# AWS의 SaaS 솔루션

## 테넌트 격리 아키텍처

*2016년 1월 일*

# 고지 사항

이 문서는 정보 제공 목적으로만 제공됩니다. 본 문서의 발행일 당시 AWS의 현재 제품 및 실행방법을 설명하며, 예고 없이 변경될 수 있습니다. 고객은 본 문서에 포함된 정보나 AWS 제품 또는 서비스의 사용을 독립적으로 평가할 책임이 있으며, 각 정보 및 제품은 명시적이든 묵시적이든 어떠한 종류의 보증 없이 "있는 그대로" 제공됩니다. 본 문서는 AWS, 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 보증, 표현, 계약 약속, 조건 또는 보증을 구성하지 않습니다. 고객에 대한 AWS의 책임 및 의무는 AWS 계약에 준거합니다. 본 문서는 AWS와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

# 목차

# 요약

점점 더 많은 엔터프라이즈 솔루션이 SaaS(Software as a Service) 모델로 제공되고 있지만, SaaS 솔루션 설계에는 어려움이 따릅니다. AWS에서 SaaS 솔루션을 배포할 때 고려해야 할 여러 측면과 다양한 옵션이 있습니다. 이 백서는 여러 SaaS 배포 모델을 다루고, 확장성, 가용성, 보안, 성능 및 비용 효율성이 뛰어난 SaaS 서비스를 실현하는 데 사용할 수 있는 AWS 서비스와 AWS 파트너 네트워크(APN) 파트너 솔루션의 조합을 다룹니다.

이제 AWS는 AWS에서 SaaS 솔루션을 구축, 시작 및 확장하는 데 도움이 되는 체계적인 AWS SaaS 파트너 프로그램을 제공합니다. 고객의 비즈니스가 성장함에 따라 고객이 필요로 하는 비즈니스 및 기술 지원을 제공하기 위해 AWS도 함께 합니다. 자세한 내용은 SaaS 파트너 프로그램 웹 사이트 를 확인하십시오.

# 서론

SaaS 모델에 배포할 수 있는 다양한 솔루션이 있으며 이러한 솔루션은 여러 유사점과 공통적인 패턴을 공유합니다. 이 문서에서는 다음 주제에 대해 살펴보겠습니다.

- 공통적인 솔루션 구성 요소 – 결제, 모니터링 및 분석과 같은 기능 솔루션은 핵심 구성 요소와는 별도로 처리하는 것이 좋습니다. 이 구성 요소에 대해 자세히 살펴보겠습니다.

- SaaS 솔루션 - 테넌트 격리 아키텍처 패턴 – AWS에서 다양한 방식으로 배포할 수 있는 솔루션입니다. 다중 테넌트 SaaS 배포와 관련한 요구 사항을 지원하는 일반적인 모델과 각 경우에 대한 고려 사항에 대해 살펴보겠습니다.

이 백서는 SaaS 배포의 기술 및 아키텍처 측면에 중점을 두고 있으며 소프트웨어 공급업체 라이선스, SLA, 가격 모델 및 DevOps 사례 고려 사항과 같은 비즈니스 및 프로세스 관련 측면에 대해서는 다루지 않으려고 합니다.

# 공통적인 솔루션 구성 요소

SaaS 솔루션의 핵심 기능 구성 요소를 구축하는 것 외에도 솔루션의 미래를 대비하고 이를 더 쉽게 관리하는 데 도움이 되는 추가 지원 구성 요소를 구축하는 것이 좋습니다. 추가 지원 구성 요소를 구축하면 시간이 지나면서 테넌트를 쉽게 확장하고 추가할 수 있습니다. 다음 섹션에서는 SaaS 솔루션 설정에 대해 권장되는 지원 구성 요소 중 일부에 대해 설명합니다.

## 보안 및 네트워킹(테넌트 격리 모델링)

다중 테넌트 시스템 설계의 첫 번째 단계는 테넌트를 안전하게 유지하고 서로 격리하는 전략을 정의하는 것입니다. 여기에는 네트워크/스토리지 계층에서의 분리 정의, 저장 데이터 또는 전송 중인 데이터 암호화, 키 및 인증서 관리, 애플리케이션 수준 보안 구성 요소 관리와 같은 보안 고려 사항이 포함될 수 있습니다. AWS CloudHSM, AWS CloudTrail, Amazon VPC, AWS WAF, Amazon Inspector, Amazon CloudWatch 및 Amazon CloudWatch Logs 등 각 수준에서 보안 고려 사항을 해결하는 데 도움이 되는 다양한 AWS 서비스가 있습니다. 이와 같은 기본 AWS 서비스를 사용하여 솔루션의 보안 및 네트워킹 요구 사항과 일치하는 모델을 정의할 수 있습니다. AWS 기본 서비스 외에도 많은 고객들은 인프라 보안 공간에서 APN 파트너 서비스를 사용하여 보안 상태를 강화하고 침입 탐지 시스템(IDS)/침입 방지 시스템(IPS) 같은 기능을 추가합니다.

## 자격 증명 관리, 사용자 인증 및 권한 부여

AWS 서비스와 SaaS 애플리케이션을 모두 관리하려면 사용자 인증 및 권한 부여 전략을 결정하는 것이 중요합니다. AWS 서비스의 경우 AWS Identity and Access Management(IAM) 사용자, IAM 역할, Amazon Elastic Compute Cloud(Amazon EC2) 역할, 소셜 자격 증명, 디렉터리/LDAP 사용자 및 SAML 기반 통합을 사용하는 연동 자격 증명을 사용할 수 있습니다. 이와 마찬가지로 애플리케이션의 경우 사용자를 인증할 수 있는 다양한 방법이 있습니다. 애플리케이션 인증 요구 사항을 지원하는 계층을 구축하는 것이 좋습니다. 모바일 사용자에 대해 Amazon Cognito 기반 인증을 고려할 수 있으며 서로 다른 자격 증명 공급자에서 인증을 관리하기 위해 자격 증명 및 액세스 제어 부분의 APN 파트너 서비스를 고려할 수도 있습니다.

## 모니터링, 로깅 및 애플리케이션 성능 관리

문제를 진단하는 데 도움이 될 뿐만 아니라 향후 문제를 방지하는 사전 조치를 사용할 수 있도록 여러 계층에서 모니터링을 활성화해야 합니다. Amazon CloudWatch의 데이터를 활용하는 이점을 얻을 수 있습니다. 이는 중요한 인프라에 대한 세부 모니터링을 가능하게 하며, 모든 문제를 알려주도록 경보를 구성하게 합니다. AWS 리소스 인벤토리, 구성 히스토리와 구성 변경이 발생할 때 알림을 받을 수 있는 보안 및 관리를 가능하게 하는 AWS Config 를 사용할 수도 있습니다. 애플리케이션 수준 모니터링의 경우 Amazon CloudWatch Logs 기능을 사용하여 로그를 서비스에 실시간으로 스트리밍할 수 있습니다. 또한 패턴을 검색할 수 있으며 Amazon CloudWatch를 구성하여 애플리케이션 로그에서 발생하는 오류 수를 추적하고 오류 비율이 사용자가 지정한 임계값을 초과할 때마다 알림을 보내게 할 수도 있습니다. 또한 많은 기업은 로깅 및 모니터링 부분에서 APN 파트너 서비스를 사용하여 애플리케이션 성능 측면을 모니터링합니다.

## 분석

대부분의 SaaS 솔루션에는 애플리케이션 로그, 사용자 액세스 로그 및 결제 관련 데이터를 비롯한 수많은 원시 데이터가 포함되어 있으며 일반적으로 이러한 데이터를 적절히 분석할 때 많은 통찰력이 얻을 수 있습니다. 일괄 처리 중심 분석 외에도 실시간 분석을 수행하여 플랫폼의 다양한 테넌트가 어떤 종류의 작업을 호출하고 있는지 확인하거나 실시간 인프라 관련 측정치를 보고 예기치 않은 동작을 감지하며 잠재적인 문제를 사전에 방지할 수 있습니다. Amazon Elastic MapReduce(Amazon EMR), Amazon Redshift, Amazon Kinesis, Amazon Machine Learning, Amazon QuickSight, Amazon Simple Storage Service (Amazon S3) 및 Amazon EC2 스팟 인스턴스 같은 AWS 서비스를 사용하여 이러한 유형의 기능을 구축할 수 있습니다. 초기 단계에서 분석은 일반적으로 플랫폼의 부수적인 기능이지만 여러 테넌트가 SaaS 플랫폼에 온보딩되는 즉시 사용 패턴을 감지하고 파악하여 권장 사항을 제공하고 결정을 내리게 이끄는 핵심 기능으로 변모합니다. 솔루션 개발 주기 초기 단계에 이 계층을 계획하는 것이 좋습니다. 그림 1은 데이터 수집부터 스토리지, 데이터 분석/처리에 이르는 일부 AWS 빅 데이터 서비스 및 기능을 보여 줍니다.

