AWS 백서

SaaS 아키텍처 기초

SaaS 아키텍처 기초: AWS 백서

Table of Contents

SaaS 아키텍처 기초

게시 날짜: 2022년 8월 3일(문서 수정)

서비스형 소프트웨어(SaaS) 모델에서 비즈니스를 운영하는 범위, 목표 및 특성을 정의하기는 어려울 수 있습니다. SaaS를 특성화하는 데 사용되는 용어와 패턴은 그 출처에 따라 다릅니다. 이 문서의 목표는 SaaS의 기본 요소를 더 잘 정의하고 AWS에서 SaaS 시스템을 설계하고 제공할 때 적용되는 패턴, 용어 및 가치 체계를 보다 명확하게 파악하는 것입니다. 목표를 더 광범위하게 잡으면, 고객이 SaaS 전송 모델을 채택할 때 고려해야 할 옵션을 보다 명확하게 파악할 수 있도록 기본 인사이트 모음을 제공하는 것입니다.

이 백서는 SaaS 여정을 시작하는 SaaS 빌더 및 아키텍트뿐만 아니라 핵심 SaaS 개념에 대한 이해를 개선하고자 하는 노련한 빌더를 대상으로 합니다. 이 정보 중 일부는 SaaS 환경에 더 익숙해지고자 하는 SaaS 제품 소유자 및 전략가에게도 유용할 수 있습니다.

소개

서비스형 소프트웨어(SaaS)라는 용어는 비즈니스 및 전송 모델을 설명하는 데 사용됩니다. 그러나 문제는 SaaS가 의미하는 바가 보편적으로 이해되지 않는다는 것입니다.

SaaS의 핵심 요소 중 일부에 대해서는 어느 정도 합의가 이루어졌지만 SaaS가 무엇을 의미하는지에 대해서는 여전히 혼란이 있습니다. 팀에서 SaaS를 보는 방식에 약간의 차이가 있는 것은 당연합니다. 동시에 SaaS 개념과 용어에 대한 명확성이 부족하면 SaaS 전송 모델을 탐색하는 사람들이 혼란을 겪을 수 있습니다.

이 문서는 핵심 SaaS 개념을 설명하는 데 사용되는 용어를 간략하게 설명하는 데 중점을 둡니다. 이러한 개념을 중심으로 사고방식을 공유하면 SaaS 아키텍처의 기본 요소를 명확하게 파악할 수 있어 SaaS 아키텍처 구조를 설명하는 데 필요한 공유 용어를 확보할 수 있습니다. 이는 이러한 주제를 기반으로 하는 추가 콘텐츠를 살펴볼 때 특히 유용합니다.

이 백서는 멀티테넌시의 아키텍처 세부 사항에서 벗어나 SaaS의 기본 개념을 어떻게 정의했는지 살펴봅니다. 또한 이를 통해 조직에서 SaaS 솔루션의 특징과 특성을 더 빠르게 파악할 수 있는 보다 명확한 용어 세트를 제공하는 것이 이상적입니다.

귀사는 Well-Architected입니까?

AWS Well-Architected 프레임워크는 클라우드에서 시스템을 구축할 때 내리는 결정의 장단점을 이해하는 데 도움이 됩니다. 이 프레임워크를 사용하여 클라우드에서 안정적이고 안전하며 효율적이

고 비용 효율적인 시스템을 설계하고 운영하기 위한 아키텍처 모범 사례를 살펴볼 수 있습니다. AWS Management Console에서 무료로 제공되는 AWS Well-Architected Tool를 사용하면 각 요소에 대한 일련의 질문에 답하여, 이러한 모범 사례와 비교하여 워크로드를 검토할 수 있습니다.

SaaS Lens에서 우리는 AWS의 서비스형 소프트웨어(SaaS) 워크로드를 설계하는 모범 사례에 중점을 둡니다.

참조 아키텍처 배포, 다이어그램, 백서 등 클라우드 아키텍처에 대한 더 많은 전문가 지침과 모범 사례를 보려면 AWS 아키텍처 센터를 참조하세요.

SaaS는 비즈니스 모델입니다

SaaS가 의미하는 바를 정의하는 것은 SaaS가 비즈니스 모델이라는 한 가지 핵심 원칙에 동의하는 것에서 시작됩니다. 이는 무엇보다도 SaaS 전송 모델의 채택이 일련의 비즈니스 목표에 의해 직접적으로 주도된다는 것을 의미합니다. 그렇습니다, 이러한 목표 중 일부는 기술이 사용될 것입니다. 하지만 SaaS는 특정 비즈니스 목표를 목표로 하는 사고방식과 모델을 마련하는 것입니다.

SaaS 전송 모델 채택과 관련된 몇 가지 주요 비즈니스 목표를 좀 더 자세히 살펴보겠습니다.

- 민첩성 — 이 용어는 SaaS의 광범위한 목표를 요약합니다. 성공적인 SaaS 회사는 시장, 고객 및 경쟁 역학에 지속적으로 적응할 준비가 되어 있어야 한다는 생각을 바탕으로 구축되었습니다. 훌륭한 SaaS 기업은 새로운 가격 책정 모델, 새로운 시장 부문 및 새로운 고객 요구를 지속적으로 수용하도록 구성되어 있습니다.
- 운영 효율성 — SaaS 기업은 운영 효율성에 의존하여 확장성과 민첩성을 높입니다. 즉, 새로운 기능의 빈번하고 신속한 출시를 촉진하는 운영 입지를 구축하는 데 초점을 맞춘 문화와 도구를 마련해야 합니다. 또한 모든 고객 환경을 종합적으로 관리, 운영 및 배포할 수 있는 단일 통합 환경을 갖추는 것도 의미합니다. 일회성 버전과 사용자 지정을 지원한다는 생각은 이제 사라졌습니다. SaaS 비즈니스는 비즈니스를 성공적으로 성장시키고 확장하기 위한 핵심 요소로 운영 효율성을 중시합니다.
- 원활한 온보딩 — 민첩성을 높이고 성장을 수용하기 위한 일환으로 테넌트 고객 온보딩 프로세스의 마찰을 줄이는 데도 중점을 두어야 합니다. 이는 B2B(기업 간) 및 B2C(기업-고객) 고객에게 보편적으로 적용됩니다. 어떤 부문이나 고객 유형을 지원하든, 여전히 고객을 위해 가치를 창출하는 데 걸리는 시간에 집중해야 합니다. 서비스 중심 모델로 전환하려면 SaaS 비즈니스가 전체 온보딩 라이프사이클의 반복성과 효율성에 특히 중점을 두고 고객 경험의 모든 측면에 집중해야 합니다.
- 혁신 — SaaS로 전환하는 것은 단순히 현재 고객의 요구를 해결하는 것이 아니라 혁신을 가능하게 하는 기본 요소를 마련하는 것입니다. SaaS 모델에서 고객 요구에 반응 및 대응하는 것은 중요합니다. 그러나 이러한 민첩성을 활용하여 고객을 위한 새로운 시장, 기회 및 효율성을 창출할 수 있는 미래의 혁신을 주도하는 것도 중요합니다.
- 시장 응답 — SaaS는 분기별 출시 및 2년 계획이라는 전통적인 개념에서 벗어납니다. 조직이 시장 변화에 거의 실시간으로 대응하고 대응할 수 있는 능력을 제공하려면 민첩성이 필요합니다. SaaS의 조직적, 기술적, 문화적 요소에 대한 투자는 새로운 고객 및 시장 역학을 기반으로 비즈니스 전략을 전환할 수 있는 기회를 창출합니다.
- 성장 — SaaS는 성장 중심의 비즈니스 전략입니다. 민첩성과 효율성을 중심으로 조직의 모든 움직이는 부분을 조정하면 SaaS 조직이 성장 모델을 목표로 삼을 수 있습니다. 즉, SaaS 서비스의 빠른 채택을 수용하고 환영하는 메커니즘을 마련해야 합니다.

