자가 필요한데, 이는 일반적으로 소프트웨어 개발 팀의 핵심 역량이 아닙니다.

AWS 에서 소스 코드 리포지토리를 구축하려면 Amazon Elastic Compute Cloud[2](EC2) 인스턴스를 만들고 여기에 원격으로 버전 제어 소프트웨어를 설치해야 합니다. 개발자들은 몇 분 만에 완전히 제어할 수 있는 가상 머신인 Amazon EC2 인스턴스를 만들 수 있습니다. 다양한 운영 체제와 배포를 Amazon Machine Image(AMI)로 사용할 수 있습니다. AMI 는 Amazon EC2 에서 실행할 수 있는 소프트웨어 구성이 기재된 템플릿입니다(운영 체제, 애플리케이션 서버, 애플리케이션). 소스 코드 리포지토리 인스턴스를 올바로 설치 및 구성한 다음에는

[1] 또는 전체 테스트 캠페인
[2] 참조: http://aws.amazon.com/ec2/

버전 제어 소프트웨어를 다시 설치하고 다시 구성할 필요 없이 그 인스턴스를 빠르게 다시 만들 수 있도록 이 설정에서 AMI 를 만드는 것이 좋습니다.

리포지토리의 데이터를 호스트 시스템과 따로 저장하여 유지 관리 또는 마이그레이션 작업을 간소화할 수 있습니다. Amazon Elastic Block Store[3](EBS)는 인스턴스 수명과 관계없이 지속되는 오프 인스턴스 스토리지 볼륨을 제공합니다. 볼륨을 만든 다음 Amazon EC2 인스턴스에 연결할 수 있습니다. 따라서 Amazon EBS 볼륨은 프로비저닝 및 인스턴스에 연결하여 버전 제어 리포지토리의 데이터를 저장합니다.

리포지토리 데이터가 포함된 EBS 볼륨의 지정 시간 스냅샷을 만들어 내구성을 확보할 수 있습니다. EBS 스냅샷은 내구성과 확장성이 뛰어난 데이터 스토어인 Amazon Simple Storage Service[4](S3)에 저장됩니다. Amazon S3 의 객체는 Amazon S3 리전에서 여러 시설의 다양한 디바이스에 중복 저장됩니다. 그 다음 이러한 스냅샷을 새로운 Amazon EBS 볼륨의 시작점으로 사용할 수 있고 장기적으로 내구성을 확보하여 데이터를 보호할 수 있습니다. EBS 스냅샷을 사용하면 소스 코드 리포지토리의 백업을 쉽게 관리할 수 있습니다. 장애가 발생하더라도 Amazon S3 에 저장한 스냅샷에서 리포지토리 데이터 볼륨을 다시 만들고 AMI 에서 소스 리포지토리 인스턴스를 다시 만들 수 있습니다.

엘라스틱 IP 주소는 Amazon EC2 인스턴스에 정적 엔드포인트를 제공하고 DNS 와 함께(예: DNS CNAME 뒤) 사용할 수도 있습니다. 이를 통해 확장 또는 축소를 하거나 교체 인스턴스를 커미셔닝할 때와 같이 인프라가 기저에서 변경되더라도 팀은 소스 코드 리포지토리 등의 호스팅된 서비스에 일관된 방식으로 액세스할 수 있습니다.

소스 코드 리포지토리의 규모가 커지고 더 많은 스토리지 용량이 필요하게 되면 두 가지 솔루션을 사용할 수 있습니다. 추가 Amazon EBS 볼륨을 프로비저닝하여 리포지토리 인스턴스에 연결하거나 기존 리포지토리 데이터의 최근 스냅샷을 기반으로 새롭고 더 큰 Amazon EBS 볼륨(최대 1TB)을 프로비저닝하는 방법입니다. 새로운 볼륨이 기존 볼륨을 대체하며 기존 볼륨은 삭제할 수 있습니다. 두 가지 경우 모두 새로운 Amazon EBS 볼륨이 추가 용량이 되고 온디맨드로 몇 분 만에 만들 수 있습니다.

프로젝트 관리 도구

소스 코드 리포지토리 외에도 팀에서는 문제 추적, 프로젝트 추적, 코드 품질 분석, 공동 작업, 콘텐츠 공유 같은 추가 도구를 자주 사용합니다. 대부분의 경우 이러한 도구는 웹 애플리케이션으로 제공됩니다. 기존의 다른 웹 애플리케이션과 마찬가지로 서버는 물론 관계형 데이터베이스도 자주 실행해야 합니다. 두 구성 요소 모두 기존 환경에서 사용된 것과 비슷한 배포 절차에 따라서 데이터 스토리지에 Amazon EBS 볼륨을 사용하는 데이터베이스와 함께 Amazon EC2 인스턴스에 설치할 수 있습니다.

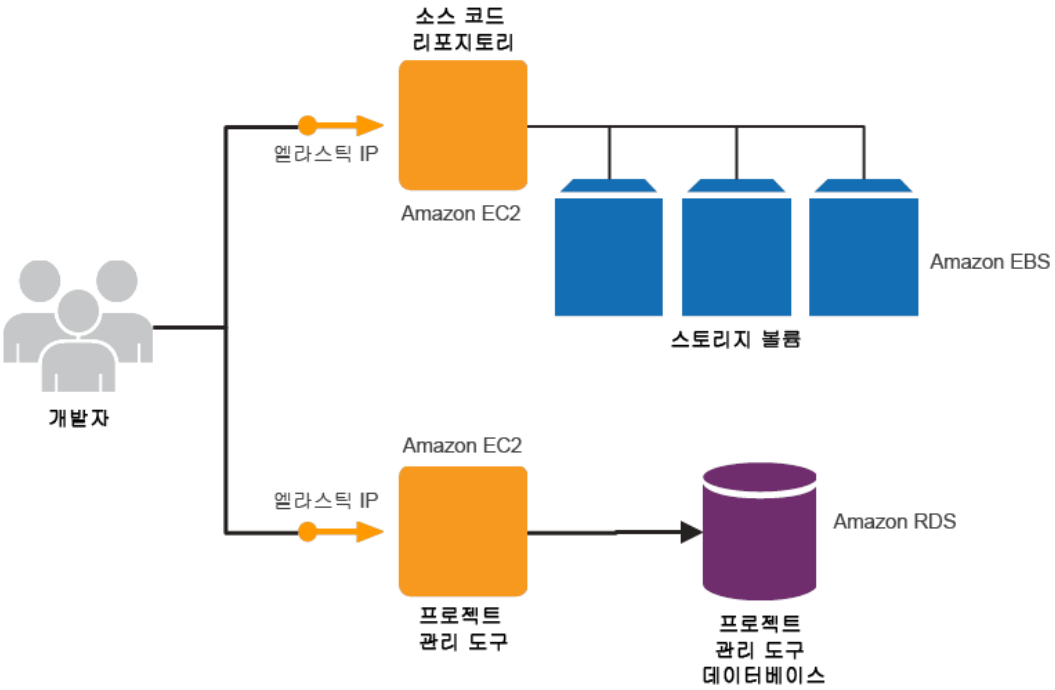
프로젝트 관리 도구는 소스 코드 리포지토리와 요구 사항이 동일합니다. 즉, 사용 가능한 상태여야 하며 내구성을 고려해 데이터를 저장해야 합니다. 원하는 리포지토리 버전에 대해 코드 분석 보고서를 재생성하여 보고서의 손실을 완화할 수 있지만, 프로젝트 또는 문제 추적 정보가 손실되면 좀 더 심각한 결과로 이어질 수 있습니다. 장애 발생 시 소스 코드 리포지토리에서와 마찬가지로 AMI 를 사용해 교체 Amazon EC2 인스턴스를 생성하여 프로젝트 관리 웹 애플리케이션 서비스의 가용성을 해결할 수 있습니다. 데이터베이스에 대해 적절한 내구성을 달성하기 위해서는 더 많은 노력과 주의가 요구됩니다. Amazon EBS 볼륨을 사용해서라도 일관성을 유지하기 위해 동결된 파일 시스템에서 스냅샷을 만들어야 합니다. 또한, 데이터베이스를 복원하려면 스냅샷에서 볼륨을 복원하고 이를 Amazon EC2 인스턴스에 연결하는 것 외에 추가 작업이 필요할 수 있습니다.