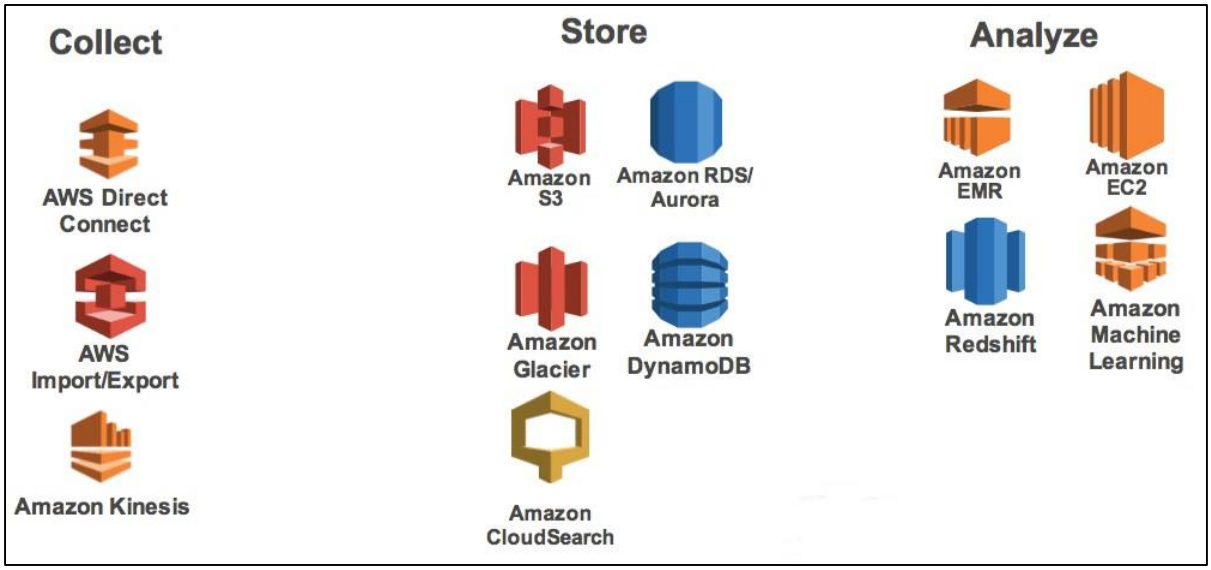


그림 1: AWS 빅 데이터 및 분석 서비스

## 구성 관리 및 프로비저닝

AWS는 솔루션 배포 자동화에 대한 다양한 가능성을 제공합니다. Amazon 머신 이미지(AMI) 자체에서 일부 배포 작업을 수행할 수 있는 기능이 있으며 다음과 같은 다양한 다른 방법을 사용하여 더욱 구성 가능하거나 빈번한 변경 사항을 자동화할 수 있습니다.

- 운영 체제 보안 강화 또는 애플리케이션 재인증이 필요없는 런타임 환경의 특정 버전 설정(Java 업그레이드 등) 등 일회성 작업 또는 시간 소모적인 설치(미들웨어/데이터베이스 설정 등)는 AMI 자체에 적용할 수 있습니다.
- 코드 리포지토리에서의 코드 업데이트, 부팅 시간 작업(도메인/클러스터 조인 등) 및 특정 환경별 구성(개발/테스트/생산 등)처럼 배포에 대해 더욱 빈번하게 변화하는 측면을 처리하려면 EC2 인스턴스의 사용자 데이터 섹션 또는 AWS CodeCommit, AWS CodePipeline 및 AWS CodeDeploy 같은 AWS 서비스의 사용자 정의 스크립트를 사용할 수 있습니다.

- 완전한 스택 실행의 경우 AWS CloudFormation을 사용하여 높은 수준의 자동화를 달성할 수 있습니다. 이는 개발자 및 시스템 관리자가 관련 AWS 리소스 모음을 쉽게 만들고 관리할 수 있으며 이러한 리소스를 질서 정연하고 예측 가능한 방식으로 프로비저닝 및 업데이트할 수 있게 해 줍니다. 요구 사항에 따라 AWS Elastic Beanstalk 및 AWS OpsWorks 도 빠른 배포와 자동화를 지원할 수 있습니다.

서로 다른 유형의 작업에서 분리 작업을 적절히 조합하여 빠른 부팅 시간(자동 확장된 계층에 종종 필요함) 및 구성 가능한 자동 설정(유연한 배포에 필요함) 사이에 올바른 균형을 유지할 수 있습니다.

## 스토리지, 백업 및 복원 기능

대부분의 AWS 서비스에는 백업을 수행할 수 있는 메커니즘이 있으므로 최근 변경 사항을 취소해야 하는 경우 마지막으로 알려진 안정된 상태로 되돌릴 수 있습니다. Amazon EC2 AMI 생성 또는 스냅샷 생성(Amazon EBS, Amazon RDS 및 Amazon Redshift 스냅샷)을 포함한 기능이 대부분의 백업 요구 사항을 잠재적으로 지원할 수 있습니다. 하지만 파일 시스템을 중단한 다음 활성 데이터베이스의 스냅샷을 일관성 있게 생성해야 하는 요구 사항처럼 어려운 요구 사항의 경우 타사 백업 도구를 사용할 수 있습니다. 이 중 많은 도구를 AWS Marketplace에서 사용할 수 있습니다.

## AWS 태깅 전략

고유의 메타데이터를 태그의 형태로 각 리소스에 배정하면 인스턴스, 이미지 및 기타 Amazon EC2 리소스를 쉽게 관리할 수 있습니다. SaaS 솔루션을 진행하기 전에 태깅 전략을 도입하는 것이 좋습니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 또한 단일 리소스에 여러 개의 태그를 가질 수도 있습니다. 태그에는 다음 두 가지 사용 목적이 있습니다.

1. **일반적인 리소스 관리:** 태그를 사용하면 용도, 소유자 또는 환경을 기준으로 하는 등 AWS 리소스를 다양한 방식으로 분류할 수 있습니다. 이를 통해 서로 다른 리소스에서 필터링 및 검색을 단순화할 수 있습니다. 또한 리소스 그룹을 사용하면 프로젝트 및 사용하는 리소스를 기반으로 필요한 정보를 구성하고 통합하는 사용자 지정 콘솔을 만들 수 있습니다. 그림 2에 나와 있는 바와 같이 리소스 그룹을 만들어 같은 화면에서 다른 리전의 리소스를 볼 수도 있습니다.