각 항목은 비즈니스 성과에 초점을 맞추고 있습니다. SaaS 시스템을 구축하는 데 사용할 수 있는 기술 전략과 패턴은 다양합니다. 하지만 이러한 기술 전략의 어떤 것도 비즈니스 스토리를 바꾸지 못합니다.

조직과 함께 SaaS 채택의 일환으로 달성하고자 하는 것이 무엇인지 물어볼 때 우리는 항상 비즈니스에 초점을 맞춘 논의부터 시작합니다. 기술 선택도 중요하지만, 그것은 이러한 비즈니스 목표의 맥락에서 실현되어야 합니다. 예를 들어 민첩성, 운영 효율성 또는 원활한 온보딩을 달성하지 못한 채 멀티테넌트를 운영하면 SaaS 비즈니스의 성공이 저해될 수 있습니다.

이를 바탕으로 앞에서 설명한 원칙을 준수하는 보다 간결한 SaaS 정의로 공식화해 보겠습니다.

SaaS는 고객과 제공업체의 가치를 극대화하는 마찰이 적은 서비스 중심 모델로 솔루션을 제공할 수 있는 비즈니스 및 소프트웨어 제공 모델입니다. 성장, 범위 및 혁신을 촉진하는 비즈니스 전략의 요소로서 민첩성과 운영 효율성을 활용합니다.

비즈니스 목표와 모든 고객에게 공유된 경험을 제공하는 데 어떻게 의존하는지 확인해야 합니다. SaaS로 전환할 때 가장 큰 비중을 차지하는 것은 기존 소프트웨어 모델의 일부일 수 있는 일회성 사용자 지정에서 벗어나는 것을 의미합니다. 고객에게 전문화를 제공하려는 노력은 일반적으로 SaaS로 달성하려는 핵심 가치에서 멀어집니다.

제품이 아니라 서비스입니다

“서비스” 모델을 채택하는 것은 단순한 마케팅이나 용어 그 이상입니다. 서비스 사고방식을 취하다 보면, 기존의 제품 기반 개발 접근 방식에서 벗어나게 되는 것을 발견하게 될 것입니다. 모든 제품에서 특징과 기능은 확실히 중요하지만 SaaS는 고객이 서비스를 통해 얻게 될 경험에 더 중점을 둡니다.

이것을 무엇을 의미하나요? 서비스 중심 모델에서는 고객이 서비스에 어떻게 적응하는지, 얼마나 빨리 가치를 달성하는지, 고객 요구를 충족하는 기능을 얼마나 빨리 도입할 수 있을지에 대해 더 많이 생각합니다. 서비스 구축, 운영 및 관리 방식과 관련된 세부 정보는 고객이 볼 수 없습니다.

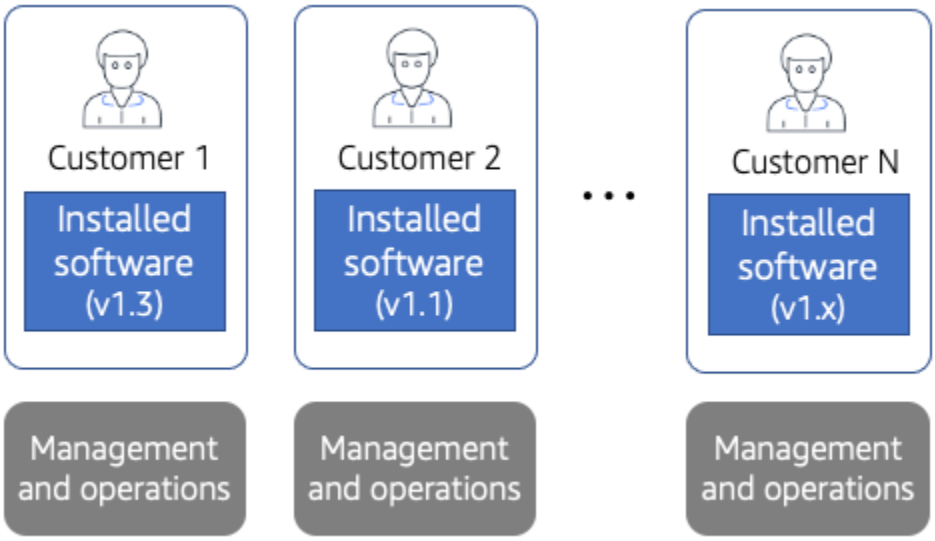
이 모드에서는, 소비할 수 있는 다른 서비스와 마찬가지로 이 SaaS 서비스를 고려합니다. 식당에 있으면 음식에도 신경을 쓰지만 서비스에도 신경을 씁니다. 서버가 테이블로 얼마나 빨리 오는지, 물을 얼마나 자주 채우는지, 음식이 얼마나 빨리 오는지, 이런 것들이 모두 서비스 경험의 척도입니다. 이는 SaaS 서비스 구축에 대한 우리의 생각을 형성해야 하는 사고방식과 가치 체계와 동일합니다.

이 서비스형 모델은 팀과 서비스를 구축하는 방식에 큰 영향을 미칠 것입니다. 작업 백로그로 인해 이제 이러한 경험 속성이 특징 및 기능보다 같거나 더 높은 위치에 놓이게 될 것입니다. 또한 비즈니스에서는 이를 SaaS 서비스의 장기적 성장 및 성공의 기반으로 간주할 것입니다.

최초 동기

SaaS를 이해하기 위해 SaaS 비즈니스를 만들 때 달성하려는 목표에 대한 매우 간단한 개념부터 시작하겠습니다. 가장 좋은 출발점은 기존(비 SaaS) 소프트웨어가 어떻게 생성, 관리 및 운영되었는지 살펴보는 것입니다.

다음 다이어그램은 여러 공급업체가 솔루션을 패키징하고 제공한 방식을 개념적으로 보여줍니다.



소프트웨어 솔루션을 패키징하고 제공하기 위한 클래식 모델

이 다이어그램에서는 고객 환경 모음을 설명했습니다. 이러한 고객은 공급업체의 소프트웨어를 구입한 여러 회사 또는 법인을 나타냅니다. 각 고객은 기본적으로 소프트웨어 제공업체의 제품을 설치한 독립형 환경에서 운영하고 있습니다.

이 모드에서는 각 고객의 설치가 해당 고객 전용의 독립형 환경으로 취급됩니다. 즉, 고객은 스스로를 이러한 환경의 소유자로 여기며 자신의 요구 사항을 지원하는 일회성 사용자 지정 또는 고유한 구성을 요청할 수 있습니다.

이러한 사고방식의 일반적인 부작용 중 하나는 실행 중인 제품 버전을 고객이 제어할 수 있다는 것입니다. 이러한 문제는 여러 가지 이유로 발생할 수 있습니다. 고객은 새 기능에 대해 불안해하거나 새 버전 채택과 관련된 장애에 대해 걱정할 수 있습니다.

이러한 역동성이 소프트웨어 제공업체의 운영 공간에 어떤 영향을 미칠 수 있는지 상상할 수 있을 것입니다. 고객에게 일회성 환경을 더 많이 허용할수록 각 고객의 다양한 구성을 관리, 업데이트 및 지원하기가 더 어려워집니다.

이처럼 일회성 환경이 필요하기 때문에 조직은 각 고객에게 별도의 관리 및 운영 경험을 제공하는 전담 팀을 구성해야 하는 경우가 많습니다. 이러한 리소스 중 일부는 고객 간에 공유될 수 있지만 이 모델에서는 일반적으로 신규 고객이 가입할 때마다 비용이 증가합니다.

각 고객이 자체 환경(클라우드 또는 온프레미스)에서 솔루션을 실행하도록 하는 것도 비용에 영향을 미칩니다. 이러한 환경을 확장하려고 시도할 수는 있지만 규모 확장은 단일 고객의 활동으로 제한됩니다. 기본적으로 비용 최적화는 개별 고객 환경의 범위 내에서 달성할 수 있는 수준으로 제한됩니다. 또한 고객 간의 활동 변동을 수용하기 위해 별도의 규모 조정 전략이 필요할 수도 있습니다.