---

[3] 참조: http://aws.amazon.com/ebs/
[4] 참조: http://aws.amazon.com/s3/

이를 수월하게 진행하기 위해 Amazon Relational Database Service(Amazon RDS)는 AWS 에서 관계형 데이터베이스를 쉽게 설치, 운영, 확장할 수 있는 방법을 제공합니다. 본 서비스는 시간 소모적인 데이터베이스 관리 작업을 처리하는 한편, 비용 효율적이고 크기를 조정할 수 있는 용량을 제공하므로, 프로젝트 팀은 이런 책임에서 벗어날 수 있습니다. Amazon RDS 데이터베이스 인스턴스(DB 인스턴스)를 몇 분 만에 프로비저닝할 수 있습니다. 경우에 따라 Amazon RDS 는 최신 패치를 통해 관계형 데이터베이스 소프트웨어가 최신 상태로 유지되도록 합니다. Amazon RDS 의 백업 자동화 기능은 DB 인스턴스에 대한 지정 시간 복구가 가능하기 때문에 DB 인스턴스를 백업 보존 기간의 어느 시점으로나 복원할 수 있습니다.

개발 팀의 규모가 커지거나 프로젝트 관리 인스턴스에 추가되는 도구가 많아지면 웹 애플리케이션 인스턴스와 DB 인스턴스 모두 추가 용량이 필요할 수 있습니다. AWS 에서 인스턴스를 수직적으로 확장하는 것은 쉽고 간단한 작업입니다. 더욱 강력해진 Amazon EC2 인스턴스 유형의 AMI 에서 새로운 웹 애플리케이션 서버를 만들고 이전 서버를 교체하기만 하면 됩니다. Amazon RDS DB 인스턴스는 AWS Management Console 에서 몇 번의 클릭으로 컴퓨팅과 메모리 리소스를 확장할 수 있습니다.



참고: 더 빠르고 쉽게 배포할 수 있도록 AWS Marketplace[5] 또는 Amazon 머신 이미지[6] 에서 다양한 프로젝트 관리 도구를 사용할 수 있습니다.

온디맨드 개발 환경

개발자는 주로 로컬 랩톱 또는 데스크톱으로 개발 환경을 실행합니다. 일반적으로 이곳에 IDE 설치 및 단위 테스트를 실행하고 소스 코드를 확인합니다. 그러나 온디맨드 개발 환경을 AWS 에서 호스팅하면 도움이 되는 경우도 있습니다.

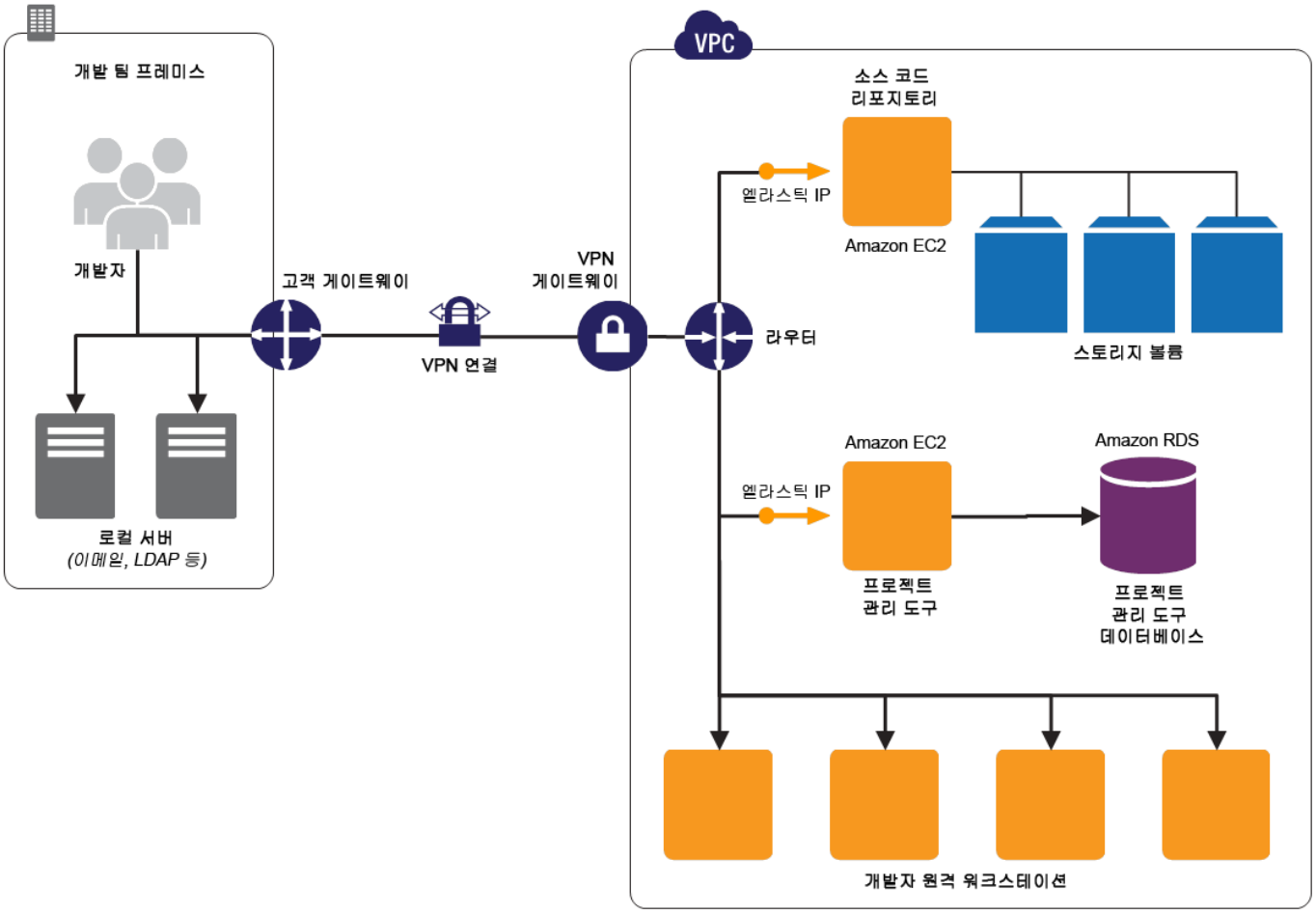
[5] 참조: https://aws.amazon.com/marketplace/b/2649274011/
[6] 참조: https://aws.amazon.com/amis

일부 개발 프로젝트는 특수 도구 집합을 사용할 수 있는데 이 경우 이러한 도구를 로컬 컴퓨터에 설치하고 유지 관리하는 일은 특히 그러한 도구를 자주 사용하지 않을 경우에는 부담이 되고 리소스 집약적이 될 수 있습니다. 그런 경우를 위해 필수 도구(개발 도구, 소스 제어, 단위 테스트 제품군, IDE 등)로 개발 환경을 준비 및 구성한 다음 AMI 로 번들링할 수 있습니다. 어떤 환경이 필요할 때 최소한의 시간과 최소한의 노력 만으로 적절한 환경을 쉽게 시작하고 즉시 실행할 수 있습니다. 그 다음에 그 환경이 더 이상 필요 없다고 판단되는 경우 종료하여 리소스를 확보할 수 있습니다. 코드를 체크아웃하고 작업 진행 중에 컨텍스트를 전환해야 하는 경우에도 도움이 될 수 있습니다. 분기를 관리하고 부분적 체크인을 처리하지 않고도 새로운 임시 환경을 가동할 수 있습니다.

AWS 에서는 다양한 인스턴스 유형에 액세스할 수 있는데, 일부 인스턴스는 하드웨어 구성이 매우 구체적으로 지정되어 있습니다. 특정 구성에 대해 구체적으로 개발을 하는 경우 시스템이 실행될 플랫폼과 같은 개발 환경을 프로덕션에도 적용하는 것이 도움이 될 수 있습니다.