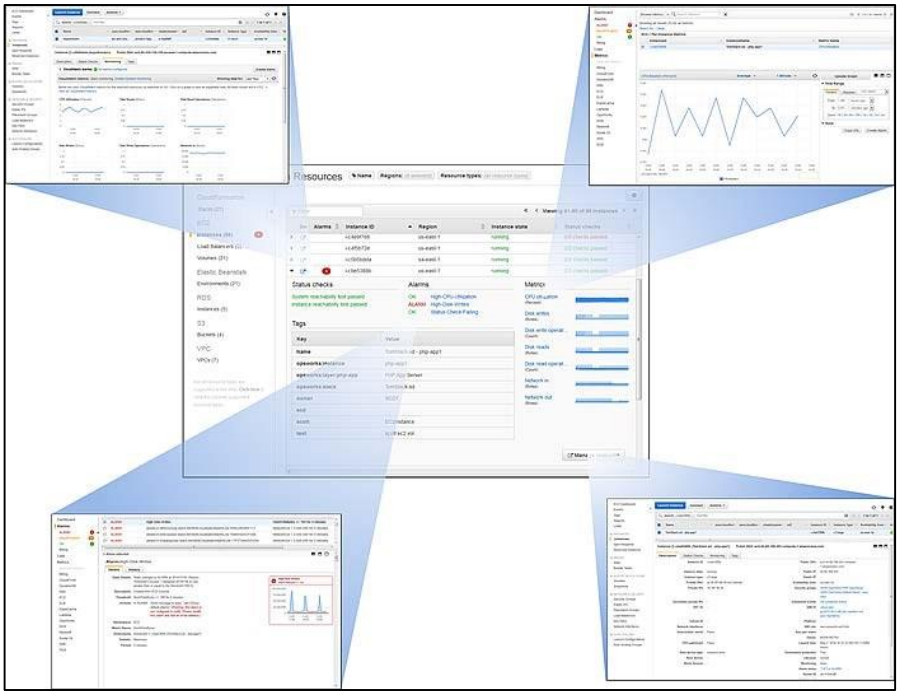

그림 2: AWS 리소스 그룹

2. **결제 분리:** 태그를 통해 비용 할당 보고서를 사용할 수 있으며 사용되는 태깅 전략에 따라 특정 비즈니스 부문 또는 환경을 바탕으로 비용을 분리할 수 있습니다. 이와 더불어 AWS 비용 탐색기는 결제 데이터 관련 가시성 및 보고서를 크게 단순화할 수 있습니다.

## 결제 처리 모듈

다중 테넌트 시스템의 또 다른 중요한 측면은 테넌트의 사용량을 바탕으로 한 비용 분리입니다. AWS 리소스의 관점에서 태깅은 거시적 수준에서 사용량을 분리하는 데 도움이 되는 훌륭한 리소스가 될 수 있습니다. 그러나 대부분의 SaaS 솔루션의 경우 사용량 모니터링에 더 높은 제어 수준이 요구되므로 필요에 따라 자체 사용자 지정 결제 모듈을 구축하는 것이 좋습니다.

결제 모듈은 그림 3에 표시된 상위 수준의 일반적인 예처럼 보일 수 있습니다.

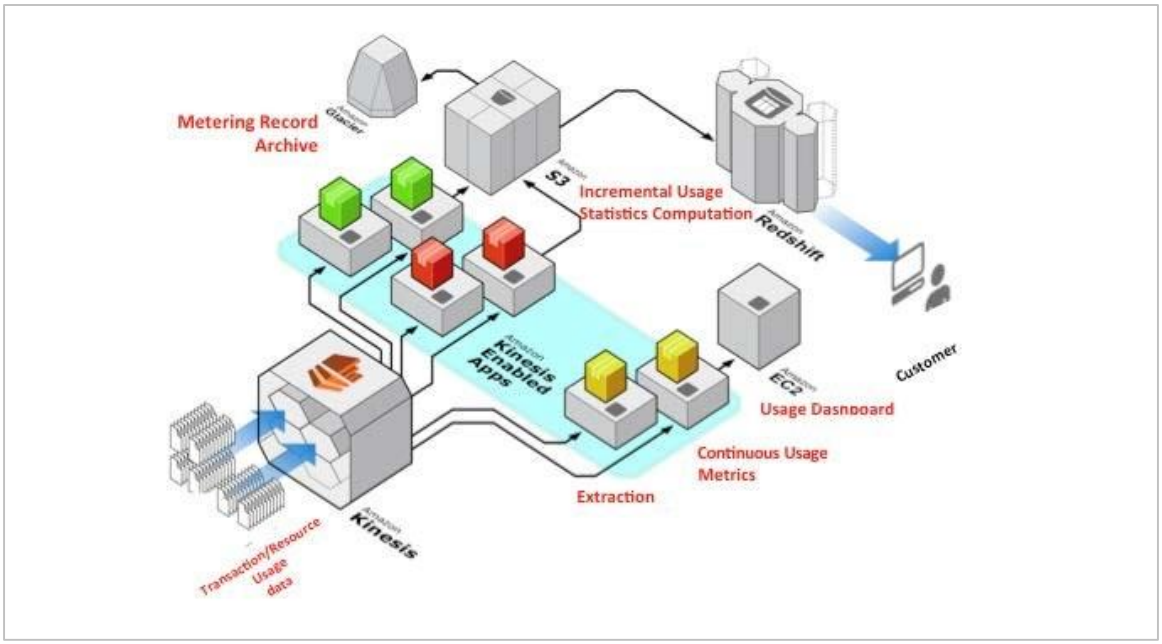


그림 3: 샘플 측정 및 결제 처리 모듈

- 시작, 중지 및 종료된 모든 리소스를 추적하고 데이터는 Amazon Kinesis 스트림으로 전송됩니다.
- API 요청 수 또는 요청을 처리하는 데 소요되는 시간 같은 세분화된 측정값을 추적하고 실시간으로 Kinesis 스트림에 데이터를 공급합니다.
- Amazon Kinesis에 저장된 데이터를 처리할 수 있는 소비자(consumer) 애플리케이션에는 다음과 같은 두 가지 유형이 있습니다.
- 다양한 테넌트가 시스템을 어떻게 활용하고 있는지 실시간 측정치를 생성하는 소비자군. 이는 특정 테넌트의 사용량을 줄이거나 실시간 피드를 기반으로 다른 교정 조치를 수행하는 여부와 같은 결정을 내리는 데 도움이 될 수 있습니다.
- Kinesis 소비자(consumer)군의 두 번째 세트는 연속 피드를 집계하고 결제에 대한 월별 또는 분기별 사용 보고서를 생성할 수 있습니다. 또한 원시 데이터를 처리하고 Amazon Redshift에 저장하여 각 테넌트에 대한 사용량 분석을 제공할 수도 있습니다. 과거 데이터 처리 또는 변환의 경우 Amazon EMR을 사용할 수 있습니다.

# SaaS 솔루션 - 테넌트 격리 아키텍처 패턴

완전 격리된 배포부터 완전 공유된 SaaS 유형 아키텍처에 이르기까지 서로 간에 여러 가지 다른 배포 옵션을 포함하는 AWS의 패키지 솔루션을 배포하는 것에는 다양한 방법이 있습니다. 배포 옵션 중 하나를 지원하려면 솔루션 또는 애플리케이션 자체가 SaaS 다중 테넌트 모델을 지원할 수 있어야 합니다. 이 모델은 서로 다른 배포 모델의 AWS 고유 구성 요소에 대해 자세히 살펴보기 전에 기본적으로 가정하는 모델입니다.

특정 AWS 배포 모델을 선택하는 데 대한 결정은 다음과 같이 여러 기준에 따라 달라집니다.

- 테넌트 및 배포 간 분리 수준
- 테넌트별 스택에 대한 애플리케이션 확장성 측면
- 테넌트별 애플리케이션 사용자 지정 수준
- 배포 비용
- 운영 및 관리 노력
- 최종 테넌트 측정 및 결제 측면

잠재적인 예기치 못한 방식으로 서로 영향을 주는 "루빅 큐브" 옵션도 다른 선택이 됩니다. 이 문서의 목표는 이러한 예기치 못한 다차원적 영향에 대해 지원하는 것입니다. 다음 섹션에서는 AWS의 일부 SaaS 배포 모델에 대해 설명하고 각 옵션에 대한 장단점을 포함하여 비즈니스 및 기술 요구 사항에 따라 최적의 솔루션을 얻을 수 있게 도와줍니다.

## 모델 #1 - AWS 계정 계층의 테넌트 격리

이 모델에서는 모든 테넌트가 개별 AWS 계정을 보유하게 되며 어느 정도로 격리됩니다. 본질적으로 이는 진정한 다중 테넌트 SaaS 솔루션은 아니지만 AWS의 관리형 솔루션으로 처리될 수 있습니다.

