SaaS 스토리지 전략

AWS 기반 다중 테넌트 스토리지 모델 구축

2016년 월 11일

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

서론 1

SaaS 파티셔닝 모델 1

사일로(silo) 모델 2

브리지(bridge) 모델 2

풀(pool) 모델 3

배경 설정 3

적합한 모델 찾기 3

트레이드오프 평가 4

사일로(silo) 모델 트레이드오프 5

풀(pool) 모델 트레이드오프 6

하이브리드: 비즈니스 절충 8

데이터 마이그레이션 9

마이그레이션 및 다중 테넌트 9

급격한 변경 최소화 10

보안 고려 사항 10

격리와 보안 11

관리 및 모니터링 11

스토리지 추세 집계 11

활동에 대한 테넌트 관점으로 보기 12

정책 및 경보 12

계층형 스토리지 모델 12

개발자 환경 13

연결 계정(Linked Account) 사일로(silo) 모델 14

DynamoDB 의 다중 테넌트 15

사일로(silo) 모델 15

브리지(bridge) 모델 17
풀(pool) 모델 18
샤드 배포 관리 20
동적으로 IOPS 최적화 21
다양한 환경 지원 21
마이그레이션 효율성 22
트레이드오프 비교 22
RDS 에서의 다중 테넌트 22
사일로(silo) 모델 23
브리지(bridge) 모델 24
풀(pool) 모델 26
단일 인스턴스 제한 고려 26
트레이드오프 비교 27
Amazon RedShift 에서의 다중 테넌트 27
사일로(silo) 모델 28
브리지(bridge) 모델 29
풀(pool) 모델 29
민첩성(Agility)을 고려 30
결론 31
기고자 32
참고 문헌 32
참고 32

요약

다중 테넌트 스토리지는 Software-as-a-Service(SaaS) 솔루션을 구축하고 제공하는 것에 대한 보다 어려운 측면을 보여 주는 요소입니다. 테넌트 데이터를 분할하기 위해 다양한 전략을 사용할 수 있으며 각 전략에는 다중 테넌트에 대한 접근 방식을 구성하는 고유의 미묘한 차이가 있습니다. 이러한 복잡성에 더불어 Amazon DynamoDB, Amazon Relational Database Service(Amazon RDS) 및 Amazon Redshift 등 AWS에서 제공하는 다양한 스토리지 모델에 각 전략을 매핑할 필요가 있습니다. 이러한 기술에 보편적으로 적용할 수 있는 상위 수준의 주제가 있긴 하지만 각 스토리지 모델에는 다중 테넌트 환경에서의 데이터 범위 설정, 관리 및 보안에 대한 자체적인 접근 방식이 있습니다. 이 문서에서는 SaaS 개발자에게 다양한 범위의 데이터 분할 옵션에 대한 통찰력을 제공하여 어떠한 전략 및 스토리지 기술의 조합이 SaaS 환경의 요구에 가장 부합하는지 확인할 수 있게 합니다.

서론

AWS는 Software-as-a-Service(SaaS) 개발자에게 데이터 범위 설정, 프로비저닝, 관리 및 보안에 대한 자체적인 접근 방식을 갖춘 다양한 스토리지 솔루션 모음을 제공합니다. 각 서비스가 데이터를 표시, 인덱싱 및 저장하는 방식도 다중 테넌트 전략에 고려해야 할 고유한 사항입니다. SaaS 개발자에게 이러한 스토리지 옵션의 다양성은 비즈니스 및 고객 요구에 가장 적합한 스토리지 기술을 SaaS 솔루션의 스토리지 요구에 적용할 수 있는 기회를 제공합니다.

또한 AWS 스토리지 옵션을 비교하면서 SaaS 솔루션의 다중 테넌트 모델이 각 스토리지 기술과 조화를 이루는 방법도 고려해야 합니다. 스토리지에 여러 버전이 있는 것처럼 다중 테넌트 파티션 전략에도 여러 버전이 있습니다. 스토리지와 테넌트 파티셔닝 요구에서 최적의 교차점을 찾는 것이 목표입니다.

이 문서에서는 이러한 문제에대해 가능한 모든 요소에 대해 살펴봅니다. 이 같은 요소는 다중 테넌트를 구현하는데 일반적으로 사용되는 모델을 확인하고 분류하며, 파티셔닝 모델을 선택하기 위한 장단점을 비교하는 데 도움이 됩니다. 이뿐만 아니라 Amazon RDS, Amazon DynamoDB 및 Amazon Redshift에서 각 모델이 실현되는 방법을 보여 주기도 합니다. 각 스토리지 기술을 자세히 살펴보면서 AWS의 구성을 사용하여 다중 테넌트 스토리지의 범위를 설정하고 관리하는 방법을 알아볼 수 있습니다.

이 문서에서는 다중 테넌트 파티셔닝 전략을 선택하기 위한 일반적인 지침을 제공하지만, 사용자 환경의 비즈니스, 기술 및 운영 차원에서 사용자의 접근 방식을 구성할 요인들을 소개하는 경우가 자주 있다는 점을 인식하는 것이 중요합니다. 많은 경우에 SaaS 조직은 이 문서에서 설명한 여러 옵션을 혼합한 하이브리드 형태 버전을 도입합니다.

SaaS 파티셔닝 모델

시작하려면 다양한 구현 전략을 이해하는 데 도움이 되도록 명확하게 정의된 개념 모델이 필요합니다.

그림 1은 SaaS 환경에서 테넌트 데이터를 파티셔닝할 때 일반적으로 사용되는 세 가지 기본 모델(사일로-silo, 브리지-bridge, 풀-pool)을 나타냅니다.

각 파티셔닝 모델은 테넌트 데이터를 관리하고 액세스하며 분리하는 데 매우 다른 접근 방식을 취합니다. 다음 섹션에서는 해당 모델에 대한 간략한 설명을 제공하여 특정 스토리지 기술 컨텍스트 이외의 각 모델의 가치와 원칙을 살펴볼 수 있게 합니다.

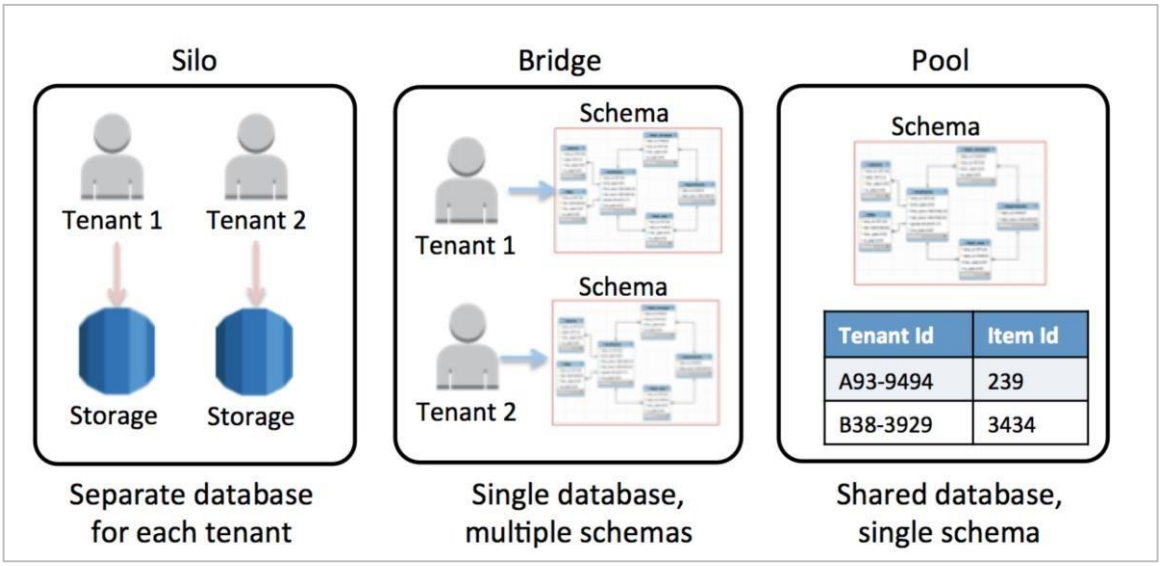

그림 1: SaaS 파티셔닝 모델

사일로(silo) 모델

사일로(silo) 모델에서 테넌트 데이터의 스토리지는 기타 테넌트 데이터와 완전하게 격리되어 있습니다. 테넌트 데이터를 나타내는 데 사용되는 모든 구성 요소는 해당 클라이언트에 대해 논리적으로 "고유한" 것으로 간주되므로 각 테넌트는 일반적으로 개별적인 표현, 모니터링, 관리 및 보안 영역을 갖습니다.

브리지(bridge) 모델

브리지(bridge) 모델은 종종 SaaS 개발자에게 매력적인 절충안이 됩니다. 브리지는 각 테넌트에 대해 어느 정도의 변형과 분리를 허용하는 한편 모든 테넌트의 데이터를 단일 데이터베이스로 옮깁니다. 사용자는 일반적으로 각 테넌트에 대해 별도의 테이블을 만들어서 이를 달성하며 각 테이블이 데이터(스키마)를 자체적으로 표현할 수 있게 허용합니다.

풀(pool) 모델

풀(pool) 모델은 테넌트가 시스템의 스토리지 구성 요소를 모두 공유하는 통합된 다중 테넌트 모델을 나타냅니다. 테넌트 데이터는 공통 데이터베이스에 배치되며 모든 테넌트는 공통 표현(스키마)을 공유합니다. 이를 위해서는 테넌트 데이터에 대한 액세스 범위를 설정하고 제어하는 데 사용되는 파티셔닝 키를 도입해야 합니다. 이 모델은 SaaS 솔루션의 프로비저닝, 관리 및 업데이트 환경을 단순화하는 경향이 있습니다. 또한 이 모델은 SaaS 공급자에게 필수적인 지속적인 제공 및 민첩성 목표에도 부합합니다.

배경 설정

사일로(silo), 브리지(bridge) 및 풀(pool) 모델은 논의 내용에 대한 배경을 제공합니다. 각 AWS 스토리지 기술을 살펴보면서 특정 AWS 스토리지 기술에서 이러한 모델의 개념적 요소가 어떻게 실현되는지 알아볼 수 있습니다. 일부 요소는 이 모델에 매우 직접적으로 매핑되며 다른 요소는 각 유형의 테넌트 격리를 얻는 데 더 많은 창의성을 필요로 합니다.

이러한 모델이 모두 유효하다는 점에 주목할 가치가 있습니다. 앞으로 각 모델의 장점을 논의하겠지만 주어진 환경의 규제, 비즈니스 및 레거시 차원은 결국 사용자가 선택하는 접근 방식을 구성하는 데 종종 중요한 역할을 합니다. 여기에서의 목표는 단순히 각 접근 방식과 관련된 메커니즘 및 트레이드오프에 대한 가시성을 확보하는 것입니다.

적합한 모델 찾기

다중 테넌트 파티셔닝 스토리지 모델 전략을 선택하는 것은 여러 가지 요소의 영향을 받습니다. 기존 솔루션에서 마이그레이션하는 경우 사일로(silo) 모델은 SaaS 애플리케이션을 다시 작성하지 않고도 다중 테넌트로 전환할 수 있는 가장 간단하고 명확한 방법을 만들기 때문에 이 모델을 도입하는 것을 선호할 수 있습니다. 더 격리된 모델이 필요한 규제 또는 업계의 역동적인 상황이 있는 경우 풀(pool) 모델의 효율성과 민첩성을 통해 빠르고 지속적인 릴리스를 수용하는 환경으로 나아갈 수 있는 방법을 발견할 수 있습니다. 여기서 핵심은 사용자가 선택하는 전략이 사용자 환경의 비즈니스와 기술적 고려 사항을 조합하여 진행된다는 점을 확인하는 것입니다.