호스팅되는 데스크톱 개념은 개발 환경에만 제한되지 않고 다른 역할 또는 기능에도 적용할 수 있습니다. 더 복잡한 작업 환경에서는 AWS CloudFormation 을 사용하여 AWS 리소스 그룹을 쉽게 설정할 수 있습니다. 이 주제는 테스트 환경 설정의 맥락인 아래 테스트 섹션에서 더 자세히 설명합니다. 많은 경우, 그러한 환경은 온프레미스 프라이빗 네트워크를 클라우드로 확장할 수 있는 Amazon Virtual Private Cloud(VPC) 내에서 설정할 수 있습니다. 그 다음 개발 환경이 AWS 에서 구동되는 대신 로컬 네트워크상에 있는 것처럼 프로비저닝할 수 있습니다. 온프레미스 리소스(예: LDAP)가 필요한 환경에서는 유용할 수 있습니다.

다음 다이어그램은 개발 환경이 Amazon VPC 내의 Amazon EC2 인스턴스에서 실행되는 배포를 보여줍니다. 그러한 인스턴스는 보안 VPN 연결을 통해 엔터프라이즈 네트워크에서 원격으로 액세스됩니다.



Amazon EC2 인스턴스 중지 또는 종료

예를 들어 작업 중이 아닌 시간 또는 특정 프로젝트를 보류하고 있는 경우 등 개발 환경이 사용되지 않는 경우에는 쉽게 종료하여 리소스와 비용을 절약할 수 있습니다. 이 경우 두 가지 가능성이 있습니다. 운영 체제 휴면과 거의 비슷하게 인스턴스를 중지하는 방법 또는 운영 체제를 폐기하는 것과 거의 비슷하게 인스턴스를 종료하는 방법입니다.

인스턴스를 중지하면(Amazon EBS 지원 AMI 의 경우 가능) 컴퓨팅 리소스가 릴리스되고 해당 인스턴스에 대해 시간당 요금이 더 이상 적용되지 않습니다. Amazon EBS 볼륨이 상태를 저장하고 다음에 인스턴스를 시작하면 중지하기 전과 같은 작업 데이터가 유지되기 때문에 랩톱을 여는 것과 동일합니다. (참고: 중지/시작 시퀀스 후에는 휘발성 드라이브에 저장된 어떤 데이터도 사용할 수 없습니다.)

인스턴스를 종료하면 인스턴스 시작 중에 연결되는 루트 디바이스와 그 외 모든 디바이스가 자동으로 삭제되므로(볼륨의 DeleteOnTermination 플래그를 “false”로 설정하지 않은 경우) 삭제한 볼륨에 사용할 수 있는 백업 또는 스냅샷이 없는 경우 데이터가 손실될 수 있습니다. 종료된 인스턴스는 더 이상 존재하지 않으며 필요한 경우 AMI 에서 다시 만들어야 합니다. 모든 작업을 커밋했거나 특정 환경을 더 이상 사용하지 않을 경우에는 일반적으로 개발 환경의 인스턴스를 종료합니다.

AWS API 및 IDE 기능 향상과의 통합

AWS 에서는 프로젝트의 대상 프로젝트가 AWS 이거나 프로젝트가 AWS 의 리소스를 조율하는 것일 경우 이제 IT 인프라에 대해 코딩 및 제어할 수 있습니다. 그런 경우 다양한 AWS SDK 를 사용하여 애플리케이션을 쉽게 AWS API 와 통합할 수 있기 때문에 웹 서비스 인터페이스에 대해 직접 코딩하고 인증, 다시 시도, 오류 처리 등에 관한 세부 정보를 처리하는 복잡한 문제를 없앨 수 있습니다. AWS SDK 도구는 Java[7], .Net[8], PHP[9], Ruby[10] 등 다양한 언어와 모바일 플랫폼[11] Android 및 iOS 에서 사용할 수 있습니다.

또한 AWS 는 AWS Toolkit for Visual Studio 와 AWS Toolkit for Eclipse 등의 도구[12]를 제공하여 더욱 쉽게 IDE 내에서 AWS 와 상호 작용할 수 있도록 합니다.

구축 단계

애플리케이션을 구축하는 프로세스에는 컴파일, 리소스 생성 및 패키지 등 여러 단계가 필요합니다. 규모가 큰 애플리케이션의 경우 각 단계에 내부 라이브러리 구축, 헬퍼 애플리케이션 사용, 다양한 형식의 리소스 생성, 문서 생성 등 여러 가지 종속성이 관련됩니다. 일부 프로젝트는 다양한 CPU 아키텍처, 플랫폼 또는 운영 체제에 대한 제공품의 구축이 필요할 수 있습니다. 결국, 전체 구축 프로세스는 몇 시간이 걸릴 수 있고 이로 인해 소프트웨어 개발 팀의 대응 능력에 직접적인 영향을 줍니다. 소스 리포지토리에 대한 모든 약정으로 인해 자동화된 빌드가 트리거되는 지속적 통합[13]과 테스트 제품군과 같은 접근 방식을 도입하는 팀의 경우 영향은 훨씬 더 큽니다.

---

[7] 참조: http://aws.amazon.com/sdkforjava/
[8] 참조: http://aws.amazon.com/sdkfornet/
[9] 참조: http://aws.amazon.com/sdkforphp/
[10] 참조: http://aws.amazon.com/sdkforruby/
[11] 참조: http://aws.amazon.com/mobile/
[12] 참조: http://aws.amazon.com/developertools/
[13] 참조: http://en.wikipedia.org/wiki/Continuous_integration

야간 구축

이 문제를 완화하기 위해 구축 시간이 긴 프로젝트를 작업하고 있는 팀은 "야간 구축"(또는 중립 구축[14]) 접근 방식을 도입하거나 프로젝트를 작은 단위의 하위 프로젝트로 나누는 경우가 많습니다(두 가지 접근 방식을 결합하기도 함). 야간 구축을 하려면 리포지토리에서 최신 소스 코드를 체크아웃하고 제공품을 야간에 구축하는 구축용 컴퓨터가 필요합니다. 개발 팀은 원하는 만큼 많은 버전을 구축할 수 없고 빌드가 제때에 완성되어야 다음 날 테스트를 시작할 수 있습니다. 각 하위 프로젝트가 더 빨리 독립적으로 구축되는 경우 프로젝트를 더 작고 관리가 편리한 여러 부분으로 분리하는 것도 해결 방법이 될 수 있습니다. 하지만 팀이 전체 프로젝트를 계속 관찰해야 하고 다양한 부분이 함께 계속 잘 작동하는지 확인하려면 다양한 하위 프로젝트를 모두 결합하는 통합 단계가 필요한 경우가 많습니다.