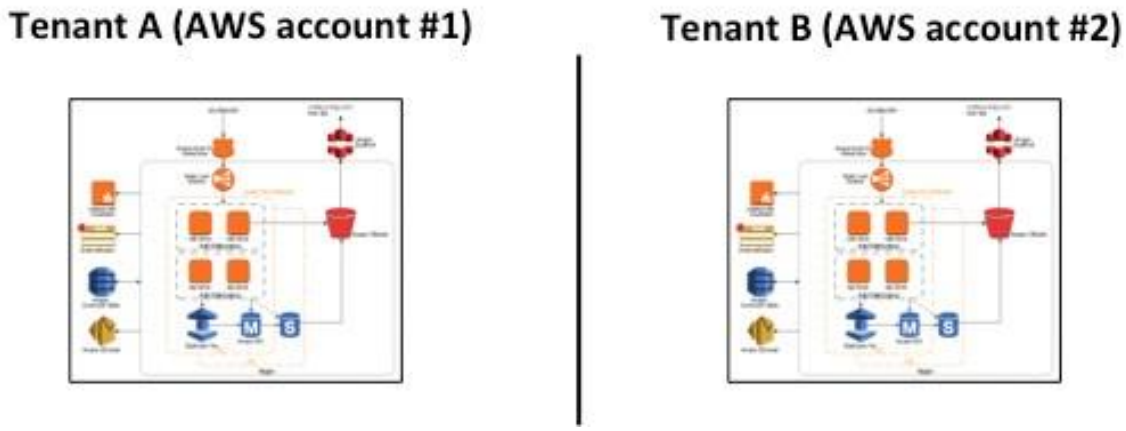


그림 4: AWS 계정 계층의 테넌트 격리

### 장점:

- 테넌트는 완전히 분리되어 있으며, 중첩되지 않은 상태이므로 각 테넌트에게 더 큰 보안성을 제공할 수 있습니다.
- 모든 배포가 테넌트(또는 조직)에 특정하기 때문에 솔루션 또는 일반 구성 사용자 지정이 쉽습니다.
- 각 테넌트(또는 조직)에 대해 별도의 월별 청구서가 생성되기 때문에 AWS 사용량을 쉽게 추적할 수 있습니다.

### 단점:

- 이 옵션에는 다중 테넌트 SaaS 모델이 제공하는 규모의 경제로 실현할 수 있는 리소스 및 비용 최적화 효과가 부족합니다.
- 많은 수의 테넌트가 있으면 운영 관점에서 별도의 AWS 계정 및 개별 테넌트 배포를 관리하는 것이 어려워질 수 있습니다.
- 모든 AWS 계정 루트 로그인에 멀티 팩터 인증(MFA)을 사용하도록 하는 것이 가장 좋습니다. 점점 증가하는 개별 테넌트 계정으로 인해 모든 MFA 장치를 관리하기가 어려워집니다.

## 모범 사례:

- 중앙 집중식 운영 및 관리 - IAM은 IAM 역할을 사용한 AWS 계정 간 액세스 권한 위임을 지원합니다. 이 기능을 사용하면 각 AWS 계정에 개별적으로 로그인할 필요 없이 다양한 작업(AWS CloudFormation을 사용하여 새 스택 시작 또는 보안 그룹 구성 업데이트 등)을 수행하는 역할을 가정하여 자체 공통 AWS 계정을 통해 모든 테넌트 AWS 계정을 관리할 수 있습니다. AWS Management Console, AWS 명령줄 인터페이스(AWS CLI), 및 API를 사용하여 이러한 기능을 활용할 수 있습니다. 그림 3은 AWS Management Console에서 이렇게 설정하는 방법의 스냅샷을 보여 줍니다.

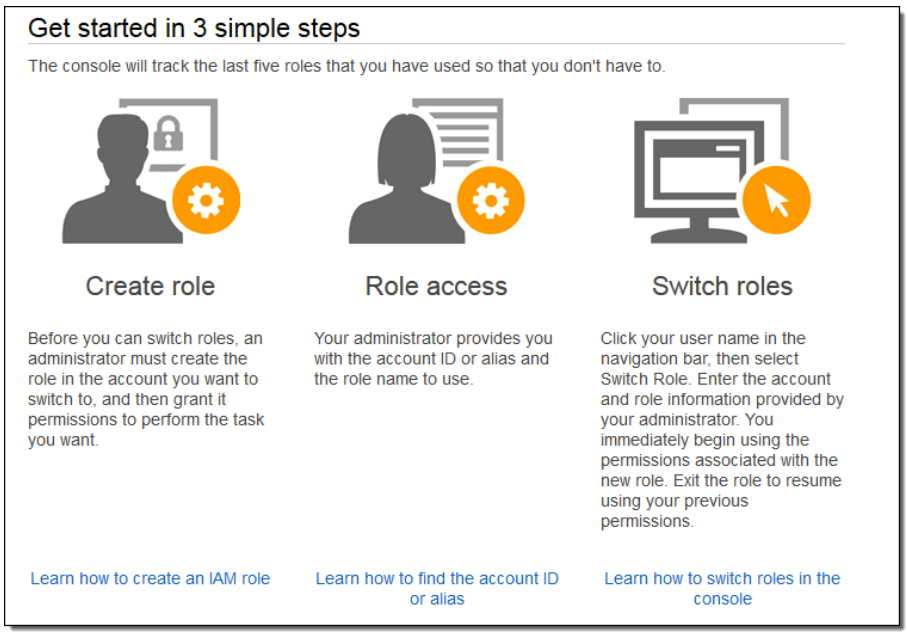

그림 5: 교차 계정, IAM 역할 기반 설정

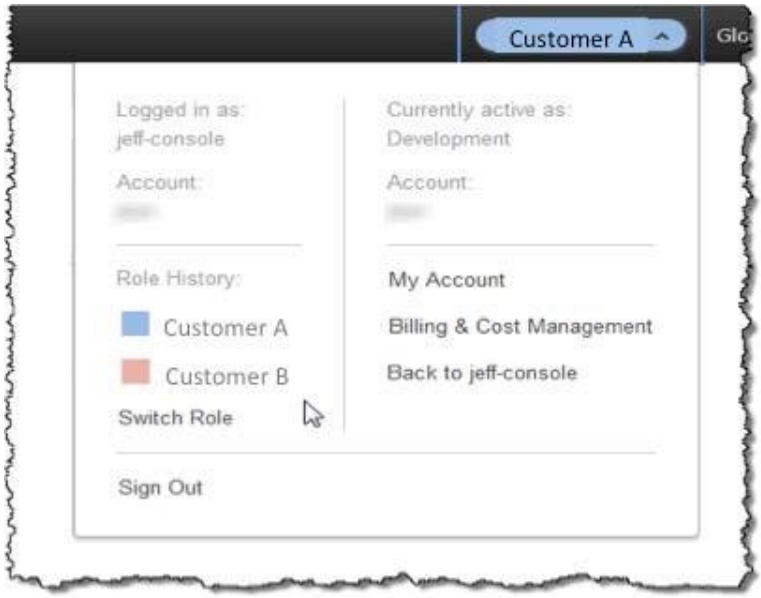


그림 6: 교차 계정, IAM 역할 기반 전환

- 통합 AWS 결제 - 통합 결제 기능을 사용하여 조직의 여러 AWS 계정 중에서 하나를 지급(payer) 계정으로 지정함으로써 결제를 통합할 수 있습니다. 통합 결제를 사용하면 모든 계정에서 발생한 AWS 요금뿐만 아니라 지급인 계정에 연결된 각각의 개별 AWS 계정에 대한 자세한 비용 보고서도 볼 수 있습니다.