다음 섹션에서는 각 모델의 장점과 단점을 강조하고 명확하게 정의된 데이터 요소 집합을 제공하여 더 광범위한 평가의 일부로 사용할 수 있게 합니다. 종종 SaaS 모델을 도입하는 핵심이 되는 민첩성 목표에 부합하도록 각 모델이 사용자의 기능에 어떤 영향을 주는지 알아봅니다. SaaS 환경에 대한 아키텍처 전략을 선택할 때 이 전략이 서비스를 중지할 수 없는 환경에서 신속하게 버전을 구축, 제공 및 배포하는 능력에 어떻게 영향을 미치는지 고려하십시오.

트레이드오프 평가

사일로(silo), 브리지(bridge) 및 풀(pool) 세 가지 파티셔닝 모델을 스펙트럼에 추가하면, 이러한 모델 중 하나를 도입하는 것이 다른 모델의 특성과 트레이드오프가 있음을 확인할 수 있습니다. 한 모델의 장점으로 나열된 특성은 종종 다른 모델의 단점으로 나타납니다. 예를 들어, 사일로(silo) 모델의 원칙 및 가치 시스템은 종종 풀(pool) 모델과 반대입니다.

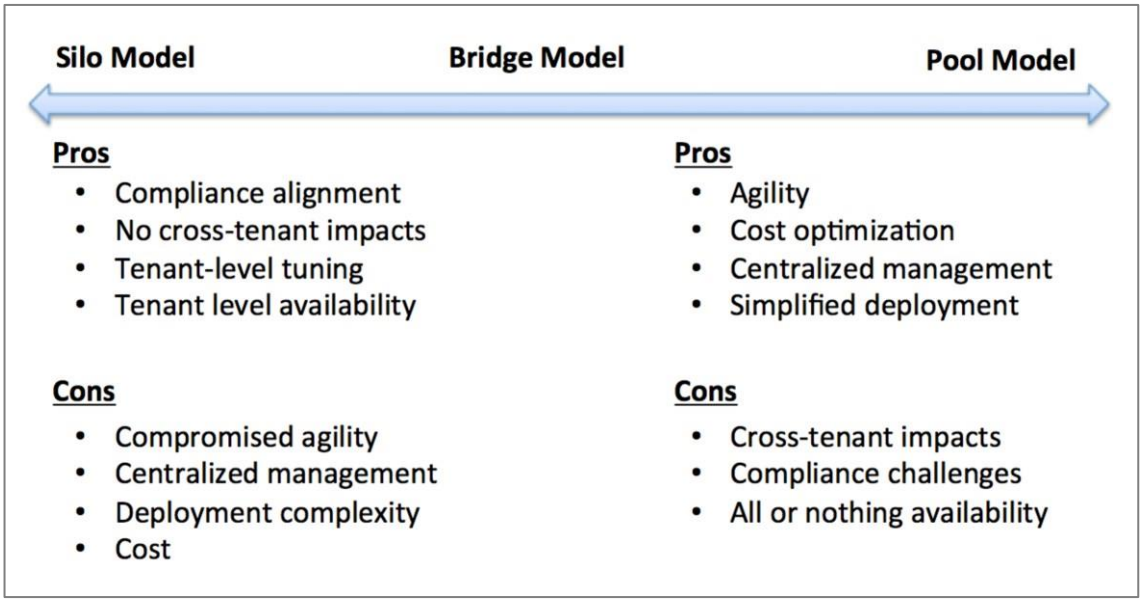

그림 2: 파티셔닝 모델 트레이드오프

그림 2는 이러한 상충되는 원칙을 강조하고 있습니다. 다이어그램 상단에는 세 가지 파티셔닝 모델이 표시되어 있습니다. 왼쪽에는 사일로(silo) 모델과 관련된 장점과 단점이 나와 있습니다. 오른쪽에는 풀(pool) 모델의 비슷한 목록이 있습니다. 브리지(bridge) 모델은 이러한 고려 사항을 혼합한 것으로, 양극단에 표시된 장점과 단점의 조합을 보여 주고 있습니다.

사일로(silo) 모델 트레이드오프

테넌트 데이터를 완전히 다른 데이터베이스로 표현하는 것이 적합할 수 있습니다. 이 접근 방식은 기존의 단일 테넌트 솔루션의 마이그레이션을 단순화하는 것 외에도 일부 테넌트가 완전한 공유 인프라를 운영하는 데 발생할 수 있는 우려 사항을 해결합니다.

장점

사일로(silo)는 엄격한 규제 및 보안 제약 조건이 있는 SaaS 솔루션에 적합합니다. 이러한 환경에서 SaaS 고객은 데이터를 다른 테넌트와 격리해야 하는 방법에 대한 매우 특정한 기대를 보입니다. 사일로(silo) 모델을 통해 테넌트는 테넌트 데이터 간에 더욱 구체적인 경계를 만드는 옵션을 가질 수 있으며 고객은 데이터가 전용 모델에 저장된다는 점을 알 수 있습니다.

교차 테넌트 효과는 제한될 수 있습니다. 여기에서 중요한 점은 사일로(silo) 모델을 격리하면 한 테넌트의 활동이 다른 테넌트에게 영향을 미치지 않도록 할 수 있다는 것입니다. 이 모델을 사용하면 시스템의 데이터베이스 성능 SLA가 주어진 테넌트의 요구에 맞게 조정될 수 있는 테넌트별 조정이 가능합니다. 데이터베이스를 조정하는 데 사용되는 노브와 다이얼은 일반적으로 사일로(silo) 모델에 더욱 자연스럽게 매핑되므로 테넌트 중심의 환경을 더 간편하게 구성할 수 있습니다.

테넌트 수준에서 가용성을 관리하여 테넌트의 중단을 최소화합니다. 자체 데이터베이스에 각 테넌트가 있으므로 데이터베이스 중단이 모든 테넌트에 계승될 수 있다는 우려를 할 필요가 없습니다. 한 테넌트에 데이터 문제가 있는 경우 시스템의 다른 테넌트에 부정적인 영향을 미칠 가능성이 낮습니다.

단점

프로비저닝과 관리는 더 복잡합니다. 테넌트별로 인프라를 도입할 때마다 테넌트 단위로 프로비저닝하고 관리해야 하는 또 다른 동적 요소도 도입합니다. 예를 들어 사일로(silo) 데이터베이스 솔루션이 테넌트가 생성될 때 시스템에 어떤 영향을 미칠지 상상해 보십시오. 가입 프로세스에는 온보딩 프로세스 중에 데이터베이스를 만들고 구성하는 자동화가 필요합니다. 이는 확실하게 달성 가능한 것이지만 SaaS 환경에서 복잡성과 잠재적인 장애 지점을 추가하기도 합니다.

테넌트 활동을 관찰하고 바로 반응하는 시스템 구축이 어렵습니다. 사용자는 SaaS를 통해 시스템 상태에 대한 교차 테넌트 보기 기능을 제공하는 관리 및 모니터링 환경을 원할 수 있습니다. 데이터베이스 성능 문제를 사전에 예측하고 더욱 종합적인 방식으로 정책에 대응하려고 합니다. 하지만 사일로(silo) 모델을 사용하면 모든 테넌트를 포괄하는 결합된 시스템 전체 상태 보기를 만들 수 있는 도구를 찾고 도입하는 것이 더 어려워집니다.

사일로(silo) 모델의 분산 특성은 테넌트의 성능 추세를 효과적으로 분석하고 평가하는 기능에 영향을 미칩니다. 각 테넌트가 자체 스토리지에 데이터를 저장하면 사용자는 테넌트 중심 모델에서 서비스 부하만 관리하고 조정할 수 있습니다. 결국 본질적으로 사용자가 관리하고 조정해야 하는 일회성 설정 및 정책의 도입으로 이어집니다. 이는 비효율적일 수 있으며 고객의 요구에 신속하게 대응할 수 있는 능력을 약화시키는 오버 헤드를 초래할 수 있습니다.

사일로(silo)는 비용 최적화를 제한합니다. 가장 큰 단점은 사일로(silo) 모델이 일회성 특성으로 인해 스토리지 리소스 사용량을 조정하는 능력을 제한하는 경향이 있다는 점일 것입니다.

풀(pool) 모델 트레이드오프

풀(pool) 모델은 SaaS 방식에 대한 궁극적인 통합 노력을 보여 줍니다. 풀(pool) 모델을 사용하면 테넌트 스토리지 프로비저닝, 마이그레이션, 관리 및 모니터링을 간소화할 수 있는 통합 접근 방식을 갖는 데 중점을 두게 됩니다.

장점

민첩성. 모든 테넌트 데이터가 하나의 스토리지 구조로 중앙 집중화되고 나면 간소화된 보편적인 접근 방식을 지원하는 도구 및 수명 주기를 훨씬 쉽게 만들어 모든 테넌트에 대해 스토리지 솔루션을 신속하게 배포할 수 있습니다. 이러한 민첩성은 온보딩 프로세스로도 확장됩니다. 풀(pool) 모델을 사용하면 SaaS 서비스에 등록하는 각 테넌트에 대해 별도의 스토리지 인프라를 프로비저닝할 필요가 없습니다. 간편하게 새 테넌트를 프로비저닝하고 테넌트의 ID를 인덱스로 사용하여 모든 테넌트가 사용하는 공유 스토리지 모델에서 테넌트 데이터에 액세스할 수 있습니다.

스토리지 모니터링 및 관리도 더 간단합니다. 풀(pool) 모델에서는 테넌트 스토리지 활동을 요약할 수 있는 위치에 도구 및 집계 분석을 배치하는 것이 훨씬 자연스럽습니다. 단일 스토리지 모델을 관리하는 데 사용할 수 있는 일상적인 도구를 여기에서 활용하여 시스템 상태에 대한 포괄적인 교차 테넌트 보기를 구축할 수 있습니다. 풀(pool) 모델을 사용하면 사전에 시스템 이벤트에 대응하는 데 사용할 수 있는 전역적 정책을 훨씬 쉽게 도입할 수 있습니다. 일반적으로 데이터를 단일 데이터베이스 및 공유 표현으로 통합하면 다중 테넌트 스토리지, 배포 및 관리 환경의 여러 측면이 단순화됩니다.

추가 옵션은 SaaS 솔루션의 비용 규모를 최적화하는 데 도움이 됩니다. 비용 기회는 종종 성능 튜닝의 형태로 나타납니다. 예를 들어, 테넌트 단위로 구분된 정책을 관리하는 대신 하나의 정책으로 모든 테넌트에 적용되는 처리량 최적화를 얻을 수 있습니다.

풀은 배포 자동화 및 운영 민첩성을 개선합니다. 풀(pool) 모델의 공통적인 특성은 일반적으로 새 제품 기능의 지속적이고 정기적인 릴리스에 대한 SaaS 요구와 부합하는 데이터베이스 배포 자동화의 전체적인 복잡성을 줄입니다.

단점

민첩성은 규모와 가용성을 관리하기 위한 높은 기준을 의미합니다. 풀링된 다중 테넌트 환경에서 스토리지 중단이 미치는 영향을 상상해 보십시오. 이제 한 명의 고객만 중단되는 대신 *모든* 고객이 중단됩니다. 그렇기 때문에 풀(pool) 모델을 도입한 조직은 환경 자동화 및 테스트에 훨씬 더 많은 투자를 하는 경향이 있습니다. 풀링된 솔루션에는 사전 모니터링 솔루션과 강력한 버전 관리, 데이터 및 스키마 마이그레이션이 필요합니다. 릴리스는 원활하게 진행되어야 하며 테넌트 문제는 효율적으로 포착되고 표면화되어야 합니다.