온디맨드 구축

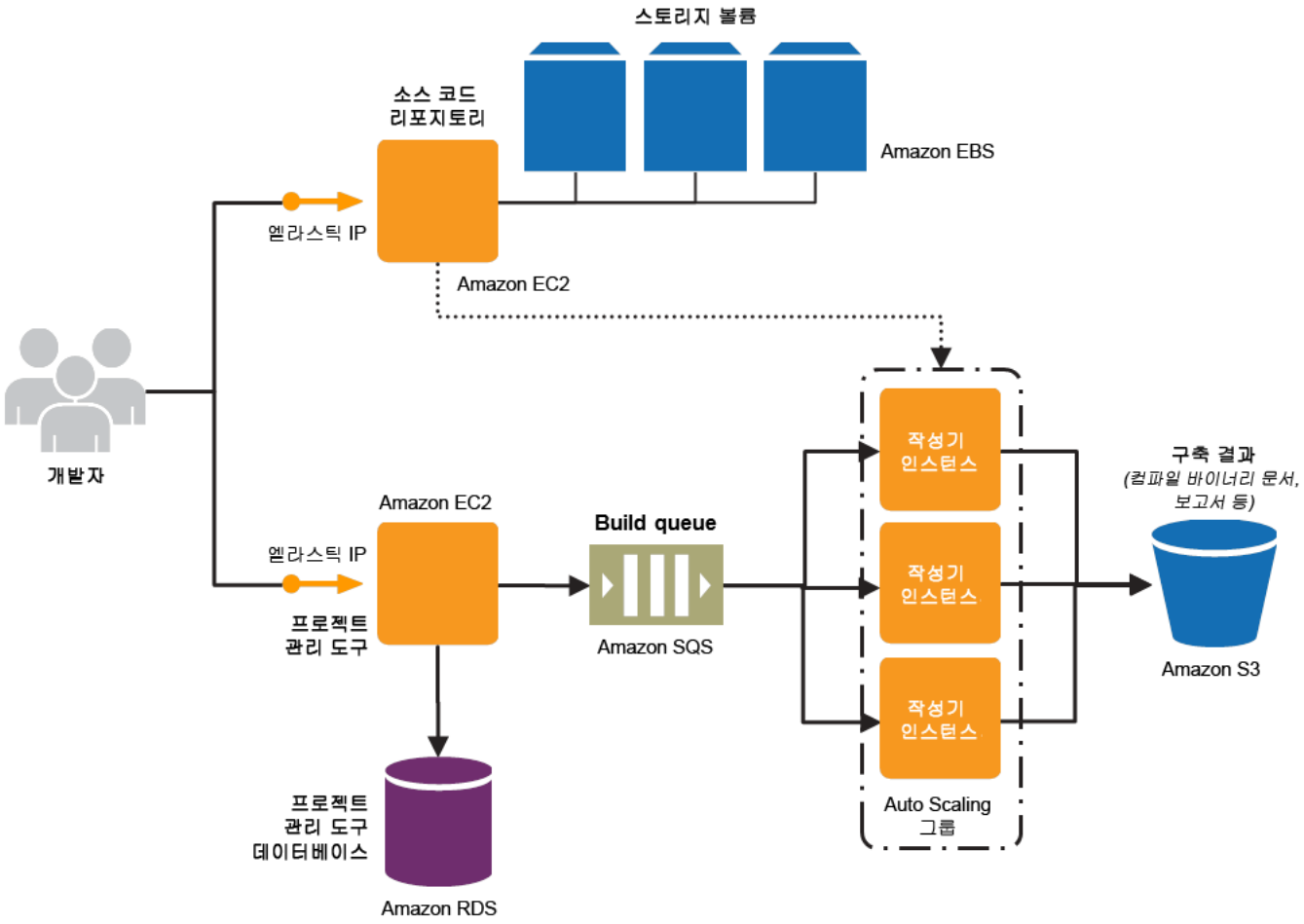
더 유용한 솔루션은 구축 프로세스에 컴퓨팅 기능을 더 많이 사용하는 것입니다. 빌드 서버가 조직이 구입한 하드웨어에서 실행되는 기존 환경에서는 경제적인 제약 또는 프로비저닝 지연으로 인해 이 옵션이 현실적이지 않을 수 있습니다. 이전 섹션에서 이미 설명했듯이 Amazon EC2 인스턴스에 실행되는 빌드 서버는 몇 분 만에 수직 확장이 가능하기 때문에 필요할 때 더 많은 CPU 또는 메모리 용량을 제공하여 구축 시간을 단축합니다.

같은 날 여러 개의 빌드가 트리거되는 팀의 경우 단일 Amazon EC2 인스턴스로는 빌드를 필요한 만큼 빠르게 생성할 수 없을 수 있습니다. Amazon EC2 의 온디맨드 및 선불형 종량 과금제 특성을 활용하여 여러 가지 빌드 인스턴스(작업자 노드)를 사용하는 것이 해결책이 될 수 있습니다. 개발 팀에서 새로운 빌드를 요청하거나 소스 코드 리포지토리에 대한 새로운 약정으로 인해 새로운 빌드가 트리거될 때마다 구축 프로세스를 작업자 노드 집합에 배포합니다. 처리할 모든 빌드가 포함된 대기열을 사용하여 작업자 노드로 작업을 배포할 수 있습니다. 작업자 노드는 다음 빌드가 확보될 때마다 처리할 다음 빌드를 선택합니다. 이 시스템을 구현하기 위해 Amazon Simple Queue Service[15](SQS)는 안정적이고 확장성이 뛰어난 호스팅 대기열 서비스를 제공합니다. Amazon SQS 는 Amazon EC2 및 다른 AWS 인프라 서비스와 긴밀하게 작동하여 자동화된 구축 워크플로우를 쉽게 만들게 해줍니다. 이 설정에서 개발자는 소스 코드 리포지토리에 코드를 커밋하고 Amazon SQS 대기열에 구축 메시지를 푸시합니다. 이 대기열은 사용자 노드로 폴링됩니다. 작업자 노드는 메시지를 가져오고 메시지에 포함된 파라미터(예: 사용할 분기 또는 소스 버전)에 따라 로컬에서 빌드를 실행합니다.

대기열을 사용하는 작업자 노드 풀을 동적으로 조정하여 이 설정을 더욱 향상시킬 수 있습니다. Auto Scaling[16]은 사전 정의된 조건에 따라 쉽게 숫자 또는 작업자 노드를 자동으로 확장 또는 축소할 수 있는 서비스입니다. Auto Scaling 을 사용하면 수요가 급격하게 증가할 경우 작업자 노드 용량을 원활하게 늘려 빠른 빌드 생성 속도를 유지하고 수요가 줄 경우 인스턴스 수를 자동으로 줄여서 비용을 최소화할 수 있습니다. AWS 클라우드 리소스 모니터링 서비스인 Amazon CloudWatch[17]를 사용하여 조정 조건을 정의할 수 있습니다. 예를 들어 Amazon CloudWatch 는 빌드 대기열의 메시지 수를 모니터링하고 대기열의 메시지 수에 따라 Auto Scaling 에 용량 확장 또는 축소 필요를 알릴 수 있습니다. 다음 다이어그램은 이 시나리오를 요약한 것입니다[18].

---

[14] 참조: http://en.wikipedia.org/wiki/Nightly_build
[15] 참조: http://aws.amazon.com/sqs/
[16] 참조: http://aws.amazon.com/autoscaling/
[17] 참조: http://aws.amazon.com/cloudwatch/
[18] 일반 설정은 참조 아키텍처 "배치성 프로세스"에 설명되어 있습니다. 참조: http://d36cz9buwru1tt.cloudfront.net/architecturecenter/AWS_ac_ra_batch_03.pdf



구축 결과 저장 및 배포

빌드를 생성할 때마다 그 결과를 어딘가 저장해야 합니다. Amazon S3 는 이 목적에 적절한 서비스입니다. 초기에는 지정된 프로젝트에 저장할 데이터의 양이 적지만 시간이 지나면서 빌드를 더 많이 생성하면서 규모가 커지게 됩니다. 이 부분에서 Amazon S3 의 선불형 종량 과금제 및 용량 특성이 더욱 매력적입니다. 구축 결과가 더 이상 필요 없을 때는 삭제하거나 Amazon S3 의 데이터 보존 정책을 사용해 삭제할 수 있습니다.

테스트, 스테이징 또는 프로덕션에 배포하거나 클라이언트에 다운로드할 목적으로 구축 결과를 배포할 수 있도록, AWS 는 다양한 옵션을 제공합니다. Amazon S3 에서 직접, 공개적으로 또는 버킷 정책 및/또는 ACL 배포를 제한하도록 구성하여 구축 결과 패키지를 배포할 수 있습니다. 또 다른 옵션은 콘텐츠 제공을 위한 웹 서비스인 Amazon CloudFront[19]를 사용하는 것입니다. 이를 통해 짧은 지연 시간과 빠른 데이터 전송 속도로 최종 사용자에게 패키지를 편리하게 배포함으로써 최종 사용자 경험을 향상시킬 수 있습니다. 이는 다수의 클라이언트가 설치 패키지 또는 업데이트를 다운로드하는 경우에 유용할 수 있습니다. Amazon CloudFront 는 예를 들어 액세스 승인 및/또는 제한과 같은 다양한 옵션을 제공합니다. 단, 자세한 내용은 본 문서의 범위에 포함되지 않습니다.