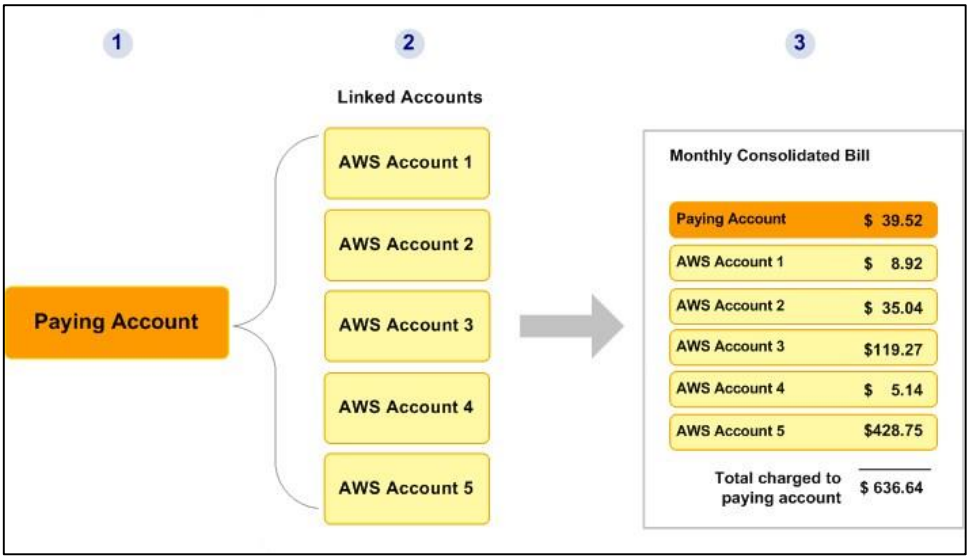


그림 7: AWS 통합 결제

- VPC 피어링 - 일련의 중앙 서비스(백업, 안티바이러스, OS 패치 등)를 이용하려는 경우 동일한 AWS 리전에서 이러한 공유 서비스 및 각 테넌트의 AWS 계정이 있는 공통 AWS 계정 간에 VPC 피어링 연결을 사용할 수 있습니다. 하지만 VPC 피어링 연결 내에서의 데이터 전송에 대해서는 가용 영역을 통한 데이터 전송에 대해 청구되는 것과 같은 요율로 요금이 청구됩니다. 따라서 솔루션의 전체 비용 모델링 연습에서 이 비용을 고려해야 합니다.

## 모델 #2 - Amazon VPC 계층의 테넌트 격리

이 모델에서는 모든 테넌트 솔루션 배포가 동일한 AWS 계정에 위치하지만 분리 수준은 VPC 계층을 따릅니다. 모든 테넌트 배포에 대해 테넌트 간의 논리적 분리를 제공하는 별도의 VPC가 있습니다.

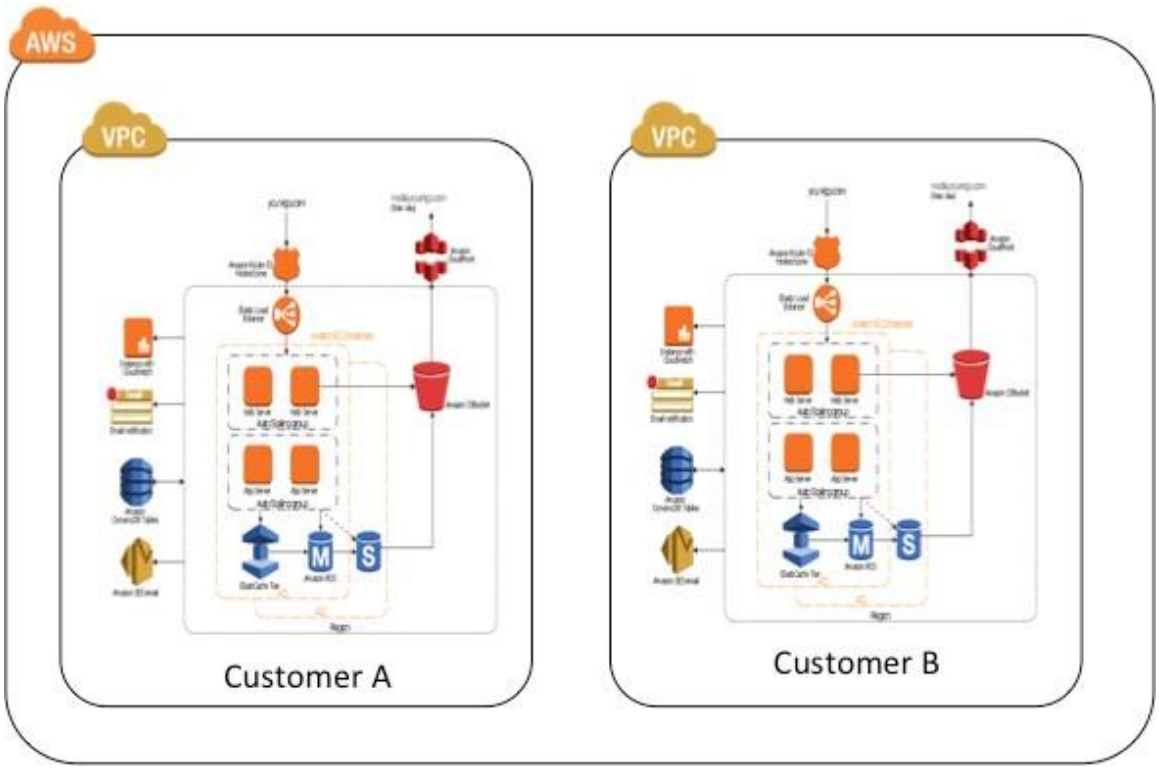


그림 8: VPC 계층의 테넌트 격리

### 장점:

- 모두 단일 계정에 위치하므로 이 모델은 다중 계정 설정보다 더 관리하기 쉽습니다.
- 각각 다른 VPC에 위치하기 때문에 서로 다른 테넌트 간에 적절한 격리가 이루어집니다.
- 이전 모델과 비교할 때 동일한 AWS 계정에 모든 예약 및 대량 구매 요금 구성 요소가 적용될 수 있기 때문에 이 모델은 Amazon EC2 예약 인스턴스의 규모에 따른 이점 및 더 높은 사용성을 제공합니다. 하지만 통합 결제를 사용하는 경우, 통합 결제는 통합 청구서의 모든 계정을 하나의 계정으로 취급하므로 이 모델은 이전 모델에 비해 이점을 제공하지는 않습니다.

### 단점:

- Amazon VPC 관련 제한에 대해 전체적인 계정 측면과 각 테넌트의 VPC 측면에서 면밀히 모니터링해야 합니다.
- 모든 VPC가 온프레미스 설정으로 다시 연결되어야 하는 경우 개별 VPN 연결을 관리하는 것이 어려워질 수 있습니다.
- 동일한 계정일지라도 공유 서비스 집합(백업, 안티바이러스 업데이트, OS 업데이트 등)을 제공해야 하는 경우 VPC 피어링은 공유 서비스 VPC에서 모든 테넌트 VPC로 설정되어야 합니다.
- 보안 그룹은 VPC에 연결되므로 배포 아키텍처에 따라 각 VPC에 대해 여러 보안 그룹을 만들고 관리해야 할 수 있습니다.
- AWS는 Amazon EC2 설명서에 나온 태깅을 지원합니다. 하지만 서비스 및 리소스의 사용량 및 비용을 태깅에 대해 지원받을 수 있는 수준 이상으로 분리해야 하는 경우 사용자 지정 결제 처리 계층을 구축하거나 개별 테넌트 사용량을 명확하게 구분하는 데 도움이 되는 별도의 AWS 계정 전략을 가져야 합니다.

### 모범 사례:

이 설정에서는 태그를 사용하여 각 테넌트 배포에 대한 AWS 비용을 분리합니다. 태그를 개별 리소스 수준에서 관리하는 대신 리소스 그룹 을 정의하고 여기에서 태그를 관리할 수 있습니다. 태깅 전략을 정의한 후에는 월별 비용 할당 보고서 를 사용하여 AWS 비용을 태그별로 나눠서 보고 필요에 따라 구분할 수 있습니다(그림 9의 샘플 보고서 참조).

| Total Cost | user:Owner | user:Stack | user:Cost Center | user:Application |
|---|---|---|---|---|
| 0.95 | DbAdmin | Test | 80432 | Widget2 |
| 0.01 | DbAdmin | Test | 80432 | Widget2 |
| 3.84 | DbAdmin | Prod | 80432 | Widget2 |
| 6.00 | DbAdmin | Test | 78925 | Widget1 |
| 234.63 | SysEng | Prod | 78925 | Widget1 |
| 0.73 | DbAdmin | Test | 78925 | Widget1 |
| 0.00 | DbAdmin | Prod | 80432 | Portal |
| 2.47 | DbAdmin | Prod | 78925 | Portal |

그림 9: 샘플 비용 할당 보고서

## 모델 #3 - Amazon VPC 서브넷 계층의 테넌트 격리

이 모델에서는 모든 AWS 계정과 모든 테넌트 배포에 대한 단일 VPC가 있는 경우에 대해 살펴봅니다. 격리는 서브넷 수준에서 이루어지며 각 테넌트는 테넌트 간에 공유되지 않는 애플리케이션 또는 솔루션의 개별 버전을 가집니다. 그림 10은 이러한 배포 유형을 보여 줍니다.