초기에 일부 소프트웨어 기업은 이 모델을 강력한 구조로 간주합니다. 이들은 일회성 커스터마이징을 제공하는 기능을 영업 도구로 사용하여 신규 고객이 자신의 환경에 맞는 고유한 요구 사항을 부과할 수 있도록 합니다. 고객 수와 비즈니스 성장은 여전히 미미하지만, 이 모델은 완벽하게 지속 가능한 것으로 보입니다.

그러나 기업이 광범위한 성공을 거두기 시작하면서 이 모델의 제약으로 인해 실질적인 문제가 발생하기 시작합니다. 예를 들어, 비즈니스가 급격하게 성장하여 많은 신규 고객을 빠른 속도로 추가하는 시나리오를 상상해 보세요. 이러한 성장으로 인해 운영 오버헤드, 관리 복잡성, 비용 및 기타 여러 문제가 가중되기 시작할 것입니다.

궁극적으로 이 모델의 총체적 오버헤드와 영향은 소프트웨어 비즈니스의 성공을 근본적으로 저해하기 시작할 수 있습니다. 첫 번째 문제점은 운영 효율성일 수 있습니다. 고객 유치와 관련된 인력 및 비용 증가로 인해 비즈니스 마진이 감소하기 시작합니다.

하지만 운영 문제는 문제의 일부에 불과합니다. 진짜 문제는 이 모델이 확장됨에 따라 새로운 기능을 출시하고 시장에 뒤쳐지지 않기 위한 기업의 역량에 직접적인 영향을 미치기 시작한다는 것입니다. 각 고객이 고유한 환경을 가지고 있는 경우 공급업체는 시스템에 새로운 기능을 도입하기 위해 다양한 업데이트, 마이그레이션 및 고객 요구 사항의 균형을 맞춰야 합니다.

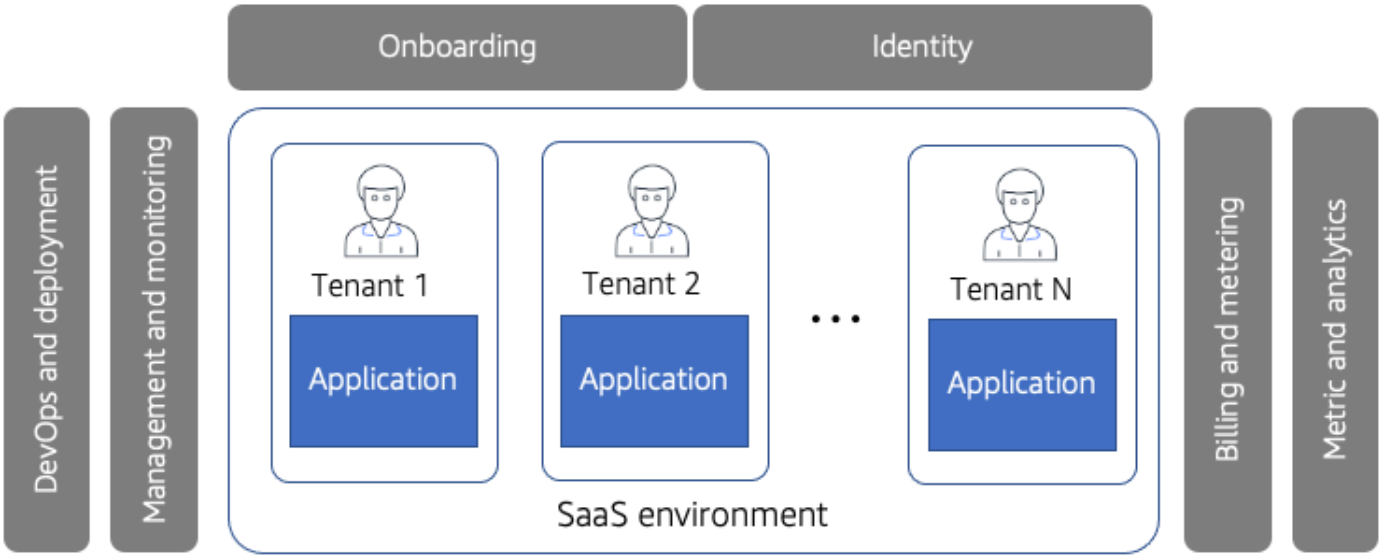
이로 인해 일반적으로 릴리스 주기가 더 길고 복잡해지며, 이로 인해 매년 출시되는 릴리스 수가 줄어드는 경향이 있습니다. 더 중요한 것은 이러한 복잡성으로 인해 팀들이 고객에게 출시되기 훨씬 전에 각각의 새로운 기능을 분석하는 데 더 많은 시간을 할애하고 있다는 것입니다. 팀들은 제공 속도보다는 새로운 기능을 검증하는 데 더 집중하기 시작했습니다. 새로운 기능을 릴리스하는 데 따르는 오버헤드가 너무 커지면 팀은 테스트 메커니즘에 더 집중하고, 제품 혁신을 주도하는 새로운 기능에는 집중하지 않게 됩니다.

이 느리고 신중한 모드에서는 팀의 사이클 시간이 길어지기 때문에 아이디어가 처음 떠오르는 시점과 고객의 손에 닿는 시점 사이의 격차가 점점 더 커지는 경향이 있습니다. 전반적으로 이는 시장 역학 및 경쟁 압력에 대응하는 능력을 저해할 수 있습니다.

통합된 경험으로 이동

이러한 전형적인 소프트웨어 딜레마의 요구 사항을 해결하기 위해 조직은 고객을 공동으로 관리하고 운영할 수 있는 단일 통합 환경을 만들 수 있는 모델로 전환합니다.

다음 다이어그램은 모든 고객이 공유 모델을 통해 관리, 온보딩, 청구 및 운영되는 환경을 개념적으로 보여줍니다.



모든 고객이 공유 모델을 통해 관리, 온보딩, 청구 및 운영되는 환경에 대한 개념적 관점

언뜻 보기에는 이전 모델과 크게 다르지 않은 것처럼 보일 수 있습니다. 하지만 좀 더 자세히 살펴보면 이 두 가지 접근 방식에는 근본적이고 중요한 차이점이 있음을 알 수 있습니다.

먼저, 고객 환경의 이름이 테넌트로 변경되었음을 알 수 있습니다. 이러한 테넌트 개념은 SaaS의 기본입니다. 기본 개념은 단일 SaaS 환경을 운영하고 각 고객을 해당 환경의 테넌트로 간주하여 필요한 리소스를 소비하는 것입니다. 테넌트는 사용자가 많은 회사일 수도 있고 개별 사용자와 직접 연관될 수도 있습니다.

테넌트의 개념을 더 잘 이해하려면 아파트나 상업용 건물을 생각해 보세요. 각 건물의 공간은 개별 임차인에게 임대됩니다. 임차인은 건물의 일부 공유 자원(수도, 전력 등)에 의존하며 소비한 만큼만 지불합니다.

SaaS 테넌트도 비슷한 패턴을 따릅니다. SaaS 환경의 인프라와 해당 환경의 인프라를 사용하는 테넌트가 있습니다. 각 테넌트가 소비하는 리소스의 양은 다를 수 있습니다. 또한 이러한 테넌트는 공동으로 관리, 청구 및 운영됩니다.

다이어그램으로 돌아가면 임차라는 개념이 현실로 구현되는 것을 볼 수 있습니다. 이제 테넌트는 더 이상 자체 환경을 갖지 않습니다. 대신 모든 테넌트는 하나의 공동 SaaS 환경 내에서 수용되고 관리됩니다.

다이어그램에는 SaaS 환경을 둘러싼 다양한 공유 서비스도 포함되어 있습니다. 이러한 서비스는 SaaS 환경의 모든 테넌트에 전 세계적으로 제공됩니다. 즉, 예를 들어 이 환경의 모든 테넌트가 온보딩과 자격 증명을 공유합니다. 관리, 운영, 배포, 청구 및 지표에 대해서도 마찬가지입니다.