풀에는 테넌트 데이터 배포 관리에 대한 어려움이 있습니다. 경우에 따라 테넌트 데이터의 크기 및 배포 역시 풀링된 스토리지에 대한 어려움이 될 수 있습니다. 테넌트는 시스템에 다양한 수준의 부하를 가하는 경향이 있으며, 이러한 변화는 스토리지 성능을 약화시킬 수 있습니다. 풀(pool) 모델에서는 테넌트 부하의 이러한 변화를 고려하기 위해 사용할 메커니즘에 대해 더욱 깊이 생각해 보아야 합니다. 데이터의 크기 및 배포는 데이터 마이그레이션 접근 방식에도 영향을 줄 수 있습니다. 이러한 문제는 일반적으로 지정된 스토리지 기술에 고유한 것이며 개별적으로 해결되어야 할 필요가 있습니다.

풀링된 환경의 공통적인 특성은 일부 영역에서 저항에 부딪힐 수 있습니다. 일부 SaaS 제품의 경우 고객은 규제 및 내부 데이터 보호 요구를 해결하기 위해 사일로(silo) 모델을 요구할 것입니다.

하이브리드: 비즈니스 절충

많은 조직의 경우 전략을 선택하는 것은 사일로(silo), 브리지(bridge) 또는 풀(pool) 모델을 선택하는 것만큼 간단하지 않습니다. 테넌트 및 비즈니스는 스토리지 전략 선택에 대한 접근 방식에 중대한 영향을 미치게 됩니다.

일부 경우에 팀은 사일로(silo) 또는 브리지(bridge) 모델이 필요한 소규모 테넌트 모음을 파악할 수 있습니다. 일단 이러한 결정을 내리고 나면 그 모델로 모든 스토리지를 구현해야 한다고 가정합니다. 이는 풀(pool) 모델에 공개되어 있을 수 있는 테넌트를 수용하는 기능을 인위적으로 제한합니다. 사실 사일로(silo) 또는 브리지(bridge) 모델의 속성이 필요하지 않는 테넌트 계층에 대해 비용 또는 복잡성이 추가될 수 있습니다.

사용 가능한 절충안 하나는 풀링된 스토리지를 기반으로 완벽하게 지원하는 솔루션을 구축하는 것입니다. 그런 다음 사일로(silo) 스토리지 솔루션을 필요로 하는 테넌트를 위해 별도의 데이터베이스를 만들 수 있습니다. 그림 3은 작업에 대한 이러한 접근 방식의 예를 보여 줍니다.

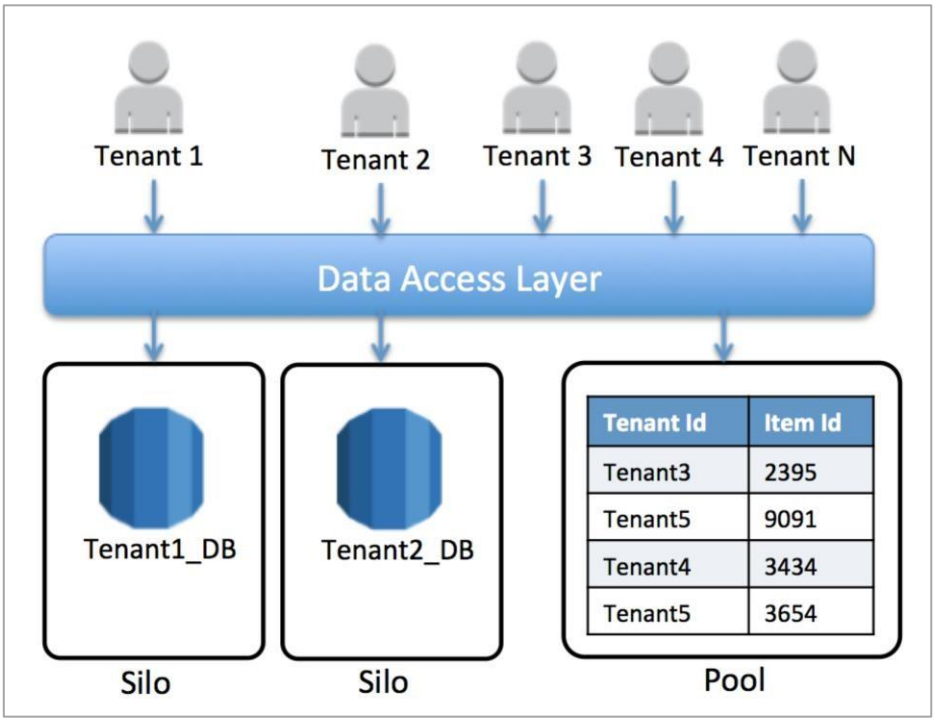

그림 3: 사일로(silo)/풀 스토리지 혼합

여기에는 사일로(silo) 모델을 활용하는 두 개의 테넌트(테넌트 1 및 테넌트 2)가 있고, 나머지 테넌트는 풀링된 스토리지 모델에서 실행됩니다. 이는 데이타 액세스 레이어를 통해 테넌트의 스토리지가 많이 추상화 되었습니다.

이렇게 하면 데이터 액세스 계층 및 관리 프로파일에 복잡성이 추가될 수 있지만 비즈니스에 두 모델의 장점을 최대한 나타낼 수 있는 서비스 계층화 방법을 제공할 수 있습니다.

데이터 마이그레이션

데이터 마이그레이션은 상충되는 SaaS 스토리지 모델 평가에서 제외되는 영역 중 하나입니다. 하지만 SaaS를 통해 아키텍처 선택이 새로운 기능과 역량을 지속적으로 배포하는 데 어떻게 영향을 미치는지 고려해 보십시오. 성능 및 일반 테넌트 환경을 강조하는 것이 중요하지만 스토리지 솔루션이 데이터의 기본 표현에 대한 지속적인 변경 사항을 어떻게 수용하는지 고려하는 것도 중요합니다.

마이그레이션 및 다중 테넌트

각각의 다중 테넌트 스토리지 모델은 데이터 마이그레이션을 해결하기 위한 고유한 접근 방식을 필요로 합니다. 사일로(silo) 및 브리지(bridge) 모델에서는 데이터를 테넌트 단위로 마이그레이션할 수 있습니다. 모든 테넌트에 대해 마이그레이션 오류가 발생할 가능성 없이 각 SaaS 테넌트를 신중하게 마이그레이션할 수 있으므로 조직은 이를 매력적으로 느낄 수 있습니다. 그러나 이 접근 방식은 배포 수명 주기의 전반적인 오케스트레이션 과정을 더욱 복잡하게 만들 수 있습니다.

풀(pool) 모델에서 데이터를 마이그레이션하는 것은 매력적이지만 어려운 문제이기도 합니다. 한 가지 측면에서 볼 때 풀(pool) 모델의 마이그레이션은 일단 마이그레이션되면 모든 테넌트가 새로운 데이터 모델로 성공적으로 전환되는 단일 지점을 제공합니다. 반면에 풀 마이그레이션 중에 발생하는 문제는 모든 테넌트에게 영향을 미칠 수 있습니다.

시작 단계부터 데이터 마이그레이션이 전체적인 다중 테넌트 SaaS 전략에 어떻게 조화를 이루는지 생각해야 합니다. 초기에 이 마이그레이션 오케스트레이션을 제공 파이프 라인에 적용하면 릴리스 프로세스의 민첩성을 향상시키는 경향이 있습니다.

급격한 변경 최소화

경험으로 보아 시스템 데이터가 어떻게 발전할지 고려하면서 명확한 정책과 원칙을 따라야 합니다. 가능한 경우 팀은 이전 버전과의 호환성이 있는 데이터 변경을 중요시해야 합니다. 애플리케이션의 데이터 표현에 대한 변경을 최소화할 수 있는 방법을 찾는 경우 데이터를 새로운 표현으로 변환하는 데 따르는 높은 오버 헤드를 제한할 수 있습니다.

일반적으로 사용되는 도구와 기술을 활용하여 마이그레이션 프로세스를 조율할 수 있습니다. 현실적으로 급격한 변경을 최소화하는 것은 종종 SaaS 개발자들에게 매우 중요하며 SaaS 도메인에서만 고유한 것이 아닙니다. 그러므로 이 문서에서 다룰 내용의 범위를 벗어납니다.

보안 고려 사항

데이터 보안은 SaaS 공급자에게 가장 중요한 문제가 되어야 합니다. 다중 테넌트 전략을 도입할 때 조직은 테넌트 데이터가 무단으로 접근되는 것으로부터 효과적으로 보호되도록 강력한 보안 전략을 필요로 합니다. 이 데이터를 보호하고 시스템이 적절한 보안 조치를 사용했다는 내용을 전달하는 것은 SaaS 고객의 신뢰를 얻는 데 매우 중요합니다.

사용자가 선택하는 스토리지 전략은 AWS에서 지원되는 일반적인 보안 패턴을 사용할 가능성이 높습니다. 예를 들어, 데이터를 암호화하는 것은 모든 모델에서 보편적으로 적용될 수 있는 수평적인 전략입니다. 이렇게 하면 데이터에 대한 무단 액세스가 있더라도 정보를 해독하는 데 필요한 키가 없으면 쓸모가 없게 되는 기본 보안 수준이 제공됩니다.

이제 사일로(silo), 브리지(bridge) 및 풀(pool) 모델의 보안 프로파일을 살펴보면 각 모델에서 보안이 실현되는 방법에 대한 추가적인 변화를 알 수 있을 것입니다.

예를 들어, AWS Identity and Access Management(Amazon IAM)가 테넌트 데이터에 대한 액세스 범위를 설정하고 제어하는 방법에 미묘한 차이가 있다는 것을 발견할 수 있을 것입니다. 일반적으로 사일로(silo) 및 브리지(bridge) 모델은 전체 데이터베이스 또는 테이블에 대한 액세스를 제한하는 데 적용될 수 있기 때문에 IAM 정책에 더 적합합니다. 풀(pool) 모델로 바꾸면 IAM을 활용하여 데이터에 대한 액세스 범위를 설정할 수 없습니다. 대신 애플리케이션 서비스의 인증 모델에 더 많은 책임이 생깁니다. 이러한 서비스는 사용자의 자격 증명을 사용하여 공유 표현의 데이터에 대해 보유한 범위와 제어를 해결해야 합니다.

격리와 보안

테넌트 격리를 지원하는 것은 일부 조직과 도메인의 기본 요소입니다. 가상화 환경에서도 데이터가 구분된다는 개념은 특정 규제 또는 보안 요구가 있는 SaaS 공급자에게 필수적인 것으로 간주될 수 있습니다.

각 AWS 스토리지 솔루션을 고려할 때 각각의 AWS 스토리지 서비스에서 어떻게 격리가 이루어지는지 생각해 보십시오. 앞으로 확인할 수 있겠지만 RDS에서 격리를 수행하는 것은 DynamoDB에서 수행하는 것과는 매우 다릅니다. 스토리지 전략을 선택하고 고객의 보안 고려 사항을 평가할 때 이러한 차이점을 고려하십시오.

관리 및 모니터링

다중 테넌트 스토리지에 대해 도입하는 접근 방식은 SaaS 솔루션의 관리 및 모니터링 프로파일에 중요한 영향을 줄 수 있습니다. 사실 시스템 상태를 집계하고 분석하기 위해 취하는 복잡성과 접근 방식은 각 스토리지 모델과 AWS 기술에 따라 크게 다를 수 있습니다.

스토리지 추세 집계

SaaS 스토리지의 효과적인 운영 보기를 구축하려면 테넌트 활동의 집계 보기를 제공하는 지표 및 대시보드가 필요합니다. 모든 테넌트를 포괄하는 환경에 영향을 줄 수 있는 스토리지 추세를 사전에 파악할 수 있어야 합니다. 이러한 집계 보기를 만드는 데 필요한 메커니즘은 사일로(silo) 모델과 풀(pool) 모델에서 매우 다르게 나타납니다. 사일로(silo) 스토리지를 사용하면 격리된 각 데이터베이스의 데이터를 수집하고 집계된 모델에서 해당 정보를 표시하기 위한 도구를 사용해야

합니다. 반면 풀(pool) 모델은 본질적으로 이미 테넌트 활동에 대한 집계 보기를 갖추고 있습니다.