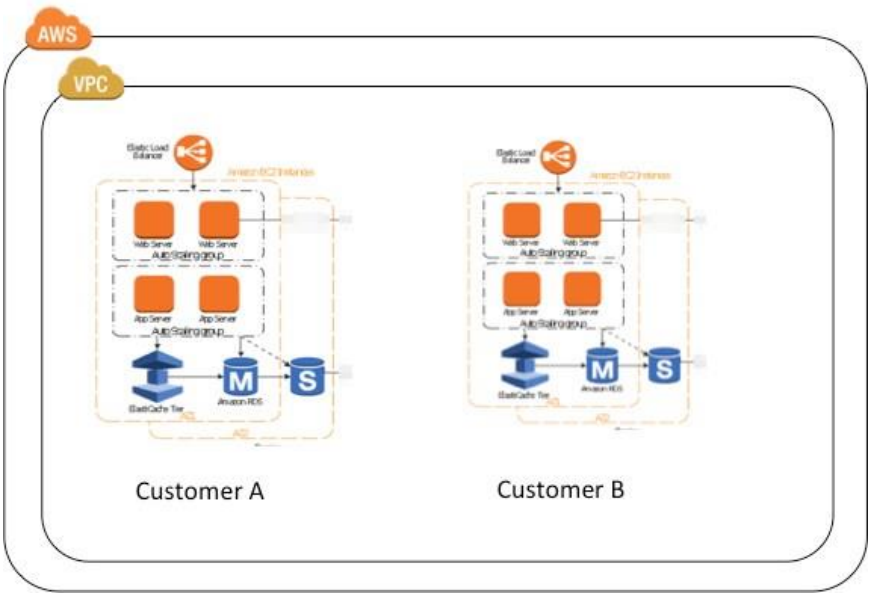


그림 10: VPC 서브넷 계정 계층의 테넌트 격리

### 장점:

- 상호 통신을 위해 VPC 피어링을 설정할 필요가 없습니다.
- 단일 온프레미스 사이트에 대한 VPN 및 AWS Direct Connect 연결은 단일 VPC에서와 같이 간단합니다.

### 단점:

- 테넌트 간의 격리는 서브넷 수준에서 관리해야 하므로 Amazon VPC 네트워크 액세스 통제 목록(NACL) 및 보안 그룹을 신중하게 관리해야 합니다.
- VPC 제한은 테넌트 수가 증가하면서 관리하기가 더 어려워집니다. 또한 크기에 따라 VPC CIDR(클래스 없는 도메인 간 라우팅)에서 서브넷 일부만 프로비저닝할 수 있으며 생성된 후에는 CIDR의 크기를 조정할 수 없습니다.
- VPC 수준 설정(DHCP 옵션 세트)을 변경하는 것은 각각 개별 배포가 있음에도 모든 테넌트에게 영향을 미칩니다.
- VPC 수준에서는 보안 그룹 수와 보안 그룹당 규칙 수에 제한이 있으므로 동일한 VPC에서 다중 테넌트로 제한을 관리하는 것은 복잡한 일일 수 있습니다.

### 모범 사례:

- 퍼블릭 AWS 서비스 엔드포인트(Amazon S3 등)에 액세스하려면 VPC 엔드포인트를 활용하십시오. 네트워크 주소 변환(NAT) 인스턴스를 통해 다중 테넌트의 트래픽을 라우팅하는 것보다 더욱 확장할 수 있습니다.
- VPC에서 보안 그룹 관련 제한을 초과하지 않으려면 다음을 수행합니다.
    - 보안 그룹을 통합하여 제한을 초과하지 않게 유지합니다.
    - 보안 그룹 상호 참조를 사용하는 대신 CIDR 범위를 참조합니다.

## 모델 #4 - 컨테이너 계층의 테넌트 격리

컨테이너 기반 배포의 등장으로 인해 요구 사항에 따라 단일 인스턴스를 보유할 수 있으며 이를 다중 테넌트 애플리케이션에 대해 분할할 수 있습니다. Amazon EC2 Container Service(Amazon ECS)는 Docker 컨테이너 기반 배포를 쉽게 설정하고 관리하는 데 도움이 되며 개별 컨테이너에 테넌트별 솔루션 구성 요소를 배포하는 데 사용될 수 있습니다. 그림 11은 서로 다른 테넌트의 컨테이너가 동일한 VPC에 배포되는 시나리오를 보여 줍니다.

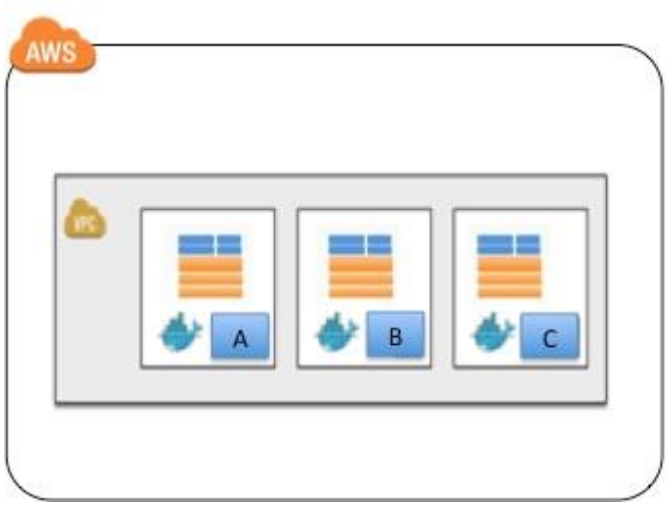


그림 11: 컨테이너 계층의 테넌트 격리

## 장점:

- 공유 인스턴스에서 컨테이너 기반 모델을 사용하여 더 높은 수준의 리소스 사용률을 얻을 수 있습니다.
- Amazon ECS는 클러스터 관리 및 일반적인 내결함성과 관련한 과도한 작업을 제거하여 클러스터를 대규모로 더욱 쉽게 관리할 수 있습니다.
- 테스트/개발 환경에서 도커 이미지를 테스트한 다음 간단한 CLI 기반 옵션을 사용하여 생산 환경에 직접 배치함으로써 배포를 단순화할 수 있습니다.
- Amazon ECS는 Amazon EC2 인스턴스에 이미지를 배포하며 VPC 기반 제어 기능을 통해 이 인스턴스를 더 세분화하고 제어할 수 있습니다. 이는 Docker의 자체 격리 모델과 더불어 대부분의 다중 테넌트 애플리케이션의 보안 요구 사항을 충족합니다.

## 단점:

- Amazon EC2 및 VPC 보안 그룹을 사용하여 Amazon EC2 인스턴스에서 트래픽을 제한할 수 있습니다. 그러나 개방되어 있는 포트를 제어하기 위해 컨테이너 구성을 관리해야 합니다. 이러한 측면을 관리하는 것은 약간의 번거로운 작업이 될 수 있습니다.
- 태그는 Amazon ECS 작업(컨테이너) 수준에서 작동하지 않으므로 태그를 기반으로 비용을 구분하지 않으며 사용자 지정 결제 계층이 필요하게 될 것입니다.

## 모범 사례:

- VPC 보안 그룹에서 제공하는 제어 기능 이상으로 컨테이너 통신을 보호하려면 GRE(Generic Routing Encapsulation)를 통해 지점 간 터널링을 사용하여 컨테이너에 대한 소프트웨어 정의 네트워크를 만들고 컨테이너 기반 서브넷 간에 트래픽을 라우팅할 수 있습니다.
- Amazon ECS를 사용하여 Auto Scaling 기능을 설계하려면 Amazon CloudWatch 및 AWS Lambda 기반 컨테이너 배포를 함께 사용하십시오. 이 설정에서는 그림 12와 같이 Amazon CloudWatch 경보가 AWS Lambda 기능을 트리거하여 다른 Amazon ECS 작업을 자동으로 추가하고 동적으로 확장합니다.