모든 테넌트에게 보편적으로 적용되는 통합 서비스 세트에 대한 이러한 개념이 SaaS의 기본입니다. 이러한 개념을 공유하면 위에서 설명한 클래식 모델과 관련된 여러 가지 문제를 해결할 수 있습니다.

이 다이어그램의 또 다른 중요한 요소는 이 환경의 모든 테넌트가 동일한 버전의 애플리케이션을 실행하고 있다는 것입니다. 각 고객에 대해 별도의 일회성 버전을 실행하려는 아이디어는 사라졌습니다. 모든 테넌트가 동일한 버전을 실행하도록 하는 것은 SaaS 환경의 근본적인 특징 중 하나입니다.

모든 고객이 동일한 버전의 제품을 실행하게 되면 기존 설치 소프트웨어 모델의 많은 문제를 더 이상 겪지 않아도 됩니다. 통합 모델에서는 단일한 공유 프로세스를 통해 모든 테넌트에 새 기능을 배포할 수 있습니다.

이 접근 방식을 사용하면 모든 테넌트를 관리하고 운영할 수 있는 단일 운영 창을 사용할 수 있습니다. 공통 운영 환경을 통해 테넌트를 관리하고 모니터링할 수 있으므로 운영 오버헤드를 늘리지 않고도 새 테넌트를 추가할 수 있습니다. 이는 팀에 운영 비용을 줄이고 전반적인 조직 민첩성을 개선할 수 있는 능력을 제공하는 SaaS 가치 제안의 핵심 부분입니다.

이 모델에서 신규 고객을 100명 또는 1,000명 추가한다는 것이 어떤 의미인지 상상해 보세요. 이러한 신규 고객이 어떻게 마진을 약화시키고 복잡성을 가중시킬지 걱정하는 대신, 이러한 성장을 기회로 볼 수 있습니다.

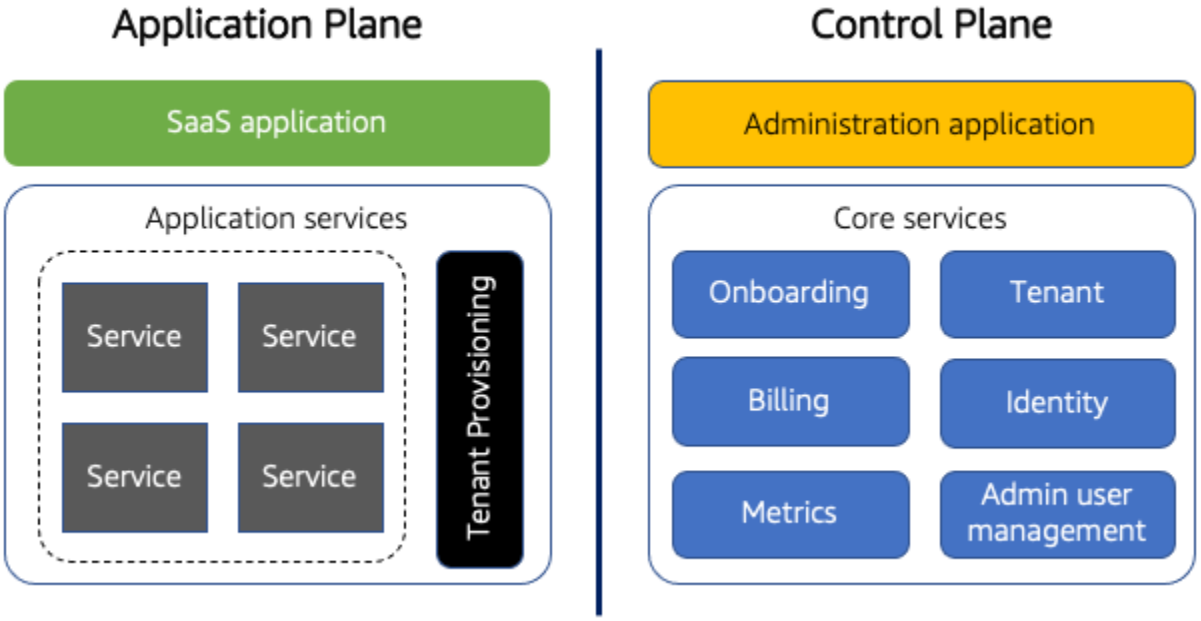
일반적으로 SaaS는 이 모델 중간에 있는 애플리케이션을 구현하는 방법에 중점을 둡니다. 기업은 데이터 저장 방식, 리소스 공유 방식 등에 집중하기를 원합니다. 그러나 실제로 이러한 세부 사항도 중요하지만 애플리케이션을 구축하여 고객에게 SaaS 솔루션으로 제공할 수 있는 방법은 많습니다.

중요한 것은 테넌트 환경을 둘러싸고 통합된 단일 환경을 구현하려는 보다 광범위한 목표입니다. 이러한 경험을 공유하면 SaaS 비즈니스의 전체 목표와 관련된 성장, 민첩성 및 운영 효율성을 주도할 수 있습니다.

컨트롤 플레인 vs. 애플리케이션 플레인

이전 다이어그램은 핵심 SaaS 아키텍처 개념의 개념적 관점을 제공했습니다. 이제 자세히 살펴보고 SaaS 환경이 어떻게 별개의 계층으로 분해되는지 더 잘 정의해 보겠습니다. SaaS 개념 간의 경계를 더 명확하게 파악하면 SaaS 솔루션의 움직이는 부분을 더 쉽게 설명할 수 있습니다.

다음 다이어그램은 SaaS 환경을 두 개의 별개 평면으로 나눕니다. 오른쪽에는 컨트롤 플레인이 있습니다. 이쪽 면에는 멀티테넌트 환경을 온보딩, 인증, 관리, 운영 및 분석하는 데 사용되는 모든 기능과 서비스가 포함되어 있습니다.



컨트롤 플레인 vs. 애플리케이션 플레인

이 컨트롤 플레인은 모든 멀티테넌트 SaaS 모델의 기본입니다. 애플리케이션 배포 및 격리 체계에 관계없이 모든 SaaS 솔루션에는 통합된 단일 환경을 통해 테넌트를 관리하고 운영할 수 있는 기능을 제공하는 서비스가 포함되어야 합니다.

컨트롤 플레인 내에서는 이것들이 두 가지 별개의 요소로 더 세분화되었습니다. 이곳의 핵심 서비스는 멀티테넌트 경험을 오케스트레이션하는 데 사용되는 서비스 모음을 나타냅니다. SaaS 솔루션마다 핵심 서비스가 다를 수 있다는 점을 감안하여 일반적으로 코어에 속하는 몇 가지 일반 서비스 예를 포함했습니다.

또한 별도의 관리 애플리케이션이 있다는 것도 알 수 있습니다. 이는 SaaS 공급자가 멀티테넌트 환경을 관리하는 데 사용할 수 있는 애플리케이션(웹 애플리케이션, 명령줄 인터페이스 또는 API) 을 나타냅니다.

중요한 주의 사항은 컨트롤 플레인과 해당 서비스가 실제로는 멀티테넌트가 아니라는 것입니다. 이 기능은 SaaS 애플리케이션(멀티테넌트이어야 함)의 실제 기능적인 속성을 제공하지 않습니다. 예를 들어 핵심 서비스 중 하나를 살펴보면 멀티테넌트 애플리케이션 기능의 일부인 테넌트 격리 및 기타 구성을 찾을 수 없습니다. 이러한 서비스는 전 세계 모든 테넌트에게 제공됩니다.

다이어그램의 왼쪽은 SaaS 환경의 애플리케이션 플레인을 나타냅니다. 여기에는 애플리케이션의 멀티테넌트 기능이 있습니다. 다이어그램에 나타나는 내용은 다소 모호할 필요가 있습니다. 도메인의 요구 사항, 기술 공간 등에 따라 각 솔루션을 다르게 배포하고 분해할 수 있기 때문입니다.