활동에 대한 테넌트 관점으로 보기

관리 및 모니터링 스토리지 솔루션은 스토리지 활동에 대한 테넌트 관점으로 보기를 생성하는 방법을 제공해야 합니다. 특정 테넌트에 스토리지 문제가 있는 경우 스토리지 측정치 및 프로파일 데이터를 조사하여 개별 테넌트에 영향을 미칠 수 있는 사항을 파악할 필요가 있을 수 있습니다. 여기서 사일로(silo) 모델은 특성상 스토리지 활동에 대한 테넌트 관점으로 보기를 구성하는 것에 더욱 적합합니다. 풀링된 스토리지 전략에는 지정된 테넌트에 대한 스토리지 활동을 추출하기 위한 일부 테넌트 필터링 메커니즘이 필요합니다.

정책 및 경보

각 AWS 스토리지 서비스에는 애플리케이션의 스토리지 성능을 평가하고 튜닝하는 자체 메커니즘이 있습니다. 스토리지는 종종 시스템의 핵심 병목 지점을 나타낼 수 있으므로 애플리케이션 스토리지의 상태 변화를 감지하고 대응하는 모니터링 정책과 경보를 도입해야 합니다.

사용자가 선택하는 파티셔닝 모델은 스토리지 모니터링 전략의 복잡성과 관리 효율성에도 영향을 미칩니다. 솔루션을 격리할수록(사일로(silo)) 테넌트 단위로 관리하고 유지해야 하는 동적 요소가 많아집니다. 반면 풀링된 스토리지 전략의 공유 특성을 통해 더 간단하게 중앙 집중식, 교차 테넌트 정책 및 경보 모음을 더 많이 얻을 수 있습니다.

이러한 스토리지 정책의 전체적인 목표는 상태 이벤트를 예측하고 이에 대응할 수 있는 일련의 사전 규칙을 마련하는 것입니다. 다중 테넌트 스토리지 모델을 선택할 때 각 접근 방식이 시스템의 스토리지 정책 및 경보를 구현하는 방법에 어떤 영향을 줄 수 있는지 고려하십시오.

계층형 스토리지 모델

AWS는 개발자에게 각 서비스를 조합하여 SaaS 테넌트의 다양한 비용 및 성능 요구를 해결하는 데 적용할 수 있는 광범위한 스토리지 서비스를

제공합니다. 여기에서 핵심은 스토리지 전략을 AWS 서비스 또는 스토리지 기술에 인위적으로 제한하는 것이 아닙니다.

애플리케이션의 스토리지 요구를 프로파일링하면서, 주어진 스토리지 서비스의 장점을 애플리케이션의 다양한 구성 요소의 특정 요구와 맞추는 더욱 세부적인 접근 방식을 취하십시오. 예를 들어, DynamoDB는 한 애플리케이션 서비스에 매우 적합할 수 있지만 RDS는 다른 애플리케이션 서비스에 더 적합할 수 있습니다. 솔루션에 대해 각 서비스마다 자체 스토리지 보기가 있는 마이크로 서비스 아키텍처를 사용하는 경우 각 서비스 프로파일에 가장 적합한 스토리지 기술을 생각해 보십시오. 애플리케이션을 구성하는 일련의 마이크로 서비스에 사용되는 다양한 스토리지 솔루션의 스펙트럼을 찾는 것은 드문 일이 아닙니다.

또한 이 전략은 스토리지를 SaaS 솔루션을 계층화하는 또 다른 방법으로 사용할 수 있는 기회를 만듭니다. 각 계층은 기본적으로 솔루션의 계층에 대한 가치 제안을 구분할 수 있는 다양한 수준의 성능 및 SLA를 제공하는 별도의 스토리지 전략을 활용할 수 있습니다. 이 접근 방식을 사용하면 테넌트 계층이 인프라에 부과하는 비용 및 부하에 더욱 부합할 수 있습니다.

개발자 환경

일반적인 아키텍처 원칙에 따라 개발자는 전형적으로 애플리케이션의 수평적 측면을 중앙 집중화하고 추상화하는 계층 또는 프레임워크를 도입하려고 시도합니다. 여기에서 목표는 정책과 테넌트 해결 전략을 중앙 집중화하고 표준화하는 것입니다. 예를 들어, 데이터 액세스 요청에 테넌트 컨텍스트를 주입하는 데이터 액세스 계층을 도입할 수 있습니다. 이는 개발을 단순화하고 시스템을 통해 테넌트 자격 증명이 진행되는 방법에 대한 개발자의 인식을 제한합니다.

이 계층을 사용하면 테넌트 단위로 다를 수 있는 정책 및 전략에 대한 더 많은 옵션이 제공됩니다. 또한 스토리지 활동의 구성 및 추적을 중앙 집중화할 수 있는 특성상의 기회를 제공합니다.

연결 계정(Linked Account) 사일로(silo) 모델

각 스토리지 서비스의 특성을 살펴보기 전에 AWS 연결 계정을 사용하여 AWS 스토리지 솔루션을 기반으로 사일로(silo) 모델을 구현하는 방법을 알아보겠습니다. 이 접근 방식으로 사일로(silo)를 구현하려면 솔루션이 모든 테넌트에 대해 별도의 연결 계정을 프로비저닝해야 합니다. 테넌트에 대한 전체 인프라가 다른 테넌트와 완전히 격리되어 있기 때문에 완벽하게 사일로(silo)를 구현할 수 있습니다.

연결 계정 방식은 고객이 하위 계정을 전체 지급인(payer) 계정과 연결할 수 있는 통합 결제 기능을 사용합니다. 여기에서는 각 테넌트에 대해 별도의 연결 계정을 사용하더라도 이러한 테넌트에 대한 결제가 집계되어 단일 청구서의 일부로 지급인 계정에 표시됩니다.

그림 4는 연결 계정을 사용하여 사일로(silo) 모델을 구현하는 방법에 대한 개념을 보여 줍니다. 여기에는 별도의 계정을 가진 2개의 테넌트가 있고 각 계정은 지급인 계정과 연결되어 있습니다. 이러한 격리성을 통해 사용자는 사용 가능한 AWS 스토리지 기술을 자유롭게 활용하여 테넌트의 데이터를 저장하게 됩니다.

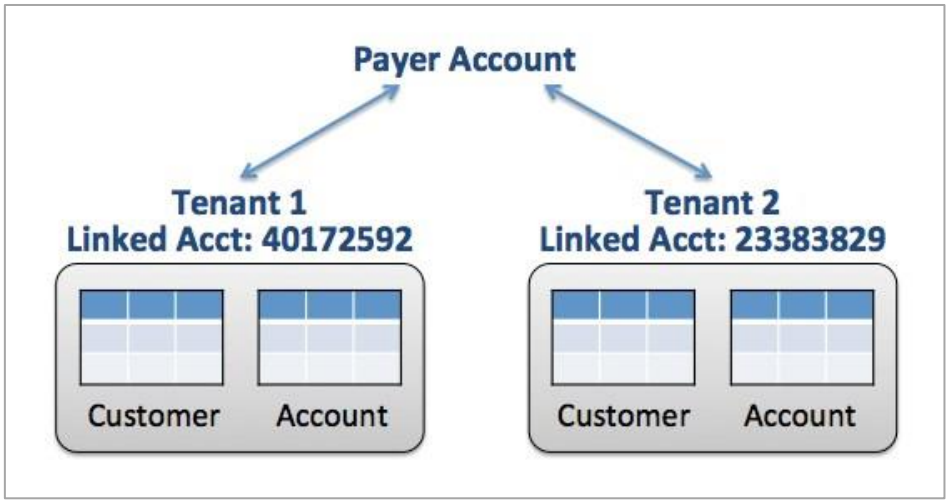

그림 4: 연결 계정을 사용하는 사일로(silo)

언뜻 보기에 이것은 사일로(silo) 환경이 필요한 SaaS 공급자에게 매우 매력적인 전략으로 보일 수 있습니다. 분명하게도, 개별 테넌트의 관리 및 마이그레이션에 대한 일부 측면을 단순화할 수 있습니다. 연결 계정 수준으로 AWS 비용을 요약할 수 있기 때문에 테넌트 비용 보기도 더욱 간단히 수집할 수 있습니다.

이러한 장점이 있긴 하지만 연결 계정 사일로(silo) 모델에는 중요한 제한이 있습니다. 예를 들어 프로비저닝이 더 복잡합니다. 사용자는 테넌트의 인프라를 만드는 것 외에도 각 연결 계정의 생성을 자동화하고 이를 필요로 하는 모든 제한을 조정해야 합니다. 하지만 더 큰 문제는 바로 규모입니다. AWS는 사용자가 만들 수 있는 연결 계정의 수에 제약이 있으며 이러한 제한은 새로운 SaaS 테넌트를 대량으로 생성하게 될 환경과 부합하지 않을 가능성이 높습니다.

DynamoDB의 다중 테넌트

DynamoDB에서 데이터의 범위를 설정하고 관리하는 방법의 특성은 다중 테넌트에 접근하는 방식에 몇 가지 새로운 단서를 줍니다. 일부 스토리지 서비스는 기존의 데이터 파티셔닝 전략에 적절히 부합할 수 있지만 DynamoDB는 사일로(silo), 브리지(bridge) 및 풀(pool) 모델에 대한 직접적인 매핑이 약간 적습니다. DynamoDB를 사용하면 다중 테넌트 전략을 선택할 때 몇 가지 추가 요소를 고려해야 합니다.

다음 섹션에서는 DynamoDB의 각 다중 테넌트 파티셔닝 계획을 실현하는 데 일반적으로 사용되는 AWS 메커니즘을 살펴봅니다.

사일로(silo) 모델

DynamoDB에서 사일로(silo) 모델을 구현하는 방법을 살펴보기 전에 먼저 데이터에 대한 서비스 범위 및 액세스 제어 방법을 고려해야 합니다. RDS와는 달리 DynamoDB에는 데이터베이스 인스턴스에 대한 개념이 없습니다. 대신 DynamoDB에서 생성된 모든 테이블은 리전 내의 계정에 대해 전역적입니다. 즉, 해당 리전의 모든 테이블 이름은 주어진 계정에 대해 고유해야 합니다.

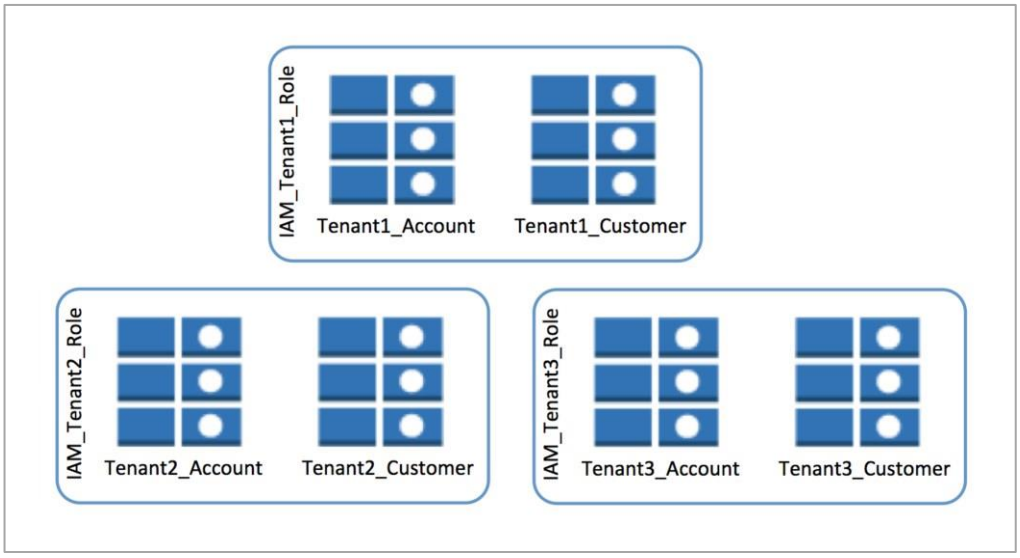

그림 5: DynamoDB 테이블을 사용하는 사일로(silo) 모델

DynamoDB에 사일로(silo) 모델을 구현하는 경우 특정 테넌트와 연결된 하나 이상의 테이블 그룹을 만드는 방법을 찾아야 합니다. 또한 사일로(silo) 고객의 보안 요구를 충족하기 위해 이러한 테이블에 대한 안전하고 통제되는 보기를 만들어 교차 테넌트 데이터 액세스 가능성을 방지하는 방식으로 접근해야 합니다.