[19] 참조: http://aws.amazon.com/cloudfront/

테스트 단계

테스트는 소프트웨어 개발의 필수적인 부분입니다. 테스트를 통해 소프트웨어 품질을 확보할 수 있으며 더 중요하게는 개발 단계 초기에서 문제를 찾아내 프로젝트 후반기의 문제 해결 비용을 낮출 수 있습니다. 테스트의 형태는 다양합니다. 단위 테스트, 성능 테스트, 사용자 승인 테스트, 통합 테스트 등이 있으며 모두 IT 리소스가 있어야 실행할 수 있습니다. 따라서 테스트 팀은 개발 팀과 같은 문제에 당면하게 됩니다. 충분한 IT 리소스를 확보하되 제한된 테스트 실행 기간 중에만 사용하는 것입니다. 게다가 테스트 환경이 자주 바뀌고 프로젝트마다 다르기 때문에 프로젝트마다 다른 IT 인프라가 필요하거나 용량 요구 사항이 다를 수 있습니다.

AWS 의 온디맨드 및 선불형 종량 과금제 가치 제안은 이러한 제약에 유연하게 적용할 수 있습니다. AWS 는 테스트 팀이 비용이 많이 드는 하드웨어에 대한 필요와 하드웨어 소유 및 운영에 수반되는 관리상의 문제를 해결하는 데 도움이 됩니다. 또한, AWS 는 테스터에게도 상당한 운영상의 장점을 제공합니다. 기존에 몇 주 내지 몇 달이 걸리던 테스트 환경 설정을 몇 분 만에 완료할 수 있고 다양한 인스턴스 유형과 같은 다양한 리소스를 테스트 실행이 필요할 때마다 사용할 수 있습니다.

테스트 환경 자동화

테스트 실행을 자동화하는 다양한 소프트웨어 도구 및 프레임워크가 있지만 그러한 테스트를 실행하려면 올바른 인프라가 준비되어 있어야 합니다. 그렇게 하려면 인프라 리소스를 프로비저닝하고 샘플 데이터 세트로 초기화하며, 테스트할 소프트웨어를 배포하고 테스트 실행을 조율하여 결과를 수집해야 합니다. 이 부분에서 발생하는 문제는 필요할 수 있는 다양한 서버 또는 서비스에 완벽한 애플리케이션을 배포할 충분한 리소스를 확보하는 것과 더불어 적절한 소프트웨어와 적절한 데이터를 사용하여 반복적으로 테스트 환경을 초기화하는 기능을 확보하는 것입니다. 모든 테스트 실행의 테스트 환경은 동일해야 합니다. 그렇지 않으면 결과를 비교하기가 더 어려워집니다.

AWS 에서 테스트를 실행하는 또 다른 이점은 다양한 방식으로 자동화하는 기능입니다. AWS API 또는 명령줄 인터페이스(CLI) 도구를 사용하여 프로그래밍 방식으로 AWS 를 운영할 수 있습니다. 기존 환경에서 사용자의 개입이 필요한 작업(새로운 서버 할당, 스토리지 할당 및 연결, 데이터베이스 할당 등)은 AWS 에서 완전히 자동화할 수 있습니다. 테스터는 AWS 에서 테스트 제품군을 설계하려면 기존에는 정적 하드웨어 디바이스였던 구성 요소 운영 단위까지 자동화해야 합니다. 자동화를 통해 테스트 환경을 만들고 초기화하는 노력이 들지 않기 때문에 테스트 팀은 더욱 효율적으로 작업할 수 있고 환경을 만드는 동안 사용자의 개입을 제한하여 오류 발생을 낮출 수 있습니다. 자동화된 테스트 환경은 지속적인 통합 원칙에 따라 구축 프로세스에 연결할 수 있습니다[20]. 성공적인 빌드를 생성할 때마다 테스트 환경을 프로비저닝하고 그 환경에서 자동화된 테스트를 실행할 수 있습니다.

다음 섹션은 Amazon EC2 인스턴스, 데이터베이스 및 전체 환경을 자동으로 프로비저닝하는 방법을 설명합니다.

인스턴스 프로비저닝

AMI 에서 Amazon EC2 인스턴스를 쉽게 프로비저닝할 수 있습니다[21]. AMI 는 운영 체제 및 인스턴스에 사전 설치된 다른 소프트웨어 또는 구성 파일을 캡슐화합니다. 인스턴스를 시작하면 모든 애플리케이션은 AMI 에서 이미 로드되고 실행할 준비가 되어 있습니다. AMI 만들기에 관한 내용은 Amazon EC2 문서[22]를

[20] 참조: http://en.wikipedia.org/wiki/Continuous_integration
[21] 참조: http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/launching-an-instance.html
[22] 참조: http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/creating-an-ami.html

참조하십시오. AMI 기반 배포의 문제는 소프트웨어를 업그레이드할 때마다 새로운 AMI 를 만들어야 한다는 것입니다. 새 AMI 를 만들고 기존 AMI 를 삭제하는 프로세스를 완전히 자동화할 수 있지만 얼마 되지 않아 다양한 버전의 AMI 를 관리 및 유지 관리하기 위한 전략을 정의해야 하는 문제가 생깁니다.

대안으로 사용할 수 있는 접근 방식은 자주 변경되지 않는 AMI(운영 체제, 언어 플랫폼 및 하위 수준 라이브러리, 애플리케이션 서버 등)에만 구성 요소를 포함시키는 방법입니다. 개발 중인 애플리케이션처럼 휘발성이 높은 구성 요소는 실행 시간에 가져와 인스턴스에 배포합니다. 자가 부트스트랩 인스턴스를 만드는 방법에 대한 자세한 내용은 "AWS 에서 애플리케이션 부트스트랩"[23]을 참조하십시오.

데이터베이스 프로비저닝

테스트 데이터베이스를 Amazon RDS 데이터베이스 인스턴스로 효율적으로 구현할 수 있습니다. 사용자의 테스트 팀은 종합 운영 데이터베이스를 쉽게 인스턴스화하고 스냅샷에서 테스트 데이터 세트를 로드할 수 있습니다. 이 테스트 데이터 세트를 만들려면 먼저 Amazon RDS 인스턴스를 프로비저닝해야 합니다. 데이터 세트를 주입한 후 인스턴스의 스냅샷을 만듭니다[24]. 그 후에는 테스트 환경에 테스트 데이터베이스가 필요할 때마다 쉽게 초기 스냅샷에서 Amazon RDS 인스턴스로 만들 수 있습니다[25]. 동일한 스냅샷에서 시작된 각 Amazon RDS 인스턴스에는 동일한 데이터 세트가 포함되어 있기 때문에 테스트의 일관성을 유지하는 데 도움이 됩니다.

전체 환경 프로비저닝

AWS API, 명령줄 도구 또는 AWS Management Console 을 사용하여 여러 개의 인스턴스가 포함된 복잡한 테스트 환경을 만들 수는 있지만 AWS CloudFormation[26]을 사용하면 훨씬 더 쉽게 관련 AWS 리소스 모음을 만들고 질서 있고 예측 가능한 방식으로 프로비저닝할 수 있습니다. AWS CloudFormation 은 템플릿을 사용하여 리소스 모음을 단일 유닛(스택)으로 생성 및 삭제할 수 있습니다. AWS 에서 실행되는 전체 테스트 환경은 JSON 형식의 텍스트 파일인 템플릿으로 설명할 수 있습니다. 템플릿은 텍스트 파일에 불과하기 때문에 소프트웨어 개발 프로젝트에 사용하는 것과 동일한 소스 코드 리포지토리에서 편집 및 관리할 수 있습니다. 그런 방식으로 템플릿은 프로젝트의 상태를 미러링하고 기존 소스 버전과 일치하는 테스트 환경을 쉽게 프로비저닝할 수 있습니다. 이 점은 회귀 버그를 처리할 때 특히 유용합니다. 몇 단계만 거치면 전체 테스트 환경을 프로비저닝할 수 있기 때문에 개발자와 테스터가 소프트웨어 기존 버전에서 감지한 버그를 시뮬레이션할 수 있습니다. 또한, AWS CloudFormation 템플릿은 로드할 특정 소프트웨어 버전, 테스트 환경의 Amazon EC2 인스턴스 크기, 데이터베이스에 사용할 데이터 세트 등을 지정하는 데 사용할 수 있는 파라미터를 지원합니다.