애플리케이션 도메인은 두 요소로 구분됩니다. 솔루션의 테넌트 경험/애플리케이션을 나타내는 SaaS 애플리케이션이 있습니다. 이는 테넌트가 SaaS 애플리케이션과 상호 작용할 때 터치하는 표면입니다. 그런 다음 SaaS 솔루션의 비즈니스 로직과 기능적 요소를 나타내는 백엔드 서비스가 있습니다. 마이크로서비스일 수도 있고, 애플리케이션 서비스의 다른 패키징일 수도 있습니다.

또한 프로비저닝이 세분화되었다는 사실도 알 수 있습니다. 이는 온보딩 중에 테넌트를 위한 리소스 프로비저닝이 모두 이 애플리케이션 도메인의 일부라는 사실을 강조하기 위한 것입니다. 어떤 사람들은 이것이 컨트롤 플레인의 문제라고 주장할 수도 있습니다. 하지만 프로비저닝하고 구성해야 하는 리소스가 애플리케이션 플레인에서 생성 및 구성된 서비스에 보다 직접적으로 연결되기 때문에 애플리케이션 도메인에 배치했습니다.

이를 별개의 평면으로 나누면 SaaS 아키텍처의 전체 환경에 대해 더 쉽게 파악할 수 있습니다. 더 중요한 것은 애플리케이션 기능 범위를 완전히 벗어나는 일련의 서비스가 필요하다는 점을 강조한다는 것입니다.

핵심 서비스

앞서 언급한 컨트롤 플레인에는 SaaS 환경을 온보딩, 관리 및 운영하는 데 사용되는 일반적인 서비스를 나타내는 일련의 핵심 서비스가 언급되어 있습니다. SaaS 환경에서의 범위와 목적을 강조하기 위해 이러한 서비스 중 일부의 역할을 더 강조하는 것이 도움이 될 수 있습니다. 다음은 이러한 각 서비스에 대한 간략한 요약을 제공합니다.

- 온보딩 — 모든 SaaS 솔루션은 SaaS 환경에 새로운 테넌트를 도입하기 위한 원활한 메커니즘을 제공해야 합니다. 셀프 서비스 가입 페이지일 수도 있고 내부적으로 관리되는 환경일 수도 있습니다. 어느 쪽이든 SaaS 솔루션은 이러한 경험에서 내부 및 외부 마찰을 제거하고 이 프로세스의 안정성, 효율성 및 반복성을 보장할 수 있는 모든 작업을 수행해야 합니다. SaaS 비즈니스의 성장과 규모를 지원하는 데 필수적인 역할을 합니다. 일반적으로 이 서비스는 다른 서비스를 오케스트레이션하여 사용자, 테넌트, 격리 정책, 프로비저닝 및 테넌트별 리소스를 생성합니다.
- 테넌트 - 테넌트 서비스는 테넌트의 정책, 속성 및 상태를 중앙 집중화하는 방법을 제공합니다. 핵심은 테넌트가 개별 사용자가 아니라는 것입니다. 실제로 테넌트는 많은 사용자와 연결되어 있을 가능성이 높습니다.
- 자격 증명 — SaaS 시스템에는 테넌트와 사용자를 연결하여 솔루션의 인증 및 권한 부여 경험에 테넌트 컨텍스트를 제공할 수 있는 명확한 방법이 필요합니다. 이는 온보딩 경험과 사용자 프로필의 전반적인 관리 모두에 영향을 미칩니다.
- 청구 — SaaS 도입의 일환으로, 조직은 종종 새로운 청구 모델을 채택합니다. 또한 타사 청구 제공업체와의 통합을 모색할 수도 있습니다. 이 핵심 서비스는 주로 새 테넌트의 온보딩을 지원하고 테넌트를 위한 청구서를 생성하는 데 사용되는 소비 및 활동 데이터를 수집하는 데 중점을 둡니다.
- 지표 — SaaS 팀은 테넌트의 시스템 사용 방식, 리소스 소비 방식, 테넌트의 시스템 이용 방식에 대한 가시성을 높이는 풍부한 지표 데이터를 캡처하고 분석하는 능력에 크게 의존합니다. 이 데이터는 운영, 제품 및 비즈니스 전략을 수립하는 데 사용됩니다.
- 관리자 사용자 관리 — SaaS 시스템은 테넌트 사용자와 관리자 사용자를 모두 지원해야 합니다. 관리자 사용자는 SaaS 공급자의 관리자를 나타냅니다. SaaS 환경을 모니터링하고 관리하기 위해 운영 경험에 로그인합니다.

멀티테넌시 재정의

멀티테넌시와 SaaS라는 용어는 밀접하게 연관되는 때가 많습니다. 조직에서는 SaaS와 멀티테넌시를 같은 것으로 설명하는 경우가 있습니다. 당연해 보일 수도 있지만, SaaS와 멀티테넌시를 동일시하면 팀들이 SaaS에 대한 순수한 기술적 관점만 취하게 되는 경향이 있는데, 실제로는 SaaS는 아키텍처 전략이라기보다 비즈니스 모델에 가깝습니다.

이 개념을 더 잘 이해하기 위해 멀티테넌시에 대한 고전적인 관점부터 살펴보겠습니다. 순전히 인프라에 초점을 맞춘 이 관점에서 멀티테넌시는 민첩성과 비용 효율성을 높이기 위해 테넌트가 리소스를 공유하는 방법을 설명하는 데 사용됩니다.

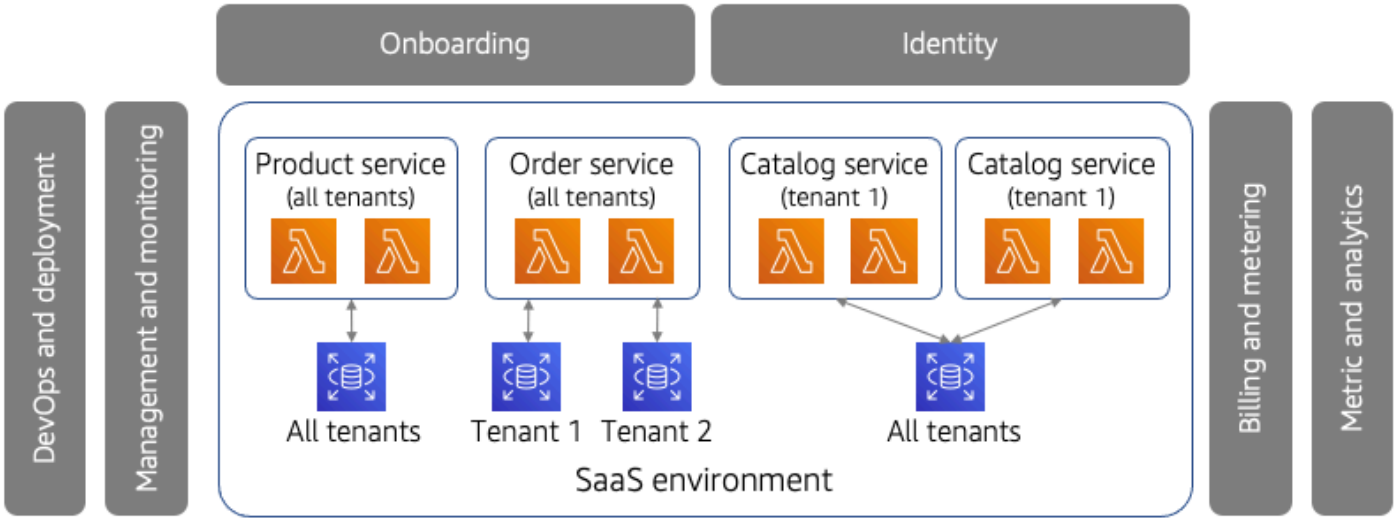
SaaS 시스템의 여러 테넌트가 사용하는 마이크로서비스 또는 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스가 있다고 가정해 보겠습니다. 테넌트는 이 서비스를 실행하는 인프라를 공유하므로 이 서비스는 멀티 테넌트 모델에서 실행되는 것으로 간주됩니다.