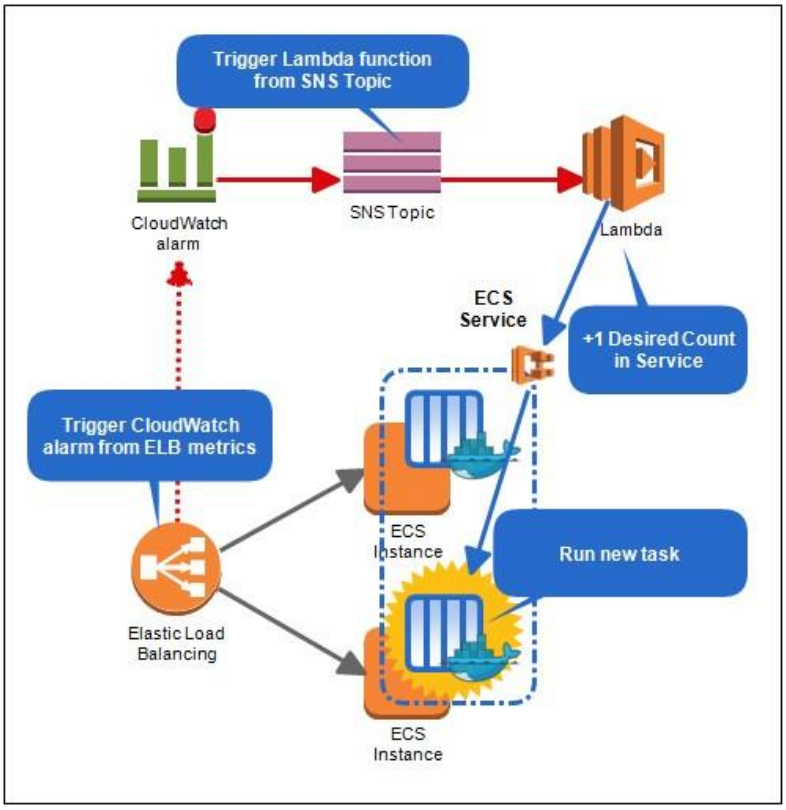


그림 12: 컨테이너 기반 배포에 대한 Auto Scaling 아키텍처

## 모델 #5 - 애플리케이션 계층의 테넌트 격리

이 모델은 이전에 살펴본 모델과는 큰 차이를 보입니다. 즉, 이제 애플리케이션 또는 솔루션 배포가 서로 다른 테넌트 간에 공유됩니다. 이는 급격한 변화이며 진정한 다중 테넌트 SaaS 모델을 향해 나아가는 것입니다. 하지만 이 모델을 사용하려면 애플리케이션 자체가 다중 테넌트를 지원하도록 설계되어야 합니다. 예를 들어 공유 웹 및 애플리케이션 계층을 통해 일반적인 3티어 애플리케이션을 얻는 경우 데이터베이스 계층(예: Amazon RDS 또는 Amazon EC2 인스턴스의 데이터베이스)에 약간의 변형이 있을 수 있습니다.

a) **별도의 데이터베이스**: 각 테넌트는 최대한 격리되기 위해 서로 다른 데이터베이스를 갖게 됩니다. 애플리케이션 계층이 각 테넌트의 요청에 따라 적절한 데이터베이스를 선택하도록 하려면 데이터베이스에 대한 테넌트의 매핑을 관리하는 별도의 스토어(Amazon DynamoDB 등)에 메타데이터를 유지해야 합니다.
b) **별도의 테이블/스키마**: 서로 다른 데이터베이스 버전은 각각 다른 구조를 가집니다. 그러나 모든 테넌트의 데이터가 동일한 데이터베이스에 위치하지만 데이터가 서로 다른 스키마나 테이블에 연결되어 격리 수준을 제공하는 또 다른 배포 모델도 있을 수 있습니다.
c) **공유 데이터베이스, 공유 스키마/테이블**: 이 모델에서는 모든 테넌트의 데이터가 함께 배치됩니다. 고유한 테넌트 ID 열은 각 테넌트에 대한 데이터 레코드를 구분합니다. 새 테넌트를 시스템에 추가해야 할 때마다 새 테넌트 ID가 생성되고 추가 용량이 프로비저닝되며 트래픽 라우팅이 기존 스택이나 새 스택으로 시작됩니다.

### 장점:

- 전체 스택에 걸쳐 규모의 경제와 더 나은 리소스 사용률 및 최적화를 달성할 수 있습니다. 결과적으로, 아키텍처 전반에 걸쳐 구성 요소를 공유할 때 대규모로 운영하기 위한 가장 경제적인 옵션으로 종종 사용될 수 있습니다.
    - 예를 들어 요청 쇄도를 처리할 수 있는 대규모 다중 테넌트 Amazon DynamoDB 테이블을 갖는 것은 개별 테넌트에 대해 프로비저닝된 Amazon DynamoDB 테이블을 사용하는 것보다 훨씬 경제적일 수 있습니다.
- 단일 배포이므로 스택을 관리하고 운영하는 것도 쉽습니다. *n*개의 서로 다른 환경을 관리해야 하는 대신 필요한 변경 사항이나 개선 사항을 한번에 실행해야 합니다.
- 단일 VPC 배포(크기는 더 클 수 있음)이기 때문에 네트워크 연결은 간단하며 다른 모델의 VPC 제한에 대한 문제도 줄어듭니다.

모든 공유 서비스(패치, OS 업데이트 및 안티바이러스 등)는 모든 테넌트에 대해 단일 단위로 중앙 집중화되어 배포됩니다.

### 단점:

- 애플리케이션은 다중 테넌트를 인식해야 하므로 기존 애플리케이션을 다시 설계해야 할 수 있습니다.
- 특정 규정 준수 및 보안 요구 사항에 따라 다른 보안 프로파일을 가진 테넌트는 공동 호스팅이 불가능할 수 있습니다.

### 모범 사례:

이 모델을 성공적으로 구현하려면 다음과 같은 중요한 측면을 고려하십시오.

- 종종 서로 다른 테넌트에는 특정 기능 또는 사용자 정의에 대한 고유한 요구 사항이 있습니다.
  - 요구 사항에 따라 테넌트를 그룹화하십시오. 유사한 요구 사항을 가진 테넌트는 동일한 배포에 속해야 합니다.
  - 핵심 플랫폼이나 애플리케이션 자체에 가장 필요한 기능을 구축하고 장기간 유지 관리를 위해 테넌트 수준의 사용자 정의를 피합니다.
- 각 테넌트 활동에 대한 스택을 긴밀히 모니터링합니다. 필요한 경우 다른 테넌트에게 부정적인 영향을 주지 않도록 특정 테넌트의 작업을 제한하거나 우선 순위를 낮춰야 합니다.
- 특정 스택에서 테넌트의 변화하는 요구 사항을 해결하려면 스택을 자동으로 확장하거나 축소할 수 있는 기능을 갖추어야 합니다. 이는 수동 업데이트로 수행되는 것이 아니라 아키텍처에 내장되어 있어야 합니다.
- 역할 기반 및 세분화된 액세스 제어를 사용하여 전체 스택에서 액세스 권한을 제한할 수 있게 합니다. Amazon DynamoDB는 세분화된 액세스 제어를 제공합니다. 이 기능은 Amazon DynamoDB 테이블 및 인덱스의 개별 데이터 항목과 속성, 그리고 여기에서 실행 가능한 작업에 액세스할 수 있는 사용자를 결정할 수 있게 해 줍니다. SaaS 아키텍처에서 Amazon DynamoDB를 사용하면 복잡성을 크게 줄일 수 있습니다.
- 테넌트의 사용량에 따른 AWS 비용 관리도 처리해야 할 중요한 측면입니다. 이 문제를 처리하려면 사용자 지정 계층을 설계하고(이전 섹션에서 설명한 바와 같음) 이를 솔루션에 통합하는 것이 좋습니다.