AWS 에서 AWS CloudFormation 을 사용하여 배포를 만들고 자동화하는 방법에 대한 자세한 내용은 주제를 참조하십시오. http://aws.amazon.com/cloudformation/aws-cloudformation-articles-and-tutorials/

로드 테스트

제어되는 환경에서 실행되는 기능 테스트는 소프트웨어 품질을 보장하는 중요한 도구이지만 로드가 과도할 경우 애플리케이션 또는 전체 배포의 성능에 대한 정보는 거의 제공하지 않습니다. 예를 들어 일부 웹 사이트는 스포츠 행사 티켓 판매, 특별 세일(블랙 프라이데이), 한정판 출시 등 제한된 시간 동안 서비스를

[23] 참조: https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf
[24] 참조: http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html
[25] 참조: http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_RestoreFromSnapshot.html
[26] 참조: http://aws.amazon.com/cloudformation/

제공하는 용도로 구체적으로 만들어집니다. 그러한 웹 사이트는 피크 사용 기간 중에 효율적으로 작동하도록 개발 및 설계해야 합니다.

경우에 따라 프로젝트 요구 사항에는 로드가 과도할 경우 충족해야 하는 최소 성능 지표가 정확하게 명시되어 있는데(예: 동시 요청 건수가 최대 10,000 건 이하일 때 검색 결과가 100ms 이내로 반환되어야 함) 로드 테스트는 시스템이 그러한 한계 이내에서 로드를 유지할 수 있는지 확인하기 위해 실시됩니다. 다른 경우에는 시스템이 유지해야 하는 로드를 지정하는 것이 불가능하거나 유용하지 않습니다. 그런 경우에는 과도한 로드 조건에서 작동을 측정하기 위해 로드 테스트를 수행합니다. 목표는 시스템의 로드를 점진적으로 늘려 시스템이 더 이상 작동하지 못할 정도로 성능이 저하되는 지점을 확인하는 것입니다.

로드 테스트는 시스템을 연습하고 스트레스를 인가하는 과도한 입력을 시뮬레이션합니다. 프로젝트에 따라 입력은 동시에 수신되는 대량의 요청, 처리해야 하는 방대한 데이터 세트 등이 될 수 있습니다. 로드 테스트의 주요 문제 중 하나는 대량의 입력을 생성하여 테스트하는 시스템을 한계로 내몰 수 있다는 것입니다. 일반적으로 대량의 IT 리소스를 시스템에 배포하여 테스트하고 테스트 입력을 생성하는 것을 의미하는데 여기에는 추가 인프라가 필요합니다. 로드 테스트는 대개 몇 시간만 실행되기 때문에 AWS 선불형 종량 과금제 모델이 이 사용 사례에 아주 적합합니다.

또한 이전 섹션에서 설명한 기술을 사용하여 로드 테스트를 자동화할 수 있기 때문에 테스터가 대규모 변경 사항이 있을 때마다 시스템 성능과 효율성에 악영향을 미치지 않는지 확인할 수 있도록 더 자주 연습할 수 있습니다. 반대로 자동화된 로드 테스트를 시작하면 새 알고리즘, 캐싱 계층 또는 아키텍처 설계가 더 효율적이고 프로젝트에 도움이 되는지 쉽게 확인할 수 있습니다.

> *참고: 빠르고 쉽게 설정할 수 있도록, AWS Marketplace 에서 테스트 도구 및 솔루션을 사용할 수 있습니다. https://aws.amazon.com/marketplace/b/2649283011/*

네트워크 로드 테스트

애플리케이션 또는 서비스의 네트워크 로드를 테스트하려면 테스트하는 시스템에 많은 수의 요청을 전송해야 합니다. 요청 시나리오를 시뮬레이션할 수 있는 소프트웨어 솔루션은 많지만 충분한 트래픽을 생성하기 위해서는 여러 개의 Amazon EC2 인스턴스를 사용해야 합니다. Amazon EC2 인스턴스는 온디맨드로 사용할 수 있고 시간당 요금을 청구하므로 네트워크 로드 테스트 시나리오에 매우 적합합니다. 다양한 인스턴스 유형의 특성을 염두에 두는 것이 중요합니다. 일반적으로 인스턴스 유형의 크기가 클수록 I/O 네트워크 용량을 더 많이 제공하고 네트워크 로드 테스트 중에 기본 리소스가 사용됩니다.

AWS 를 사용하면 테스트 팀이 AWS 외부에서 실행되는 애플리케이션에 대해서도 네트워크 로드 테스트를 수행할 수 있습니다. 로드 테스트 에이전트를 AWS 의 여러 리전에 분산해 여러 지역에서 테스트를 할 수 있기 때문에 최종 사용자 경험을 더 잘 파악하는 등의 효과를 볼 수 있습니다. 그러한 시나리오에서는 로드를 시뮬레이션하는 인스턴스에서 로그 정보를 수집하는 것이 합리적입니다. 그러한 로그에는 테스트 시스템으로부터 수신한 응답 시간 등의 중요한 정보가 포함되어 있습니다. 다양한 리전으로부터 로드 에이전트를 실행하면 다양한 지역에 대해 테스트한 애플리케이션의 응답 시간을 측정할 수 있습니다. 이를 통해 전 세계 사용자 경험을 파악할 수 있습니다. Amazon EC2 인스턴스 로드 테스트는 테스트 직후 종료할 수 있기 때문에 스토리지 및 이후 분석 시 Amazon S3 에 로그 데이터를 전송해야 합니다.

AWS 로드 테스트

AWS 에서 실행되는 애플리케이션을 로드 테스트하면 탄력성 기능이 올바로 구현되는지 확인하는 데 도움이 됩니다. 시스템의 네트워크 로드 테스트는 웹 프런트 엔드, Auto Scaling 및 Elastic Load Balancing[27] 구성이 올바른지 확인하는 데 중요합니다. Auto Scaling 은 여러 파라미터를 제공하고[28] Amazon CloudWatch 로 정의한 여러 조건을 사용하여 프런트 엔드 인스턴스의 수를 확장하거나 축소할 수 있습니다. 이러한 파라미터 및 조건은 Auto Scaling 그룹이 인스턴스를 추가 제거하는 속도에 영향을 미칩니다. Amazon EC2 인스턴스의 프로비저닝 후 시간도 애플리케이션이 원하는 만큼 빠른 속도로 확장하는 기능에 영향을 미칠 수 있습니다. Amazon EC2 인스턴스에서 실행되는 운영 체제가 초기화된 후, 웹 서버, 애플리케이션 서버, 메모리 캐시, 미들웨어 서비스 등의 추가 서비스가 초기화됩니다. 이 다양한 서비스의 초기화 시간은 추가 소프트웨어 패키지를 리포지토리에서 가져와야 하는 경우 특히 확장 지연에 영향을 미칩니다. 로드 테스트를 통해 추가 용량을 얼마나 빠르게 특정 시스템에 추가할 수 있는지 중요한 지표를 얻을 수 있습니다.

Auto Scaling 은 프런트 엔드 시스템에만 사용되는 것이 아닙니다. 소비자가 Amazon SQS 대기열을 폴링하는 경우[29] 또는 작업자 및 결정자가 Amazon Simple Workflow Service(SWF) 워크플로우에 참여하는 경우[30]와 같은 내부 인스턴스 그룹을 확장하는 데 사용할 수도 있습니다. 두 경우 모두 시스템을 로드 테스트하면 Auto Scaling 그룹 또는 다른 자동화 확장 기술이 올바로 구현 및 구성되었는지 확인하여 최종 애플리케이션의 비용 효율성 및 확장성을 최대한 높이는 데 도움이 될 수 있습니다.