이 정의의 문제점은 멀티테넌시라는 기술적 개념을 SaaS에 너무 직접적으로 연결한다는 것입니다. SaaS의 특징은 공유 멀티테넌트 인프라가 있어야 한다는 점이라고 가정합니다. SaaS에 대한 이러한 관점은 다양한 환경에서 SaaS가 구현되는 다양한 방식을 살펴보면서 무너지기 시작합니다.

다음 다이어그램은 멀티테넌시를 정의할 때 발생하는 몇 가지 문제를 보여주는 SaaS 시스템을 보여줍니다.

여기서는 테넌트를 공동으로 관리하고 운영할 수 있는 공유 서비스로 둘러싸인 일련의 애플리케이션 서비스를 통해 앞서 설명한 클래식 SaaS 모델을 볼 수 있습니다.

새로 추가된 것은 여기에 포함된 마이크로서비스입니다. 다이어그램에는 제품, 주문, 카탈로그의 세 가지 샘플 마이크로서비스가 포함되어 있습니다. 각 서비스의 테넌시 모델을 자세히 살펴보면 모두 약간 다른 테넌시 패턴을 사용하고 있음을 알 수 있습니다.



SaaS 및 멀티테넌시

제품 서비스는 모든 리소스(컴퓨팅 및 스토리지)를 모든 테넌트와 공유합니다. 이는 멀티테넌시의 기존 정의와 일치합니다. 그러나 주문 서비스를 살펴보면 컴퓨팅은 공유되지만 각 테넌트마다 별도의 스토리지가 있음을 알 수 있습니다.

카탈로그 서비스는 컴퓨팅이 모든 테넌트마다 분리되어 있지만(각 테넌트에 대한 별도의 마이크로서비스 배포) 모든 테넌트의 스토리지를 공유하는 또 다른 변형을 추가합니다.

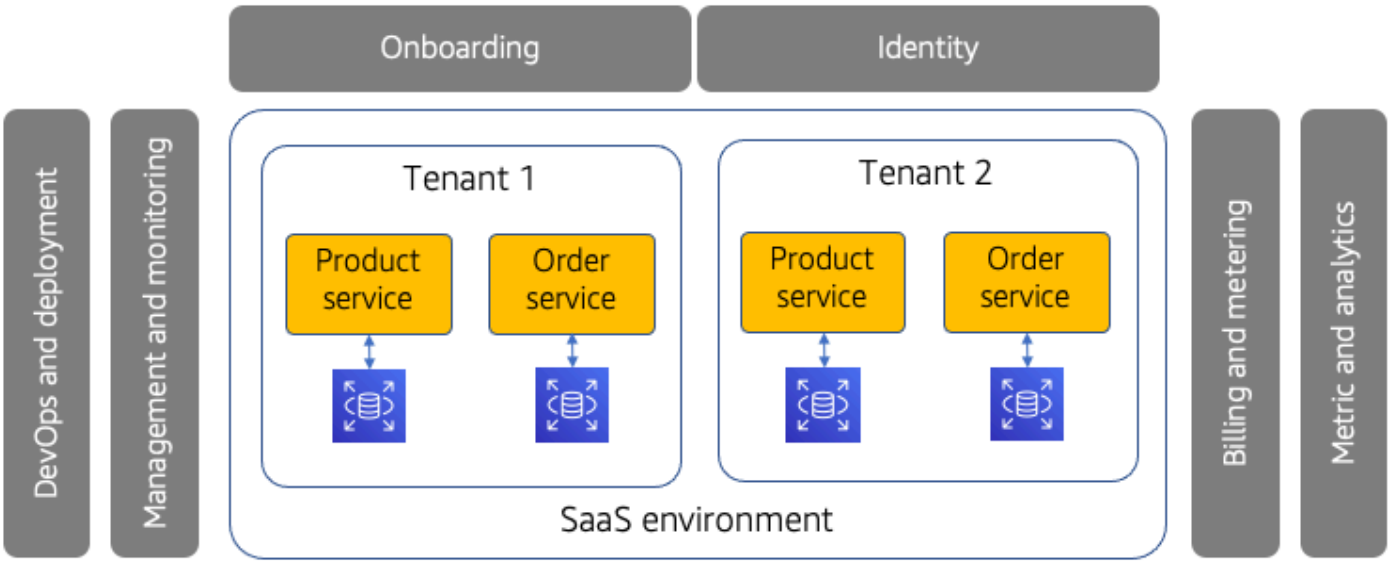
이러한 특성의 변형은 SaaS 환경에서 흔히 볼 수 있습니다. 잡음이 많은 이웃, 계층화 모델, 격리 요구 사항 등이 SaaS 솔루션의 일부를 선택적으로 공유하거나 사일로화할 수 있는 이유 중 하나입니다.

이러한 변형과 기타 여러 가능성을 고려하면 멀티테넌트라는 용어를 사용하여 이 환경을 어떻게 특징지어야 할지 파악하기가 더 어려워집니다. 전반적으로 고객의 관점에서 볼 때 이 환경은 멀티테넌트 환경입니다. 그러나 가장 기술적인 정의를 사용하면 이 환경의 일부는 멀티테넌트이고 일부는 그렇지 않습니다.

SaaS 환경을 특징짓기 위해 멀티테넌트라는 용어를 사용하지 않는 것이 필요한 이유가 바로 여기에 있습니다. 대신 멀티테넌시가 애플리케이션 내에서 어떻게 구현되는지에 대해 설명하겠지만, 솔루션을 SaaS라고 특징짓는 데 멀티테넌시를 사용하는 것은 피해야 합니다. 멀티테넌트라는 용어를 사용하는 경우 전체 SaaS 환경을 멀티테넌트라고 설명하는 데 사용하는 것이 더 합리적입니다. 아키텍처의 일부는 공유될 수 있고 일부는 공유되지 않을 수 있다는 점을 염두에 두어야 합니다. 전반적으로 보면 여전히 멀티테넌트 모델에서 이 환경을 운영 및 관리하고 있습니다.

극단적인 사례

이러한 테넌시 개념을 더 잘 설명하기 위해 리소스를 전혀 공유하지 않는 테넌트가 있는 SaaS 모델을 살펴보겠습니다. 다음 다이어그램은 일부 SaaS 공급자가 사용하는 샘플 SaaS 환경을 제공합니다.



테넌트별 스택

이 다이어그램을 보면 이러한 테넌트를 둘러싼 공통된 환경이 여전히 남아 있음을 알 수 있습니다. 하지만 각 테넌트는 전용 리소스 컬렉션과 함께 배포됩니다. 이 모델에서는 테넌트가 아무 것도 공유하지 않습니다.

이 예시에서는 멀티테넌트가 무엇을 의미하는지 의문을 제기합니다. 공유되는 리소스가 없는데도 멀티테넌트 환경일까요? 이 시스템을 사용하는 테넌트는 공유 리소스가 있는 SaaS 환경에 대해 기대하는 것과 동일합니다. 실제로 SaaS 환경에서 리소스가 어떻게 배포되는지 알지 못할 수도 있습니다.

이러한 테넌트는 사일로화된 인프라에서 운영되지만 여전히 공동으로 관리되고 운영됩니다. 이들은 하나의 통합된 온보딩, 자격 증명, 지표, 청구 및 운영 경험을 공유합니다. 또한 새 버전이 출시되면 모든 테넌트에게 배포됩니다. 이렇게 하려면 개별 테넌트에 대한 일회성 사용자 지정을 허용할 수 없습니다.