그림 5는 이 테넌트 범위로 테이블을 그룹화할 수 있는 방법의 한 가지 예를 보여줍니다. 각 테넌트에 대해 두 개의 테이블이 생성됩니다(계정 및 고객). 이러한 테이블에는 테이블 이름 앞에 붙은 테넌트 식별자도 있습니다. 이는 DynamoDB의 테이블 이름 지정 요구를 해결하고 테이블 및 연결된 테넌트 간에 필요한 바인딩을 만듭니다.

이러한 테이블에 대한 액세스는 IAM 정책의 도입을 통해 이루어집니다. 각 테넌트에 대한 정책 작성을 자동화하고 지정된 테넌트가 소유한 테이블에 해당 정책을 적용하는 프로비저닝 프로세스가 필요합니다.

이 접근 방식은 각 테넌트의 데이터 간 명확한 경계를 정의하는 사일로(silo) 모델의 근본적인 격리 목표를 달성합니다. 또한 테넌트 단위로 튜닝 및 최적화가 가능합니다. 사용자는 두 가지 특정 영역을 조정할 수 있습니다.

- Amazon CloudWatch 측정치는 테이블 수준에서 캡처되어 스토리지 활동에 대한 테넌트 측정치 집계를 단순화할 수 있습니다.

- 초당 입력/출력 작업(IOPS)으로 측정된 테이블 쓰기 및 읽기 용량은 테이블 수준에서 적용되므로 각 테넌트에 대해 개별적인 조정 정책을 만들 수 있습니다.

이 모델의 단점은 운영 및 관리 측면에서 더욱 드러나는 경향이 있습니다. 분명한 것은 이 접근 방식을 통해 사용자의 운영 보기는 테넌트 중심 컨텍스트에서 정보를 필터링하고 게시하기 위한 테넌트 테이블 이름 지정 체계에 대해 인식할 필요가 있습니다. 또한 해당 접근 방식에는 이러한 테이블과 상호 작용해야 하는 모든 코드에 대한 간접 계층이 추가됩니다. DynamoDB 테이블과 상호 작용하기 위해서는 테넌트 컨텍스트를 삽입하여 각 요청을 적절한 테넌트 테이블에 매핑해야 합니다.

마이크로서비스 기반 아키텍처를 도입하는 SaaS 공급자에게는 또 다른 고려 사항이 있습니다. 마이크로서비스를 통해 팀은 일반적으로 스토리지 책임을 개별 서비스에 분산시킵니다. 각 서비스는 데이터를 저장하고 관리하는 방법을 자유롭게 결정할 수 있습니다. 이렇게 하면 DynamoDB에 대한 격리 과정이 복잡해질 수 있으므로 각 서비스의 요구를 수용하도록 테이블의 개체군을 확장해야 합니다. 또한 각 서비스에 대해 각각의 테이블이 서비스에 대한 바인딩을 파악하는 다른 차원의 범위 설정 방식이 추가됩니다.

이러한 문제점 중 일부를 상쇄하고 DynamoDB 모범 사례에 더욱 부합하게 하려면 *모든* 테넌트 데이터에 대해 단일 테이블을 사용하는 것을 고려하십시오. 이 접근 방식은 몇 가지 효율성을 제공하며 솔루션의 프로비저닝, 관리 및 마이그레이션 프로파일을 단순화합니다.

대부분의 경우 별도의 DynamoDB 테이블 및 IAM 정책을 사용하여 테넌트 데이터를 격리하면 사일로(silo) 모델의 요구가 해결됩니다. 유일한 다른 옵션은 앞서 설명한 연결 계정 사일로(silo) 모델을 고려하는 것입니다. 그러나 이전에 설명한 것처럼 연결 계정 격리 모델에는 추가 제한 사항 및 고려 사항이 있습니다.

브리지(bridge) 모델

DynamoDB의 경우 브리지(bridge) 모델과 사일로(silo) 모델 사이의 경계가 불명확합니다. 본질적으로 브리지(bridge) 모델을 사용하는 목표가 각 클라이언트에 대해 일회성 스키마 변형을 가진 단일 계정을 갖는 것이라면 앞서 설명한 사일로(silo) 모델을 사용하여 이 목표를 달성할 수 있습니다.

브리지(bridge) 모델의 경우 사일로(silo) 모델을 통해 설명한 격리 요건 중 일부를 줄일 수 있는지 여부에 대해서만 확인하면 됩니다. 테이블 수준 IAM 정책을 도입하지 않는 방식으로 이를 달성할 수 있습니다. 테넌트가 완전한 격리를 요구하지 않는다고 가정할 때, IAM 정책을 제거하면 프로비저닝 계획이 단순화될 수 있다고 분명하게 이야기할 수 있습니다. 하지만 브리지(bridge) 모델에서도 격리에 대한 장점이 있습니다. 따라서 IAM 격리를 포기하는 것이 적합할 수는 있지만 교차 테넌트 액세스를 제한할 수 있는 구성 요소와 정책을 활용하는 것도 SaaS에 대한 좋은 사례입니다.

풀(pool) 모델

DynamoDB에서 풀(pool) 모델을 구현하려면 먼저 서비스가 데이터를 관리하는 방법을 고려해야 합니다. DynamoDB에 데이터가 저장되면 서비스는 지속적으로 데이터를 평가하고 데이터를 분할하여 규모를 조정해야 합니다. 또한 데이터 프로파일이 동일하게 배포되는 경우 이 기본 파티셔닝 체계를 사용하여 SaaS 테넌트의 성능 및 비용 프로파일을 최적화할 수 있습니다.

여기에서 문제는 다중 테넌트 SaaS 환경의 데이터가 일반적으로 균일하게 배포되지 않는다는 것입니다. SaaS 테넌트는 모든 형태와 크기로 제공되므로 데이터는 전혀 균일하지 않습니다. SaaS 공급업체가 데이터 공간의 가장 큰 부분을 사용하는 소수의 테넌트로 작업하게 되는 것은 아주 흔한 일입니다.

이러한 점을 알면 DynamoDB에 풀(pool) 모델을 구현할 때 문제가 발생하는 것을 확인할 수 있습니다. 테넌트 식별자를 DynamoDB 파티션 키에 매핑하는 경우 파티션 "핫스폿"도 만든다는 사실을 빠르게 파악할 수 있습니다. DynamoDB가 효과적으로 데이터를 분할하는 방법을 저해할 매우 큰 테넌트가 하나 있다고 상상해 보십시오. 이러한 핫스폿은 솔루션의 비용 및 성능에 영향을 줄 수 있습니다. 최적화되지 않은 키 배포를 통해 핫 파티션의 영향을 상쇄하려면 IOPS를 높여야 합니다. IOPS가 높아지면 솔루션 비용도 많이 들게 됩니다.

이 문제를 해결하려면 테넌트 데이터의 배포를 더욱 효과적으로 제어할 수 있는 메커니즘을 도입해야 합니다. 데이터를 분할하기 위해 단일 테넌트 식별자에 의존하지 않는 접근 방식이 필요합니다. 이러한 요소는 모두 하나의 경로로 이어집니다. 즉, 각 테넌트를 여러 파티션 키와 연결하는 보조 샤딩 모델을 만들어야 하는 것입니다.

이 같은 솔루션을 어떻게 실현할 수 있는지 한 가지 예를 살펴보겠습니다. 먼저 해당 DynamoDB 파티션 키에 대한 테넌트의 매핑을 캡처하고 관리하기 위해서는 "테넌트 조회 테이블"이라고 하는 별도의 테이블이 필요합니다. 그림 6은 테넌트 조회 테이블을 구성할 수 있는 방법의 예를 나타냅니다.

Partition Key	Attributes	

TenantID
Tenant1
	CustomerTable
	{
	ShardCount: 4,
	ShardSize: [4, 9, 4, 5],
	ShardIds: [“93”, “932”, “21”, “736”]
	}
		AccountTable
		{
		ShardCount: 4,
		ShardSize: [3, 4, 4, 5],
		ShardIds: [“43”, “19”, “971”, “85”]
		}

TenantID
Tenant2
	CustomerTable
	{
	ShardCount: 2,
	ShardSize: [3, 2],
	ShardIds: [“221”, “538”]
	}
		AccountTable
		{
		ShardCount: 3,
		ShardSize: [5, 3, 5],
		ShardIds: [“61”, “216”, “492”]
		}

그림 6: 테넌트 조회 테이블 소개

이 테이블에는 두 개의 테넌트에 대한 매핑이 포함되어 있습니다. 이러한 테넌트와 관련된 항목에는 테넌트와 관련된 각 테이블의 샤딩 정보를 포함하는 특성이 있습니다. 여기에서 테넌트는 고객 및 계정 테이블에 대한 샤딩 정보를 모두 가지고 있습니다. 또한 각 테넌트-테이블 조합에 대해 테이블에 대한 현재 샤딩 프로파일을 나타내는 세 가지 정보가 있다는 점을 알 수 있습니다. 이러한 정보는 다음과 같습니다.

- **ShardCount.** 해당 테이블과 연결된 샤드의 수 표시.
- **ShardSize.** 각 샤드의 현재 크기
- **ShardIds.** 테넌트(테이블용)에 매핑된 파티션 키의 목록.

이 메커니즘을 사용하면 각 테이블에 대한 데이터 배포 방식을 제어할 수 있습니다. 조회 테이블의 간접 참조 기능을 사용하면 저장 중인 데이터의 양에 따라 테넌트의 샤딩 체계를 동적으로 조정할 수 있습니다. 특히 데이터 공간이 큰 테넌트에게는 더 많은 샤드가 주어집니다. 이 모델은 테이블 단위로 샤딩을 구성하기 때문에 테넌트의 데이터 요구 사항을 특정 샤딩 구성에 매핑하는 것보다 훨씬 세부적으로 제어할 수 있습니다. 이를 통해 테넌트의 데이터 프로파일에 종종

나타나는 자연적인 변형을 파티셔닝에 맞게 효과적으로 조정할 수 있습니다.

테넌트 조회 테이블은 테넌트 데이터 배포를 해결할 수 있는 방법을 제공하지만 여기에 따로 비용이 들지 않습니다. 이제 이 모델은 간접 참조 수준을 도입하며 사용자는 이를 솔루션의 데이터 액세스 계층에서 해결해야 합니다. 테넌트 식별자를 사용하여 데이터에 직접 액세스하는 대신 먼저 해당 테넌트의 샤드 매핑을 참조하고 해당 식별자의 조합을 사용하여 테넌트 데이터에 액세스합니다. 그림 7의 샘플 고객 테이블은 이 모델에서 데이터가 어떻게 표시되는지 보여 줍니다.