스팟 인스턴스를 사용한 비용 최적화

로드 테스트는 대량의 로드를 지원하도록 설계된 시스템을 연습할 때 특히 많은 인스턴스가 필요합니다. 시간당 요금만 지급하고 온디맨드로 Amazon EC2 인스턴스를 프로비저닝하여 테스트가 완료되면 폐기할 수 있지만, Amazon EC2 스팟 인스턴스를 사용하여 그 테스트를 수행하면 더욱 비용 효율적인 방법이 됩니다. 스팟 인스턴스[31]를 사용하면 미사용 Amazon EC2 용량에 입찰할 수 있습니다. 이러한 인스턴스에는 Amazon EC2 가 책정하는 스팟 가격이 적용되며 이 가격은 스팟 인스턴스 용량에 대한 수요와 공급에 따라 변경됩니다. 스팟 인스턴스를 사용하려면 스팟 인스턴스 요청을 배치하여 인스턴스 유형, 원하는 가용 영역[32], 실행할 스팟 인스턴스의 수, 시간당 지불하려는 최고 가격을 지정해야 합니다. 최근 90 일의 스팟 가격 기록은 Amazon EC2 API 및 AWS Management Console[33]을 통해 볼 수 있습니다. 최고 가격의 입찰이 현재 스팟 가격을 초과하는 경우 요청이 이행되며 인스턴스가 시작되어 종료되거나 스팟 가격이 최고 가격을 초과할 때까지(먼저 도래하는 시점까지) 실행됩니다.

스팟 인스턴스에 대한 자세한 내용은 http://aws.amazon.com/ec2/spot-instances/를 참조하십시오. 로드 테스트에 스팟 인스턴스를 사용하는 사례 연구는 http://aws.amazon.com/solutions/case-studies/browsermob/에서 볼 수 있습니다.

---

[27] 기술 문서 http://aws.amazon.com/articles/1636185810492479 에서 볼 수 있는 "Elastic Load Balancing 평가 모범 사례"를 참조하십시오.
[28] Auto Scaling 문서 참조: http://docs.amazonwebservices.com/AutoScaling/latest/DeveloperGuide/Welcome.html
[29] 블로그 문서 참조: http://aws.typepad.com/aws/2011/07/additional-cloudwatch-metrics-for-amazon-sqs-and-amazon-sns.html
[30] 블로그 문서 참조: http://aws.typepad.com/aws/amazon-simple-workflow-service/
[31] 참조: http://aws.amazon.com/ec2/spot-instances/
[32] 가용 영역은 다른 가용 영역에서 분리되도록 설계된 리전 내의 개별적인 지점으로, 동일 리전의 다른 가용 영역에 대한 저렴하고 지연 시간이 짧은 네트워크 연결을 제공합니다.
[33] 현재 스팟 가격: http://aws.amazon.com/ec2/spot-instances/#6

사용자 승인 테스트

사용자 승인 테스트의 목표는 최종 사용자 기반을 대표하는 테스트 팀에 최신 릴리스를 제공하여 프로젝트 요구 사항 및 사양이 충족되는지 확인하는 것입니다. 사용자가 소프트웨어를 조기에 테스트할 수 있다면 분석 단계 중에 발생하는 개념적 약점을 찾아내거나 프로젝트 요구 사항의 회색 영역을 확인할 수 있습니다. 소프트웨어를 더 자주 테스트하게 되면 사용자는 기능적 구현 오류 및 사용자 인터페이스 또는 애플리케이션 흐름에 대한 오해를 조기에 식별할 수 있어 수정으로 인한 비용과 영향을 낮출 수 있습니다. 사용자 승인 테스트로 감지된 결함은 다른 수단으로는 감지하기가 매우 어려울 수 있습니다. 승인 테스트를 자주 실시할수록, 최종 사용자가 요구 사항의 변화에 따라 개발 팀에 중요한 피드백을 제공할 수 있기 때문에 프로젝트는 향상됩니다.

하지만 다른 테스트 방법과 마찬가지로 승인 테스트가 애플리케이션을 테스트할 환경을 실행하기 위해서는 리소스를 배포해야 합니다. 이전 섹션에서 설명했듯이 AWS 는 비용 효율적인 방식으로 필요할 때 온디맨드 용량을 제공하기 때문에 승인 테스트에도 매우 적절합니다. AWS 는 앞에서 설명한 몇 가지 기술을 사용하여 새로운 테스트 환경을 프로비저닝하고 더 이상 필요하지 않은 환경을 폐기하는 프로세스의 완벽한 자동화를 지원합니다. 테스트 환경은 특정 시간대에만 제공하거나 최신 소스 코드 버전에서 지속적으로 제공하며, 주요 릴리스가 있을 때마다 제공할 수 있습니다.

Amazon VPC 내에 승인 테스트 환경을 배포하면 내부 사용자가 테스트할 애플리케이션에 투명하게 액세스할 수 있습니다. 또한, 그러한 애플리케이션을 LDAP, 이메일 서비스 등 회사 내부의 다른 프로덕션 서비스와 통합할 수도 있어 최종 사용자에게 실제 및 최종 프로덕션 환경에 훨씬 가까운 테스트 환경을 제공합니다.

항목별 테스트

항목별 테스트는 제어 시스템과 테스트 시스템을 비교하는 데 사용하는 방법입니다. 목표는 테스트 시스템에 적용되는 변경 사항이 제어 시스템에 비해 원하는 지표를 향상시켜 주는지 평가하는 것입니다. 이 기술을 사용하여 수많은 여러 파라미터가 전체 효율성에 영향을 미칠 수 있는 복잡한 시스템의 성능을 최적화할 수 있습니다. 어떤 파라미터가 원하는 효과를 낼 것인지 파악하는 것은 수많은 구성 요소를 함께 사용하고 서로 성능에 영향을 미치는 경우에는 특히 분명하지 않을 때가 있습니다. 또한 새로운 알고리즘, 캐시, 다양한 데이터베이스 엔진 또는 타사 소프트웨어 등 프로젝트에 중요한 변경 사항을 적용할 때 이 기술을 사용할 수 있습니다. 그런 경우 목표는 변경 사항이 시스템의 전역 성능에 긍정적인 영향을 주도록 하는 것입니다.

테스트 및 제어 시스템을 배포한 다음에는 로드 테스트 기술 또는 간단한 테스트 입력을 사용하여 동일한 입력을 전송합니다. 마지막으로 두 시스템에서 성능 지표와 로그를 수집 및 비교하여 테스트 시스템에 적용한 변경 사항이 제어 시스템 전반에 기능 향상을 가져왔는지 확인합니다.

온디맨드로 전체 테스트 환경을 프로비저닝하면 항목별 테스트를 효율적으로 수행할 수 있습니다. 환경 프로비저닝을 자동화하지 않고도 항목별 테스트를 할 수 있지만 앞에서 설명한 자동화 기술을 사용하면 테스트가 필요할 때마다 AWS 의 선불형 종량 과금제 모델을 활용하여 더욱 쉽게 테스트를 수행할 수 있습니다. 반대로 기존 하드웨어를 사용하면 여러 프로젝트의 다양한 테스트 환경을 동시에 실행하기가 어려울 수 있습니다.

항목별 테스트는 비용 최적화 관점에서도 유용합니다. 서로 다른 AWS 계정에서 두 가지 환경을 비교함으로써 비용/성능 비율을 쉽게 계산해 두 환경을 비교할 수 있습니다. 비용 성능에 대한 아키텍처 변경 사항을 지속적으로 테스트하면 아키텍처를 최적화하여 효율성을 높일 수 있습니다.

내결함성 테스트

AWS 가 개발한 애플리케이션에 대한 대상 프로덕션 환경인 경우 구체적인 테스트 방법들은 시스템이 구성 요소 장애 등의 코너 사례를 처리하는 방법에 대한 통찰력을 제공합니다. AWS 는 내결함성 시스템 구축을 위한 다양한 옵션을 제공합니다. Amazon S3, Amazon DynamoDB, Amazon SimpleDB, Amazon SQS, Amazon Route 53, Amazon CloudFront 등 일부 서비스는 본질적으로 내결함성이 있습니다. Amazon EC2, Amazon EBS 및 Amazon RDS 등 다른 서비스도 내결함성 및 가용성이 높은 시스템을 설계하는 데 도움이 되는 기능을 제공합니다. 예를 들어 Amazon RDS 는 다른 가용 영역에 있는 복제본을 자동으로 프로비저닝 및 관리하여 데이터베이스의 가용성을 향상시키는 다중 가용 영역[34] 옵션을 제공합니다. 내결함성 아키텍처를 구축하는 방법에 대한 자세한 내용은 백서 "AWS 에 내결함성 애플리케이션 구축"[35] 및 AWS 아키텍처 센터[36]에서 볼 수 있는 리소스를 참조하십시오.