이 극단적인 예는 멀티테넌트 SaaS의 개념을 테스트하기 위한 좋은 모델을 제공합니다. 공유 인프라의 효율성을 모두 실현할 수는 없지만 완전히 유효한 멀티테넌트 SaaS 환경입니다. 일부 고객의 경우 도메인에 따라 일부 또는 모든 고객이 이 모델을 사용하도록 지시할 수 있습니다. 그렇다고 SaaS가 아니라는 의미는 아닙니다. 이러한 공유 서비스를 사용하고 모든 테넌트가 동일한 버전을 실행하는 경우에도 SaaS의 기본 원칙을 준수합니다.

이러한 매개변수와 SaaS의 광범위한 정의를 고려하면 멀티테넌트라는 용어의 사용을 발전시켜야 할 필요성을 알 수 있습니다. 총체적으로 관리 및 운영되는 모든 SaaS 시스템을 멀티테넌트라고 부르는 것이 더 합리적입니다. 그런 다음 더 세분화된 용어를 사용하여 SaaS 솔루션 구현 내에서 리소스가 공유되거나 전용되는 방식을 설명할 수 있습니다.

싱글 테넌트 용어 제거

멀티테넌트라는 용어를 사용하는 경우, SaaS 환경을 설명할 때 싱글 테넌트라는 용어를 사용하려는 사람들도 당연히 생깁니다. 그러나 앞서 설명한 배경을 고려할 때 싱글 테넌트라는 용어는 혼란을 야기합니다.

위 다이어그램은 싱글 테넌트 환경일까요? 각 테넌트는 엄밀히 따지자면 자체 스택을 가지고 있지만 이러한 테넌트는 여전히 멀티테넌트 모델에서 운영 및 관리되고 있습니다. 이것이 싱글 테넌트라는 용어를 일반적으로 사용하지 않는 이유입니다. 대신 모든 환경은 멀티테넌트로 특징지어집니다. 일부 또는 모든 리소스가 공유되거나 전용되는 다양한 테넌시를 구현하기 때문입니다.

사일로 및 풀 소개

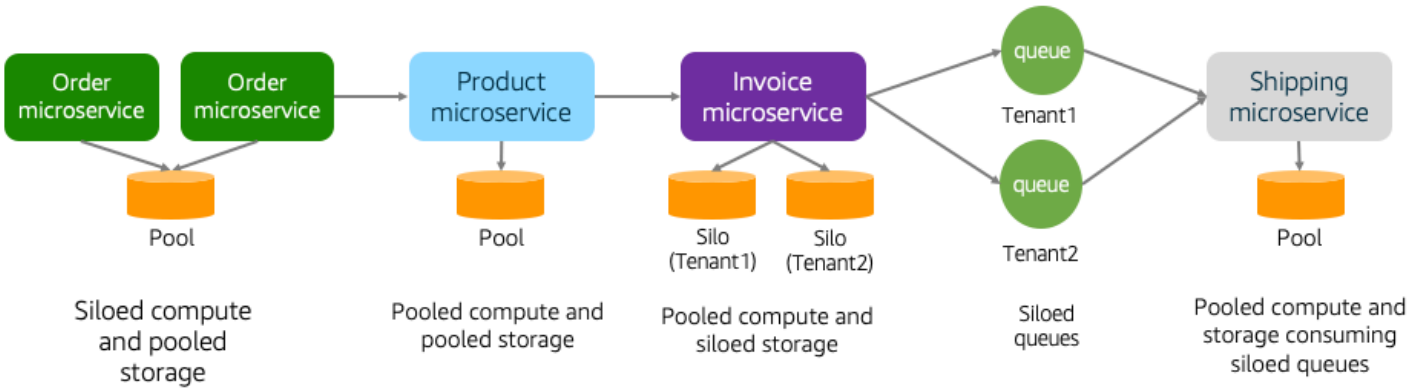
이러한 다양한 모델을 사용하고 멀티테넌시라는 용어와 관련된 문제를 고려하여 SaaS를 구축할 때 사용되는 다양한 모델을 보다 정확하게 캡처하고 설명할 수 있는 몇 가지 용어를 도입했습니다.

SaaS 환경에서 리소스 사용을 특성화하기 위해 사용하는 두 가지 용어는 사일로와 풀입니다. 이러한 용어를 통해 멀티테넌트를 여러 기본 모델에 적용할 수 있는 중요한 설명으로 사용하여 SaaS 환경의 특성을 분류할 수 있습니다.

가장 기본적인 수준에서 사일로라는 용어는 리소스가 특정 테넌트 전용으로 사용되는 시나리오를 설명하기 위한 것입니다. 반대로 풀 모델은 테넌트가 리소스를 공유하는 시나리오를 설명하는 데 사용됩니다.

사일로와 풀 용어가 사용되는 방식을 살펴볼 때 사일로와 풀은 전부 또는 전무의 개념이 아님을 분명히 하는 것이 중요합니다. 사일로와 풀은 전체 테넌트의 리소스 스택에 적용하거나 전체 SaaS 환경의 일부에 선택적으로 적용할 수 있습니다. 따라서 일부 리소스가 사일로 모델을 사용한다고 해서 해당 환경의 모든 리소스가 사일로화된 것은 아닙니다. 풀링이라는 용어를 사용하는 방식도 마찬가지입니다.

다음 다이어그램은 SaaS 환경에서 사일로 및 풀링된 모델을 보다 세부적으로 사용하는 방법의 예를 제공합니다.



사일로 및 풀 모델

이 다이어그램에는 사일로 및 풀 모델의 보다 구체적인 특성을 설명하기 위한 일련의 샘플이 포함되어 있습니다. 이 그림을 왼쪽에서 오른쪽으로 따라가면 주문 마이크로서비스로 시작하는 것을 볼 수 있습니다. 이 마이크로서비스에는 사일로화된 컴퓨팅과 풀링된 스토리지가 있습니다. 이는 풀링된 컴퓨팅과 풀링된 스토리지가 있는 제품 서비스와 상호 작용합니다.

그런 다음 제품 서비스는 풀링된 컴퓨팅과 사일로화된 스토리지가 있는 청구서 마이크로서비스와 상호 작용합니다. 이 서비스는 대기열을 통해 배송 서비스에 메시지를 전송합니다. 대기열은 사일로화된 모델로 배포됩니다.

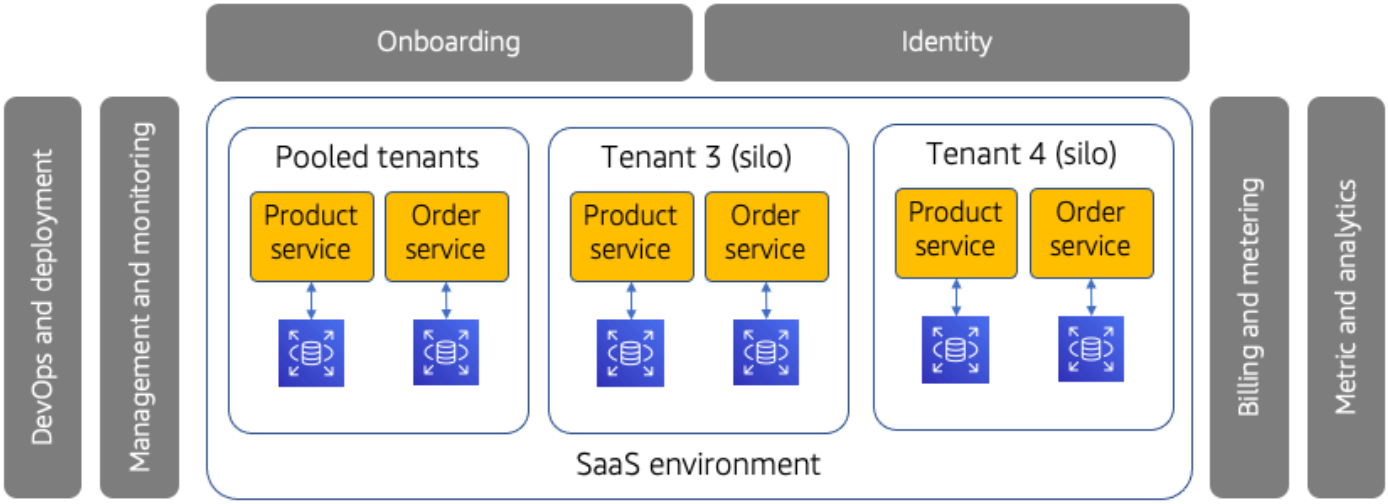
마지막으로 배송 마이크로서비스는 사일로화된 대기열에서 메시지를 수집합니다. 이것은 풀링된 컴퓨팅 및 스토리지를 사용합니다.