Partition Key	Attributes	
ShardID 93	**CustomerID** 4923000093	**Name** Bob Jones
ShardID 221	**CustomerID** 9830193911	**Name** Jane Thomas
ShardID 21	**CustomerID** 3492098u72	**Name** Sally Smith
ShardID 932	**CustomerID** 1158304894	**Name** Randy Hanson
ShardID 93	**CustomerID** 8194922299	**Name** Wendy Wilkerson
ShardID 538	**CustomerID** 4800021941	**Name** Henry Hanks
ShardID 932	**CustomerID** 7918499931	**Name** Mary Young
ShardID 736	**CustomerID** 5939202749	**Name** Lisa Franks

그림 7: 샤드 ID가 있는 고객 테이블

이 예에서 `ShardID`는 그림 6에 나온 테이블에서 직접 매핑되는 것입니다. 이 테넌트 조회 테이블에는 고객 테이블에 대한 두 개의 개별적인 샤드 식별자 목록이 포함되어 있는데, 하나는 `Tenant1`용이고 다른 하나는 `Tenant2`용입니다. 이러한 샤드 식별자는 이 샘플 고객 테이블에 표시된 값과 직접적인 연관성이 있습니다. 실제 테넌트 식별자는 이 고객 테이블에 절대 표시되지 않습니다.

샤드 배포 관리

이 모델의 메커니즘이 특히 복잡하지는 않습니다. 데이터를 효율적으로 배포하는 전략을 구현하는 방법에 대해 생각할 때 더욱 흥미로운 문제가 발생합니다. 테넌트가 추가적인 샤드를 필요로 할 때 어떤 방식으로 감지합니까? 이 프로세스를 자동화하기 위해 수집할 수 있는 측정치 및 기준은 무엇입니까?

데이터 및 도메인의 특성이 데이터 프로파일에 어떠한 영향을 줍니까? 모든 솔루션에 대한 이러한 질문을 보편적으로 해결하는 단 하나의 접근 방식은 없습니다. 일부 SaaS 조직은 고객의 통찰력을 토대로 하여 수동으로 이 문제를 조정합니다. 자체적인 접근 방식으로 안내하는 더 자연스러운 기준을 가진 조직도 있습니다.

여기에서 설명하는 접근 방식은 데이터 배포를 처리하기 위해 선택할 수 있는 한 가지 방법입니다. 궁극적으로, 여기에서 설명하는 원칙을 혼합하여 사용자 환경의 요구에 가장 부합하는 방법을 찾게 될 것입니다. 풀(pool) 모델을 도입할 경우 DynamoDB가 데이터를 분할하는 방법을 알고 있어야 한다는 것이 핵심입니다. 데이터가 어떻게 배포될지 고려하지 않고 맹목적으로 데이터를 이동하면 SaaS 솔루션의 성능 및 비용 프로파일이 약화될 수 있습니다.

동적으로 IOPS 최적화

SaaS 환경의 IOPS 요건은 관리하기 어려울 수 있습니다. 테넌트가 시스템에 발생시키는 부하 정도는 크게 다를 수 있습니다. IOPS를 최악의 경우로 설정하면 최대 수준으로 인해 실제 부하를 기반으로 비용을 최적화하려는 의도가 약화됩니다.

대신, 테이블의 IOPS가 애플리케이션의 부하 프로파일을 바탕으로 실시간으로 조정되는 동적 모델을 구현하는 것을 고려하십시오. Dynamic DynamoDB 는 이 문제를 해결하는 데 사용할 수 있는 구성 가능한 오픈 소스 솔루션 중 하나입니다.[1]

다양한 환경 지원

DynamoDB에 대해 설명된 전략에 대해 생각해 보면서 다양한 환경(QA, 개발, 생산 등)이 있는 상태에서 이들 모델이 각각 어떻게 실현될 것인지 고려하십시오. 다양한 환경에 대한 필요성은 AWS에서 각 스토리지 전략을 구분하기 위해 환경을 더욱 분할하는 방법에 영향을 줍니다. 예를 들어 브리지(bridge) 및 풀(pool) 모델을 사용하면 테이블 이름에 한정자를 추가하여 환경 컨텍스트를

제공할 수 있습니다. 이는 테이블 이름의 프로비저닝 및 실행 시간 해결을 고려해야 하는 잘못된 방향으로 이끌기도 합니다.

마이그레이션 효율성

스키마가 적은 DynamoDB의 특성은 SaaS 공급자에게 실질적인 이점을 제공하여 사용자가 애플리케이션에 업데이트를 적용하고 새 테이블이나 복제를 도입하지 않고도 테넌트 데이터를 마이그레이션할 수 있게 해 줍니다. DynamoDB는 SaaS 버전 간에 테넌트를 마이그레이션하는 프로세스를 단순화하고 SaaS 솔루션의 최신 버전에서 애자일 방식 테넌트를 동시에 호스팅하는 동안 다른 테넌트가 이전 버전을 계속 사용할 수 있게 합니다.

트레이드오프 비교

각 모델에게는 비즈니스 요구에 가장 부합하는 모델을 결정할 때 고려해야 할 트레이드오프가 있습니다. 사일로(silo) 패턴은 매력적으로 보일 수 있지만 프로비저닝 및 관리에 있어서 솔루션의 민첩성을 저해하는 복잡성이 추가됩니다. 개별적인 환경 지원과 고유한 테이블 그룹 생성은 자동화된 배포의 복잡성에 영향을 줄 것이 분명합니다. 브리지(bridge)는 DynamoDB에서 사일로(silo) 모델의 작은 변형을 보여 줍니다. 이와 같이 사일로(silo) 모델에 대해 발견한 대부분의 특징을 반영하고 있습니다.

DynamoDB에서 풀(pool) 모델은 몇 가지 중요한 이점을 제공합니다. 통합된 데이터 공간은 프로비저닝, 마이그레이션, 관리 및 모니터링 환경을 단순화합니다. 또한 이를 통해 사용자는 교차 테넌트 기준으로 읽기 및 쓰기 IOPS를 조정하여 사용 및 테넌트 환경을 최적화하는 다중 테넌트 접근 방식을 취할 수 있습니다. 이를 통해 성능 문제에 더욱 폭넓게 대응하고 비용을 최소화할 수 있는 기회를 가질 수 있습니다. 이러한 요소는 풀(pool) 모델을 SaaS 조직에 매우 적합하게 만드는 경향이 있습니다.

RDS에서의 다중 테넌트

관계형 데이터베이스에서 제공되는 초기 SaaS 시스템이 너무 많은 상황에서 개발자 커뮤니티는 이러한 환경에서 다중 테넌트를 해결하기 위한 몇 가지 공통적인 패턴을 수립했습니다. 실제로 RDS는 사일로(silo), 브리지(bridge) 및 풀(pool) 모델에 더욱 자연스럽게 매핑됩니다.

RDS의 데이터 구조 및 표현은 관리되지 않는 관계형 환경을 확장하는 것과 같습니다. 예를 들어, MySQL에서 사용할 수 있는 기본 메커니즘을 RDS에서도 사용할 수 있습니다. 이로 인해 모든 RDS 버전에서 다중 테넌트를 상대적으로 간단하게 실현할 수 있습니다.

다음 섹션에서는 RDS에서 파티셔닝 모델을 구현하는 데 일반적으로 사용되는 다양한 전략에 대해 설명합니다.

사일로(silo) 모델

AWS에서 다양한 방식으로 사일로(silo) 패턴을 구현할 수 있습니다. 그러나 격리를 구현하기 위한 가장 보편적이고 간단한 접근 방식은 각 테넌트에 대해 별도의 데이터베이스 인스턴스를 만드는 것입니다. 인스턴스를 통해, 완전히 분리된 계정을 프로비저닝하는 데 대한 오버헤드 없이 고객의 규정 준수 요건을 충족시키는 수준으로 분리를 구현할 수 있습니다.

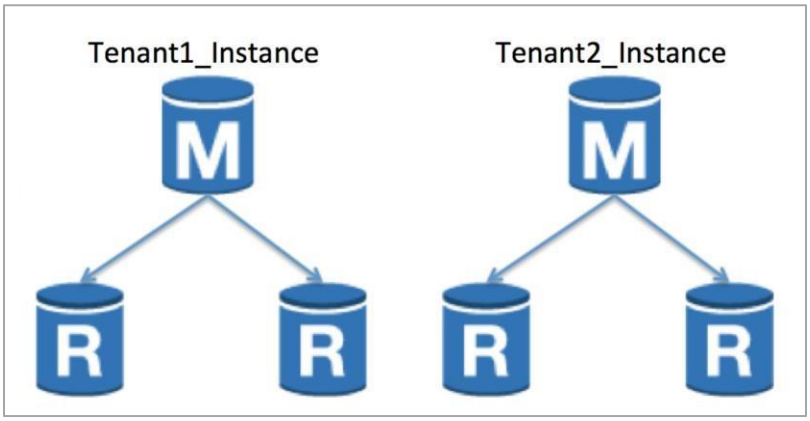

그림 8: 사일로(silo) 형식의 RDS 인스턴스

그림 8은 RDS를 기반으로 실현될 수 있는 기본적인 사일로(silo) 모델을 보여 줍니다. 여기에서는 각 테넌트에 대해 두 개의 개별적인 인스턴스가 프로비저닝됩니다.

이 다이어그램은 각 테넌트 인스턴스에 대한 마스터 데이터베이스와 두 개의 읽기 전용 복제본을 보여 줍니다. 이 접근 방식을 사용하여 각 테넌트에 대해 최적화된 고가용성 전략을 설정하고 구성하는 방법을 강조하기 위한 선택적 개념이라고 할 수 있습니다.

브리지(bridge) 모델

RDS에서 브리지(bridge) 모델을 구현하는 것은 모든 스토리지 모델에 대한 주제와 동일한 측면이 있습니다. 기본 접근 방식은 모든 테넌트에 대해 단일 인스턴스를 활용하면서 해당 데이터베이스 내의 각 테넌트에 대해 개별적인 표현을 생성하는 것입니다. 여기에는 각 테이블을 주어진 테넌트에 매핑하기 위해 프로비저닝 및 실행 시간 테이블 분석을 할 필요성이 생깁니다.

브리지(bridge) 모델은 테넌트 데이터를 마이그레이션할 때 서로 다른 스키마 및 약간의 유연성이 있는 테넌트를 보유할 수 있는 기회를 제공합니다. 예를 들어, 정해진 시간에 서로 다른 제품 버전을 실행하는 다양한 테넌트를 보유할 수 있으며 테넌트 단위로 스키마 변경 사항을 점진적으로 마이그레이션할 수 있습니다.

그림 9는 RDS에서 브리지(bridge) 모델을 구현할 수 있는 한 가지 방식의 예를 제공합니다. 이 다이어그램에는 `Tenant1` 및 `Tenant2`에 대한 개별적인 고객 테이블을 포함하는 단일 RDS 데이터베이스 인스턴스가 있습니다.

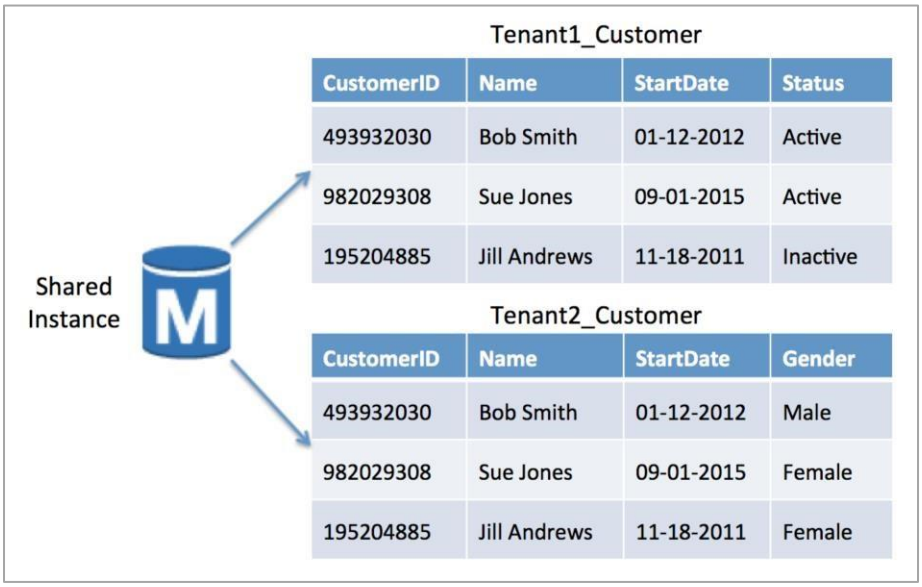

그림 9: RDS에서의 브리지(bridge) 모델의 예

이 예에서는 테넌트 수준에서 스키마 변형을 갖는 기능을 강조합니다. `Tenant1`의 스키마에는 `Status` 열이 있으며, `Tenant2`는 이 열을 제거하고 `Gender` 열로 바꿉니다.