많은 AWS 고객이 AWS 에서 미션 크리티컬 애플리케이션을 실행하고 있는데 이 경우 아키텍처의 내결함성을 확인해야 합니다. 따라서 모든 시스템에 적용되는 중요한 방법은 내결함성 기능을 테스트하는 것입니다. 테스트 시나리오가 시스템 연습을 하는 동안(로드 테스트와 유사한 기술 사용) 일부 구성 요소를 일부러 다운시켜 시스템이 시뮬레이션한 장애에서 복구할 수 있는지 확인합니다. AWS Management Console 또는 명령줄 인터페이스를 사용하여 테스트 환경과 상호 작용할 수 있습니다. 예를 들어 Amazon EC2 인스턴스를 종료하여 Auto Scaling 그룹이 예상된 대로 작동하는지와 대체 인스턴스가 자동으로 프로비저닝되는지 테스트할 수 있습니다. 또한, 이런 종류의 테스트를 자동화할 수 있습니다. 예를 들어 가끔씩 그리고 무작위로 Amazon EC2 인스턴스를 종료하는 자동화 도구를 사용하는 것이 모범 사례입니다.

리소스 관리

AWS 에서는 개발 및 테스트 팀이 자체 리소스를 보유하고 요구 사항에 따라 확장할 수 있습니다. AWS CloudFormation 스택 또는 설명한 다른 자동화 기술 몇 가지를 사용하면 쉽게 다양한 인스턴스로 구성된 복잡한 환경 또는 플랫폼을 프로비저닝할 수 있습니다. 여러 팀으로 구성된 규모가 큰 조직에서는 AWS 에서 실행되는 IT 리소스를 중앙 집중화 및 관리하는 일을 담당하는 내부적 역할 또는 서비스를 만드는 것이 모범 사례입니다. 이 역할은 일반적으로 다음과 같이 구성됩니다.

- 여기에서 설명한 내부 개발 및 테스트 방법 홍보
- 조직에서 사용하는 다양한 도구 및 플랫폼으로 템플릿 AMI 및 템플릿 AWS CloudFormation 스택 개발 및 유지 관리
- 네트워크 구성(예: Amazon VPC), 보안 구성(예: 보안 그룹 및 IAM 자격 증명) 등 조직의 정책에 따라 프로젝트 팀에서 리소스 요청을 수집하고 AWS 에서 리소스를 프로비저닝
- Amazon CloudWatch 를 사용하여 리소스 사용량, 요금 모니터링 및 팀 예산에 할당

AWS Management Console 을 사용하여 위와 같은 작업을 수행할 수 있지만 내부 프로세스와 더욱 긴밀히 통합하기 위해 자체의 내부 프로비저닝 및 관리 포털을 개발하는 것이 좋습니다. 프로그래밍 방식으로 AWS 에서 실행되는 리소스에 액세스할 수 있는 AWS SDK 중 하나를 사용하면 이 작업을 수행할 수 있습니다.

---

[34] 참조: http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/Concepts.DBInstance.html#Concepts.MultiAZ
[35] 자료 참조: http://d36cz9buwru1tt.cloudfront.net/AWS_Building_Fault_Tolerant_Applications.pdf
[36] 참조: http://aws.amazon.com/architecture/

비용 할당 및 여러 AWS 계정

개발 및 테스트 활동용으로 특정 계정을 만드는 것이 유용하다고 생각하는 고객이 일부 있습니다. 프로덕션 환경을 AWS 에서도 실행하고 팀과 책임을 분리해야 하는 경우 중요한 사항이 될 수 있습니다. 분리된 계정은 기본적으로 서로 격리되어 있으므로 예를 들면 개발 사용자와 테스트 사용자가 프로덕션 리소스에 방해를 주지 않을 수 있습니다. 공동 작업을 지원하기 위해 AWS 는 Amazon S3 객체, AMI 및 Amazon EBS 스냅샷 등 여러 계정에 대한 리소스 공유를 지원하는 여러 가지 기능을 제공합니다.

개발 및 테스트 주기의 다양한 활동 및 단계를 분리하고 비용을 할당할 수 있도록 AWS 는 다양한 옵션을 제공합니다. 한 가지 옵션은 분리된 계정을 사용하고(예: 개발, 테스트, 스테이징 및 프로덕션 별도 계정) 계정별로 청구서를 받는 것입니다. 또한 결제 간소화 등의 목적으로 여러 계정을 통합할 수도 있습니다. 또 한 가지 옵션은 비용 할당 보고서를 사용하는 방법인데, 이 방법은 리소스 태그 지정을 사용하여 AWS 비용을 정리 및 추적할 수 있습니다. 개발 및 테스트의 맥락에서 태그는 개발 주기의 다양한 단계 또는 팀을 대표할 수 있으며 가장 유용한 차원을 자유롭게 선택할 수 있습니다.

결론

개발 및 테스트 방법은 개발 주기의 특정 시점에 특정 리소스를 필요로 합니다. 기존 환경에서는 그러한 리소스를 전혀 사용할 수 없거나 필요한 일정 내에 사용할 수 없습니다. 그러한 리소스가 사용 가능할 때 고정된 양의 용량을 제공하게 되는데 테스트와 같은 다양한 활동에서 특히 부족해지고 리소스를 사용하지 않을 때는 낭비됩니다(요금은 지불)[37].

Amazon Web Services 는 기존 개발 및 테스트 인프라를 대체할 비용 효율적 방법을 제공합니다. 하드웨어를 몇 주, 심지어 몇 개월 기다리는 대신 즉시 필요한 리소스를 프로비저닝하고 워크로드 증가에 따라 즉시 확장하여, 더 이상 필요하지 않게 되면 리소스를 해제할 수 있습니다. 개발 및 테스트 환경이 몇 개로 구성되거나 몇백 개의 인스턴스로 구성되어도, 몇 시간 또는 연중무휴로, 사용한 양만큼 요금을 지불할 수 있습니다. AWS 는 프로그래밍 언어 및 운영 체제와 무관한 플랫폼이며 기업에서 사용하는 개발 플랫폼 또는 프로그래밍 모델을 선택할 수 있습니다. 이러한 유연성 덕분에 인프라 운영 및 유지 관리가 아니라 프로젝트에 집중할 수 있습니다.

또한, AWS 는 기존 하드웨어로는 실현하기 어려웠던 가능성을 지원합니다. AWS 에서는 리소스 전체를 자동화하여 사용자의 개입 없이 환경을 프로비저닝 및 폐기할 수 있습니다. 온디맨드로 개발 환경을 시작하고 필요할 때 빌드를 시작하며, 리소스 가용성의 제약을 받지 않고 테스트 리소스를 프로비저닝하여 테스트 실행 또는 캠페인 전체를 자동으로 조율할 수 있습니다.

AWS 는 빠른 속도로 변경되는 인프라에 실험 및 반복 기능을 제공합니다. 사용자의 프로젝트 팀은 선행 비용 또는 장기적인 약정 없이 자유롭게 저렴한 용량을 사용하여 모든 종류의 테스트를 수행하거나 새로운 아이디어를 실험할 수 있습니다. 따라서 AWS 는 개발 및 테스트 팀에 적합한 플랫폼입니다.

[37] 참조: http://aws.amazon.com/economics/

참고 문헌

- 백서: “AWS 운영 체크리스트” http://media.amazonwebservices.com/AWS_Operational_Checklists.pdf
- 백서: “AWS 요금 책정 방식” http://media.amazonwebservices.com/AWS_Pricing_Overview.pdf
- AWS 아키텍처 센터 - http://aws.amazon.com/architecture
- AWS 기술 백서 - http://aws.amazon.com/whitepapers/