# 일반 권장 사항

패키지화된 SaaS 솔루션 설계 및 AWS에서 제공하는 것에 대한 다음과 같은 일반적인 모범 사례를 고려합니다.

- 대규모의 모놀리식 응용 프로그램 애플리케이션을 구축하는 대신 전체적인 비즈니스 기능을 얻기 위해 연동할 수 있는 소규모의 독립적인 단일 책임 서비스를 만드는 것이 종종 도움이 됩니다. 이러한 소규모 마이크로서비스 기반 아키텍처는 관리하기가 더 쉬울 수 있으며 독립적으로 확장할 수 있습니다. Amazon ECS 및 AWS Lambda 같은 서비스를 사용하여 이러한 소규모 구성 요소를 만들 수 있습니다. 또한 AWS Simple Queue Service (Amazon SQS) 도 통신에 대한 대기열 계층을 도입함으로써 마이크로서비스를 분리하는 데 잠재적으로 도움을 줄 수 있습니다. Amazon API Gateway를 사용해서 계층 간 API 기반 상호 작용을 활성화하여 인터페이스 계층에만 통합된 상태로 유지하게 할 수도 있습니다. 이 마이크로서비스 기반 아키텍처 패턴에 대해 자세히 알아보려면 SquirrelBin: A Serverless Microservice Using AWS Lambda 블로그 게시물을 참조합니다.
- 솔루션에 대해 향후 대비할 수 있도록 퍼블릭 인터페이스에 대한 영향 없이 기본 구현을 변경할 수 있게 하는 방식으로 각 계층에 추상화를 구축합니다. 향후 몇 년 내에 솔루션이 어떤 위치에 있길 바라는지 고려하고 기술 동향에 대해 생각해 보합니다. 예를 들어 5년 전에 모바일은 지금과 같은 큰 비중을 차지하진 않았습니다. 미래를 계획하고, 미래의 요구 사항을 충족할 수 있도록 확장 가능한 방식으로 솔루션을 설계합니다.
- 솔루션을 자주 업데이트할 수 있게 릴리스 관리 프로세스를 정의합니다. AWS CodeCommit, AWS CodePipeline 및 AWS CodeDeploy는 이러한 측면의 배포를 지원할 수 있습니다.
- 테넌트별 사용자 정의를 최소한으로 유지하고 플랫폼 자체 내에 대부분의 기능을 구축합니다. 테넌트별 구성 메타데이터의 경우 AWS DynamoDB가 유용할 수 있습니다.
- 타사 시스템과 통합해야 하는 경우 솔루션 또는 플랫폼용 API를 구축합니다.
- 다양한 애플리케이션 구성 요소 내에 하드 코딩된 자격 증명을 사용하는 대신 Amazon EC2에 대한 IAM 역할을 사용합니다.
- 솔루션을 비용 최적화하기 위한 방법을 찾습니다. 예를 들어 예약 인스턴스 또는 스팟 인스턴스를 사용하거나, AWS Lambda를 도입하여 이벤트 중심 아키텍처를 설계하거나, Amazon ECS를 사용하여 더 작은 기능 블록을 컨테이너화할 수 있습니다.

- Auto Scaling을 사용하여 환경을 로드별로 동적으로 확장 및 축소합니다.
- Amazon EC2 인스턴스의 크기와 개수를 적절히 조정할 수 있도록 애플리케이션 성능을 벤치마크합니다.
- AWS Trusted Advisor 권장 사항을 활용하여 AWS 배포를 더욱 최적화합니다.
- APN 기술 파트너에서 제공하는 패키지 솔루션으로 플랫폼에 구축하길 원하는 사용자가 종종 있습니다. 기존 솔루션을 활용하는 대신, 스스로 무엇을 구축할지 선택할 수 있는 기회를 찾습니다. 다양한 APN 파트너 솔루션 및 서비스, AWS Marketplace를 활용하여 AWS 서비스가 제공하는 기능을 더욱 강화합니다.
- AWS SaaS 파트너 프로그램 에 등록하여 AWS에서 SaaS 비즈니스에 대해 알아보고 구축하며 성장시킵니다.
- 회사가 AWS의 솔루션을 효과적으로 관리할 수 있는지 확인하는 것이 중요합니다. AWS MSP 컨설팅 파트너와 함께 작업하는 것도 또 다른 옵션이 됩니다.
- AWS 운영 체크리스트를 사용하여 운영 모델을 검증합니다.
- AWS 보안 감사 체크리스트를 사용하여 보안 모델을 검증합니다.
- 다양한 APN 파트너 솔루션 및 서비스, AWS Marketplace를 활용하여 AWS 서비스가 제공하는 기능을 강화합니다.

# 결론

패키지화된 SaaS 솔루션은 본질적으로는 모두 다르지만 서로 공통된 요소를 공유합니다. 이 문서에서 설명한 사례 및 아키텍처 방법을 사용하여 AWS에서 확장 가능하고 안전하며 최적화된 SaaS 솔루션을 배포할 수 있습니다. 이 문서는 사용자가 도입할 수 있는 서로 다른 모델에 대해 설명합니다. 사용자가 구축하고 있는 SaaS 솔루션의 유형에 따라 여러 모델을 사용하거나 하이브리드 방식을 사용하는 것이 요구 사항에 적합할 수 있습니다.

# 기고자

다음은 이 문서의 작성에 도움을 준 개인 및 조직입니다.

- Kamal Arora, 솔루션스 아키텍트, Amazon Web Services
- Tom Laszewski, 선임 관리자, 솔루션스 아키텍트, Amazon Web Services
- Matt Yanchyshyn, 선임 관리자, 솔루션스 아키텍트, Amazon Web Services

# 참고 문헌

## APN 파트너 솔루션

사용자 지정 SaaS 솔루션에서 다양한 기능을 구축하려면 다양한 기능에 걸쳐 자주 사용되는 ISV 솔루션과 통합해야 할 수 있습니다. 선택을 돕기 위해 APN은 AWS 컴피턴시 프로그램을 개발했습니다. 이 프로그램은 전문적인 솔루션 영역에서 기술적 숙련도와 입증된 고객 성공 사례가 있는 APN 파트너를 강조하기 위해 마련되었습니다. 다음은 일부 AWS 컴피턴시 솔루션 페이지입니다. 이 페이지에서 자세한 정보를 참조할 수 있습니다.

- DevOps - https://aws.amazon.com/solutions/partners/dev-ops/
- 모바일 - https://aws.amazon.com/mobile/partner-solutions/
- 보안 - https://aws.amazon.com/security/partner-solutions/
- 디지털 미디어 - https://aws.amazon.com/partners/competencies/digitalmedia/

- 마케팅 및 상거래 - https://aws.amazon.com/digitalmarketing/partner-solutions/
- 빅 데이터 - https://aws.amazon.com/partners/competencies/big-data/
- 스토리지 - https://aws.amazon.com/backup-recovery/partner-solutions/
  헬스케어 - https://aws.amazon.com/partners/competencies/healthcare/
- 생명 과학 - https://aws.amazon.com/partners/competencies/lifesciences/
- Microsoft 솔루션 - https://aws.amazon.com/partners/competencies/microsoft/
- SAP 솔루션 - https://aws.amazon.com/partners/competencies/sap/
- Oracle 솔루션 - https://aws.amazon.com/partners/competencies/oracle/

# 참고

- 다양한 AWS 사용 및 결제 보고서에 대한 세부 정보: http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-whatis.html
- Amazon EC2 IAM 역할: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-foramazon-ec2.html
- Amazon CloudWatch 및 AWS Lambda를 사용하는 Auto Scaling Amazon ECS 서비스 https://aws.amazon.com/blogs/compute/scaling-amazon-ecs-servicesautomatically-using-amazon-cloudwatch-and-aws-lambda/
- Tag Editor 작업 - http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/tag-editor.html
- 리소스 그룹 작업: http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/resourcegroups.html
- AWS를 이용한 백업 아카이브 및 복원 방식 - http://do.awsstatic.com/whitepapers/Backup_Archive_and_Restore_Approa ches_Using_AWS.pdf
- AWS 관리형 서비스 프로그램 - http://aws.amazon.com/partners/managed-service/
- AWS SaaS 파트너 프로그램 - http://aws.amazon.com/partners/saas/