여기에서 또 다른 옵션은 인스턴스 내의 각 테넌트에 대해 별도의 데이터베이스 개념을 도입하는 것입니다. 용어는 RDS의 각 버전에 따라 다릅니다. 일부 RDS 스토리지 컨테이너는 이것을 데이터베이스라고 지칭하며 다른 버전은 스키마로 표시합니다.

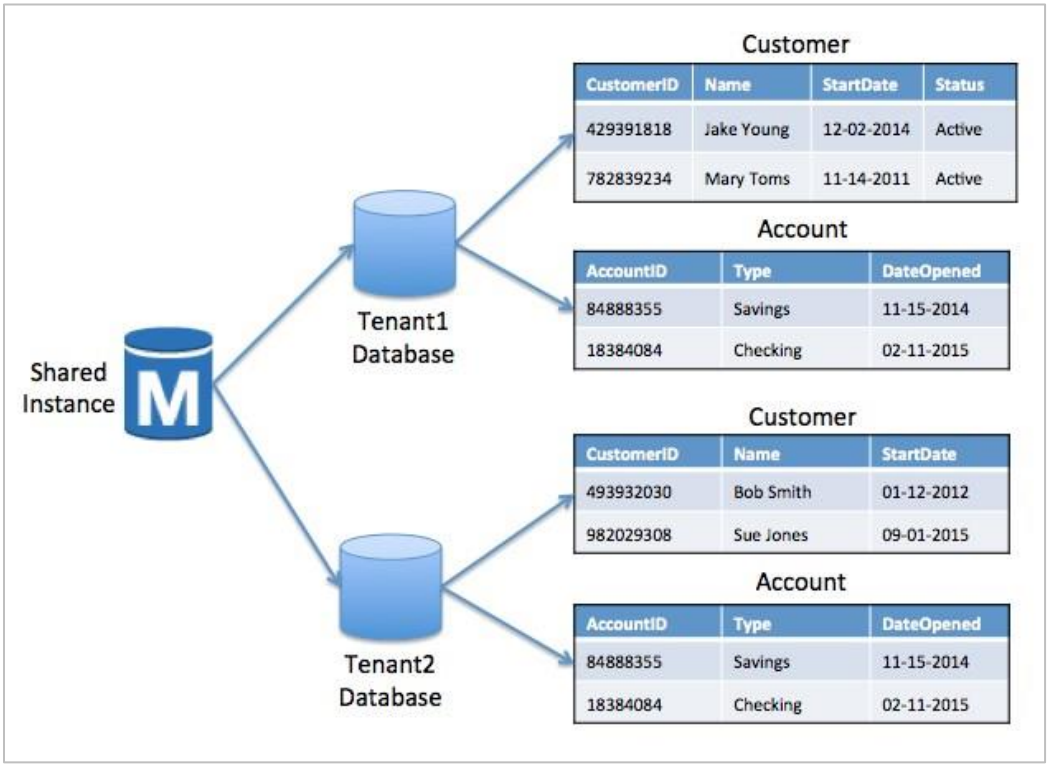

그림 10: 별도의 테이블/스키마가 있는 RDS 브리지

그림 10은 이 대체 브리지(bridge) 모델을 설명하는 그림입니다. 각 테넌트에 대한 데이터베이스를 만들었으며 이제 테넌트는 자체 테이블 모음을 가지고 있습니다. 일부 SaaS 조직의 경우 이는 테넌트 데이터를 더욱 자세히 관리하도록 지원하여 개별 테이블에 이름 지정을 적용해야 할 필요성을 방지합니다.

이 모델은 매력적이지만 RDS의 모든 버전에 가장 적합하지는 않을 수 있습니다. 일부 RDS 컨테이너는 인스턴스에 대해 만들 수 있는 데이터베이스/스키마 수를 제한합니다. 예를 들어 SQL Server 컨테이너는 인스턴스당 30개의 데이터베이스만 허용하는데, 이는 대부분의 SaaS 환경에서 허용될 수 없을 가능성이 큽니다. 브리지(bridge) 모델은 테넌트에서 테넌트 사이의 변형을 허용하지만, 일반적으로 사용자는 스키마 변경을 제한하려는 정책을 도입해야 한다는 점을 알고 있어야 합니다. 스키마 변경 사항을 도입할 때마다 가동 중단 시간 없이 SaaS 테넌트를 새 모델에 성공적으로 마이그레이션하는 문제를 해결할 수 있습니다.

따라서 이 모델은 이러한 마이그레이션을 단순화하지만 일회성 테넌트 스키마 또는 테넌트의 데이터 표현에 대해 정기적으로 변경하는 것은 권장하지 않습니다.

풀(pool) 모델

RDS용 풀(pool) 모델은 기존의 관계형 인덱싱 스키마를 사용하여 테넌트 데이터를 분할합니다. 모든 테넌트 데이터를 공유 인프라 모델로 이동하는 과정의 일부로 사용자는 단일 RDS 인스턴스에 테넌트 데이터를 저장하고 테넌트는 공통 테이블을 공유합니다. 이러한 테이블은 각 테넌트의 데이터를 액세스하고 관리하는 데 사용되는 고유한 테넌트 식별자로 인덱싱됩니다.

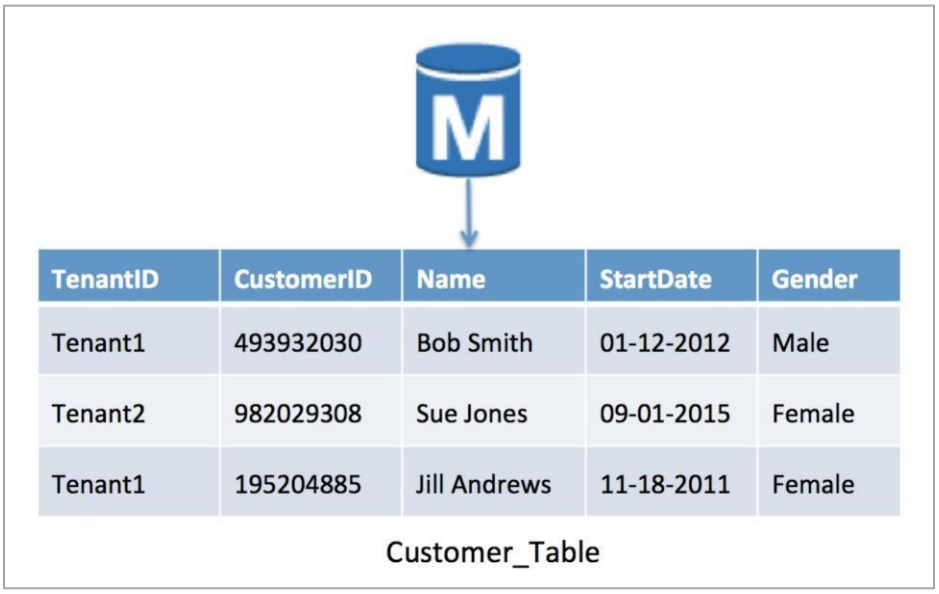

그림 11 - 공유 스키마가 있는 RDS 풀(pool) 모델

그림 11은 작업에 대한 풀(pool) 모델의 예를 보여 줍니다. 여기에서 *하나의* 고객 테이블이 있는 단일 RDS 인스턴스는 모든 애플리케이션의 테넌트에 대한 데이터를 보유하고 있습니다. RDS는 RDBMS이므로 모든 테넌트는 동일한 스키마 버전을 사용해야 합니다. RDS는 각 테넌트가 단일 테이블 내에서 고유한 스키마를 가지도록 허용하는 유연한 스키마를 갖춘 DynamoDB와는 다릅니다.

단일 인스턴스 제한 고려

지금까지 설명했던 많은 모델은 데이터를 단일 인스턴스에 저장하고 해당 인스턴스 내에서 데이터를 분할하는 데 크게 집중하고 있습니다. SaaS 환경의 크기 및 성능 요구에 따라, 단일 인스턴스를 사용하는 것은

테넌트 데이터의 프로파일에 적합하지 않을 수 있습니다. RDS는 단일 인스턴스에 저장할 수 있는 데이터의 양에 대한 제한이 있습니다. 다음은 제한에 대한 분석입니다.

- MySQL, MariaDB, Oracle, PostgreSQL – **6TB**
- SQL Server – **4TB**
- Aurora – **64TB**

또한 단일 인스턴스로 인해 리소스 경합 문제(CPU, 메모리, I/O)가 발생합니다.

단일 인스턴스의 사용이 현실적으로 어려운 시나리오에서는 테넌트 데이터가 여러 인스턴스에 배포되는 샤딩 체계를 도입하는 것으로 자연스럽게 확장할 수 있습니다. 이 접근 방식을 사용하면 샤딩된 인스턴스의 작은 모음부터 시작합니다. 그런 다음 테넌트 데이터의 프로파일을 지속적으로 관찰하고 인스턴스 수를 확장하여 단일 인스턴스가 한도에 이르거나 병목 현상이 발생하지 않게 하십시오.

트레이드오프 비교

RDS를 사용하는 것의 트레이드오프는 꽤 단순합니다. 민첩성을 위한 트레이딩 관리 및 프로비저닝의 복잡성에 관한 주제가 가장 주된 것입니다. 전반적으로 RDS에서 사일로(silo) 모델을 사용하면 프로비저닝 자동화의 문제점이 줄어들 가능성이 큽니다. 그러나 풀(pool) 모델과 관련된 비용 및 관리 효율성은 종종 주목할만한 요소입니다. 이러한 모델이 지속적인 제공 환경에 어떻게 부합할 것인지 여부를 고려할 때 특히 중요합니다.

Amazon RedShift에서의 다중 테넌트

Amazon Redshift는 다중 테넌트에 대한 생각에 영향을 줄 수 있는 추가적인 단서를 제공합니다. Amazon Redshift는 대규모 데이터 웨어하우스를 수용할 수 있는 고성능 클러스터를 구축하는 데 중점을 둡니다. 또한 Amazon Redshift는 각 클러스터 내에서 생성할 수 있는 구성 요소에 몇 가지 제한을 두었습니다. 다음 제한을 고려하십시오.

- 클러스터당 60개의 데이터베이스
- 데이터베이스당 256개의 스키마
- 데이터베이스당 500개의 동시 연결 수

- 50개의 동시 쿼리
- 클러스터에 액세스하면 클러스터의 모든 데이터베이스에 액세스할 수 있습니다.

이러한 제한이 Amazon Redshift에 전달되는 규모와 성능에 어떤 영향을 미치는지 상상해 볼 수 있습니다. 또한 Amazon Redshift를 통해 이러한 제한이 다중 테넌트에 대한 접근 방식에 어떤 영향을 미칠 수 있는지 확인할 수도 있습니다. 보통 규모의 테넌트 수를 목표로 하는 경우 이러한 제한은 솔루션에 거의 영향을 미치지 않을 수 있습니다. 그러나 많은 수의 테넌트를 목표로 하고 있다면 전체적인 전략에 대해 이러한 제한을 고려해야 합니다.

다음 섹션에서는 Amazon Redshift에서 각 다중 테넌트 스토리지 모델을 실현하는 데 일반적으로 사용되는 전략에 대해 강조합니다.

사일로(silo) 모델

Amazon Redshift에서 테넌트의 진정한 사일로(silo) 모델 격리를 구현하려면 각 테넌트에 대해 개별적인 클러스터를 프로비저닝해야 합니다. 일반적으로 고객이 교차 테넌트의 액세스로부터 데이터를 성공적으로 격리하는 데 필요한 테넌트 간 경계를 클러스터를 통해 명확하게 정의할 수 있습니다. 이 접근 방식은 Amazon RedShift의 정상적인 보안 메커니즘을 가장 잘 활용하므로 사용자는 IAM 정책과 데이터베이스 권한을 함께 사용하여 클러스터에 대한 테넌트의 액세스를 제어하고 제한할 수 있습니다. IAM은 전반적인 클러스터 관리를 제어하며 데이터베이스 권한은 클러스터 내의 데이터에 대한 액세스를 제어하는 데 사용됩니다.

사일로(silo) 모델은 각 테넌트에 대해 조정된 환경을 만들 수 있는 기회를 제공합니다. Amazon Redshift를 사용하여 클러스터의 노드 수와 유형을 구성할 수 있으므로 각 개별 테넌트의 부하 프로파일을 대상으로 하는 환경을 만들 수 있습니다. 또한 비용 최적화를 위한 전략으로 사용할 수도 있습니다.

다른 사일로(silo) 모델에서 확인했다시피 이 모델의 문제점은 각 테넌트의 클러스터가 온보딩 프로세스의 일부로 프로비저닝되어야 한다는 것입니다. 이 프로세스를 자동화하고 프로비저닝 프로세스와 관련된 추가 시간과 오버헤드를

처리하는 것에는 배포 규모에 대한 복잡성이 부가됩니다. 또한 새로운 테넌트가 할당될 수 있는 속도에도 약간의 영향을 미칩니다.

브리지(bridge) 모델

브리지(bridge) 모델은 Amazon Redshift에 적합한 매핑을 갖추고 있지 않습니다. 사용자는 기술적으로 각 테넌트에 대한 별도의 스키마를 만들 수 있습니다. 하지만 Amazon Redshift의 256개 스키마 제한 문제가 발생할 가능성이 큽니다. 상당한 수의 테넌트가 있는 환경에서 간단하게 확장되지는 않습니다. 보안 역시 브리지(bridge) 모델의 Amazon Redshift에 대한 문제입니다. Amazon Redshift 클러스터의 사용자로 권한이 부여되면 해당 클러스터 내의 모든 데이터베이스에 대한 액세스 권한이 부여됩니다. 이는 세분화된 액세스 제어를 실행하는 데 대한 책임을 SaaS 애플리케이션에게 전가합니다.

브리지(bridge) 모델의 사용 동기와 이러한 기술적 고려 사항을 감안할 때 대부분의 SaaS 공급자에게는 Amazon Redshift에서 이러한 접근 방식을 사용하는 것이 현실적으로 어렵습니다. 솔루션에 대한 제한을 관리할 수 있는 경우에도 격리 프로파일은 고객에게 허용되지 않을 가능성이 높습니다. 궁극적으로 가장 좋은 방법은 격리가 필요한 모든 테넌트에 대해 사일로(silo) 모델을 사용하는 것입니다.

풀(pool) 모델

Amazon Redshift에서 풀(pool) 모델을 구축하는 것은 지금까지 논의했던 다른 스토리지 모델과 매우 유사합니다. 기본적인 아이디어는 데이터베이스 및 테이블을 공유하는 단일 Amazon Redshift 클러스터에 모든 테넌트에 대한 데이터를 저장하는 것입니다. 이 접근 방식에서는 테넌트에 대한 데이터가 고유한 테넌트 식별자를 나타내는 열의 도입을 통해 분할됩니다.

이 접근 방식은 다른 풀(pool) 모델에서 확인한 대부분의 장점을 제공합니다. 단일 Amazon Redshift 클러스터에 모든 테넌트 데이터를 저장하면 전반적인 관리, 모니터링 및 민첩성이 개선됩니다.

동시 연결의 제한은 Amazon Redshift에서 풀(pool) 모델을 구현하는 데 어려움을 더하는 영역입니다. 최대 동시 연결 수를 500개로 제한하면 많은 다중 테넌트

SaaS 환경이 이러한 제한을 빠른 속도로 초과할 수 있습니다. 이 문제는 풀(pool) 모델을 경합에서 배제하지는 않습니다. 대신 SaaS 개발자에게 이러한 연결이 사용 및 해제되는 방법과 시기를 관리하는 데 사용되는 효과적인 전략을 수립할 더 큰 책임을 부여합니다.

연결 관리를 해결하는 몇 가지 일반적인 방법이 있습니다. 개발자는 종종 클라이언트 기반 캐싱을 활용하여 Amazon Redshift에 대한 실제 연결의 필요성을 제한합니다. 연결 풀링도 이 모델에 적용될 수 있습니다. 개발자는 Amazon Redshift 연결 제한을 초과하지 않고 애플리케이션의 데이터 액세스 패턴을 효과적으로 충족시킬 수 있는 전략을 선택해야 합니다.

풀(pool) 모델을 도입한다는 것은 공유 인프라에서 운영할 때마다 발생하는 일반적인 문제에 주의를 기울이는 것을 의미합니다. 예를 들어, 데이터 보안에는 교차 테넌트 액세스를 제한하는 애플리케이션 수준 정책이 필요합니다. 또한 하나의 테넌트가 다른 테넌트의 환경을 저하하지 않도록 환경 성능을 지속적으로 조정하고 구체화해야 할 것입니다.

민첩성(Agility)을 고려

다중 테넌트 스토리지 옵션의 매트릭스는 매우 까다로울 수 있습니다. 유연성, 격리 및 관리성의 조합을 가장 적절하게 나타내는 솔루션을 식별하는 것도 어려울 수 있습니다. 모든 옵션을 고려하는 것이 중요하지만, 다중 테넌트 스토리지에 대해 민첩성을 지속적으로 고려하는 것도 매우 중요합니다. 솔루션이 제공하는 민첩성의 정도는 보통 SaaS 조직의 성공에 엄청난 영향을 미칩니다.

사용자가 선택하는 스토리지 기술 및 격리 모델은 새 기능을 쉽게 배포할 수 있는 능력에 직접적인 영향을 줍니다. 구조의 형태 및 데이터의 콘텐츠는 새로운 기능을 지원하기 위해 종종 변경되는 경우가 있으며 이는 곧 기본 스토리지 모델이 필요한 가동 중단 시간 없이 이러한 변경 사항을 수용해야 한다는 것을 의미합니다. 원활한 마이그레이션을 지원하는 것과 관련하여 각 격리 모델에는 장단점이 있습니다. 옵션을 고려할 때 이러한 요소에도 적절한 주의를 기울이십시오.

사일로(silo), 브리지(bridge) 및 풀(pool) 모델이 모두 민첩성이라는 특징을 갖고 있긴 하지만 사용자는 민첩성을 최대한 유지하기 위해 공통적인 원칙을 적용할

수 있습니다. 핵심 원칙은 테넌트 데이터에 대한 일회성 변형을 최소화하는 것으로, 때론 이러한 필요성이 약화되긴 하지만 분명하게 필요한 것입니다. 예를 들어 사일로(silo) 및 브리지(bridge) 모델은 스토리지 변형으로 이어질 수 있습니다. 이는 하나의 자동화된 이벤트로서 모든 SaaS 고객을 위해 새로운 기능을 개발하는 기능을 복잡하게 만들 수 있습니다. 팀은 종종 자동화 및 지속적인 배포를 사용하여 다중 테넌트 스토리지 전략에 의해 발생하는 문제를 제한합니다.

스토리지 전략을 확립하면서도 스토리지 요구 사항이 계속 변화한다는 사실을 예상하고 이를 받아들이십시오. SaaS 고객의 요구는 항상 변화하며 오늘 선택한 스토리지 모델이 내일은 적합하지 않을 수도 있습니다. 또한 AWS는 스토리지에 대한 접근 방식을 향상시키기 위해 새로운 기회를 제공할 수 있는 새로운 기능과 서비스를 계속해서 도입하고 있습니다.

결론

SaaS 고객의 스토리지 요구 사항은 간단하지 않습니다. 비즈니스 영역, 고객 및 기존 고려 사항이 비즈니스의 요구에 가장 부합하는 다중 테넌트 스토리지 옵션의 조합을 결정하는 방법에 영향을 미친다는 점이야말로 SaaS의 현실입니다.

보편적으로 모든 환경에 잘 맞는 단일 전략은 없지만 일부 모델이 SaaS 제공 모델의 핵심 원칙과 더욱 적합하다는 점은 분명한 사실입니다. 일반적으로 모든 AWS 스토리지 기술에서 스토리지에 대한 풀(pool) 기반 접근 방식은 다중 테넌트 환경 관리 및 운영에 대한 통합 접근 방식의 필요성과 부합합니다. 모든 테넌트가 하나의 공유 리포지토리 및 표현을 갖게 하면 접근 방식의 운영 및 배포 규모를 간소화하고 통합하여 상태 및 성능에 대한 교차 테넌트 보기를 사용할 수 있습니다.

사일로(silo) 및 브리지(bridge) 모델은 분명한 특징을 가지고 있으며 일부 SaaS 공급자에게 있어 절대적으로 필요합니다. 여기서 중요한 점은 이 방향을 고집하면 민첩성 문제가 더 복잡해질 수 있다는 것입니다. 일부 AWS 스토리지 기술은 격리된 테넌트 스토리지 체계를 더 효과적으로 지원할 수 있습니다. 예를 들, RDS에서 사일로(silo) 모델을 구축하는 것은 DynamoDB에서 구축하는 것보다 복잡하지 않습니다. 일반적으로 연결 계정을 파티셔닝 모델로 사용하면 더 많은 프로비저닝, 관리 및 조정 문제를 처리할 수 있습니다.

다중 테넌트를 달성하는 메커니즘 외에도 각 AWS 스토리지 기술의 프로파일이 다중 테넌트 애플리케이션 기능의 다양한 요구에 어떻게 부합할 수 있는지 생각해 보십시오. 테넌트가 데이터에 어떤 방식으로 액세스해야 할지, 테넌트의 요구를 충족하기 위해 데이터의 형태가 어떻게 변화해야 할지 고려하십시오. 애플리케이션을 자율 서비스로 분해할수록 각 서비스에 대해 별도의 스토리지 전략을 고르고 선택할 수 있는 능력이 높아집니다.

이러한 서비스 및 분할 스키마를 살펴본 후에는 다중 테넌트 스토리지 전략을 선택할 수 있게 안내할 패턴 및 변곡점에 대해 더 깊이 이해해야 합니다. AWS는 SaaS 공급자에게 다양한 다중 테넌트 스토리지 요구를 해결하기 위해 함께 사용할 수 있는 다양한 서비스 및 구성 요소를 제공합니다.

기고자

다음은 이 문서의 작성에 도움을 준 개인 및 조직입니다.

- Tod Golding, 파트너 솔루션스 아키텍트, AWS 파트너 프로그램
- Clinton Ford, 선임 제품 마케팅 관리자, DynamoDB
- Zach Christopherson, 데이터베이스 엔지니어, Amazon Redshift
- Brian Welker, 수석 제품 소유자, RDS MySQL 및 MariaDB

참고 문헌

자세한 내용은 다음을 참조하십시오.

- Amazon RDS
- Amazon DynamoDB
- Amazon Redshift

참고

[1] https://dynamic-dynamodb.readthedocs.io/en/latest/index.html