다소 복잡해 보일 수 있지만 목표는 사일로 및 풀 개념의 세분화된 특성을 강조하는 것입니다. SaaS 솔루션을 설계하고 구축할 때는 도메인과 고객의 요구 사항을 기반으로 이러한 사일로 및 풀 관련 결정을 내려야 합니다.

시끄러운 이웃, 격리, 계층화 및 기타 여러 가지 이유가 사일로 또는 풀링 모델을 적용하는 방법과 시기에 영향을 미칠 수 있습니다.

전체 스택 사일로 및 풀

사일로 및 풀을 사용하여 전체 SaaS 스택을 설명할 수도 있습니다. 이 접근 방식에서는 테넌트의 모든 리소스가 전용 또는 공유 방식으로 배포됩니다. 다음 다이어그램은 이것이 SaaS 환경에 어떻게 적용되는지에 대한 예를 제공합니다.



전체 스택 사일로 및 풀 모델

이 다이어그램에서는 전체 스택 테넌트 배포를 위한 세 가지 모델이 있음을 알 수 있습니다. 먼저 전체 스택 풀 환경이 있다는 것을 알 수 있습니다. 이 풀의 테넌트는 모든 리소스(컴퓨팅, 스토리지 등)를 공유합니다.

표시된 나머지 두 스택은 전체 스택 사일로화된 테넌트 환경을 나타냅니다. 이 경우 테넌트 3과 테넌트 4는 각각 다른 테넌트와 리소스를 공유하지 않는 전용 스택이 있는 것으로 표시됩니다.

동일한 SaaS 환경에서 사일로 모델과 풀링 모델을 혼합하는 것은 그다지 이례적인 일이 아닙니다. 예를 들어 시스템 사용에 대해 적당한 가격을 지불하는 기본 계층 테넌트가 있다고 가정해 보겠습니다. 이러한 테넌트는 풀링된 환경에 배치됩니다.

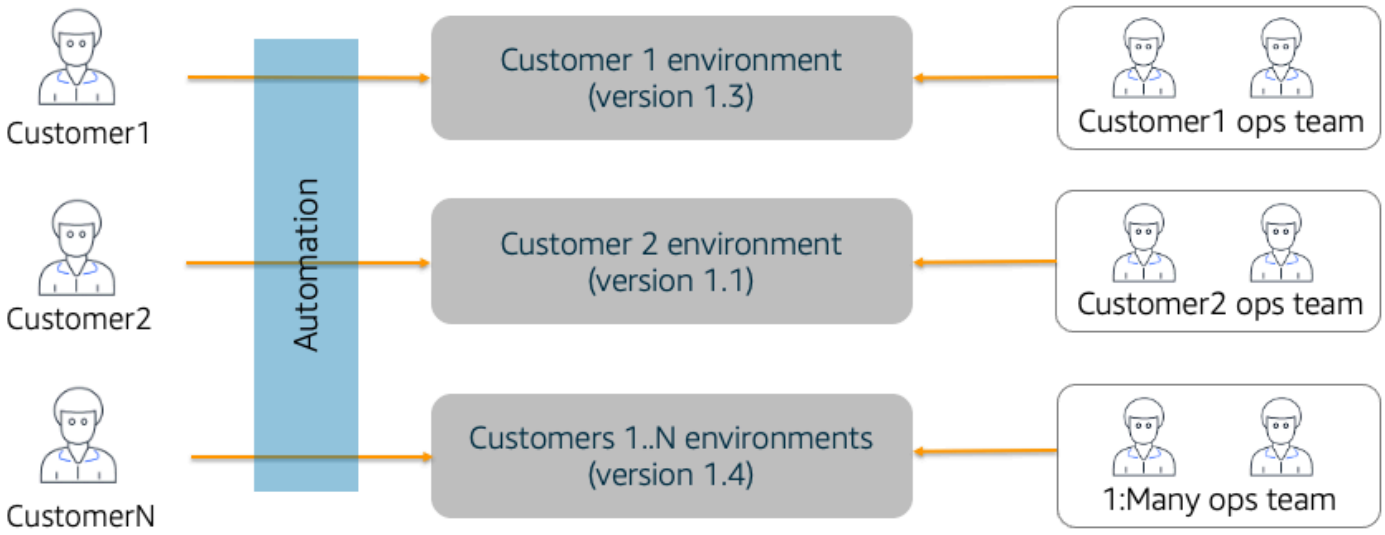
한편, 사일로에서 운영할 수 있는 특권을 얻기 위해 기꺼이 추가 비용을 지불하려는 프리미엄 등급 테넌트도 있을 수 있습니다. 이러한 고객은 별도의 스택으로 배포됩니다(다이어그램 참조).

테넌트가 자체적인 전체 스택 사일로에서 운영하도록 허용한 이 모델에서도 이러한 사일로는 이러한 테넌트에 대한 일회성 변형이나 사용자 지정을 허용하지 않는 것이 중요합니다. 모든 측면에서 이러한 각 스택은 동일한 소프트웨어 버전을 사용하여 동일한 스택 구성을 실행해야 합니다. 새 버전이 출시되면 풀링된 테넌트 환경과 각 사일로 환경에 배포됩니다.

SaaS와 관리형 서비스 제공업체(MSP) 비교

또한 SaaS와 관리형 서비스 제공업체(MSP) 모델 간의 경계를 둘러싸고 있는 혼란도 있습니다. MSP 모델을 보면 SaaS 모델과 비슷한 목표를 가지고 있는 것처럼 보일 수 있습니다.

그러나 MSP를 좀 더 자세히 살펴보면 MSP와 SaaS는 실제로 다르다는 것을 알 수 있습니다. 다음 다이어그램은 MSP 환경의 개념적 관점을 제공합니다.



관리형 서비스 제공업체(MSP) 모델

이 다이어그램은 MSP 모델에 대한 한 가지 접근 방식을 나타냅니다. 왼쪽에는 MSP 모델을 사용하는 고객이 있습니다. 일반적으로 여기서는 가능한 모든 자동화를 사용하여 각 고객 환경을 프로비저닝하고 해당 고객을 위한 소프트웨어를 설치하는 접근 방식을 사용합니다.

오른쪽은 MSP가 이러한 고객 환경을 지원하기 위해 제공할 운영 공간을 대략적으로 나타낸 것입니다.

중요한 점은 MSP가 해당 고객이 실행하려는 제품 버전을 설치하고 관리하는 경우가 많다는 점입니다. 모든 고객이 동일한 버전을 실행할 수 있지만 MSP 모델에서는 일반적으로 이 기능이 필요하지 않습니다.

일반적인 전략은 이러한 환경의 설치 및 관리를 담당하여 소프트웨어 공급자의 업무를 단순화하는 것입니다. 이렇게 하면 공급자의 업무가 단순해지기는 하지만 SaaS 서비스에 필수적인 가치와 사고방식과 직접적으로 일치하지는 않습니다.

관리 책임을 덜어주는 데 중점을 두는 것입니다. 이러한 전환은 모든 고객이 동일한 버전에서 단일 통합 관리 및 운영 환경을 사용하도록 하는 것과는 다릅니다. 대신 MSP는 별도의 버전을 허용하는 경우가 많으며 이러한 각 환경을 운영상 분리된 것으로 간주하는 경우가 많습니다.

MSP가 SaaS와 겹치기 시작할 수 있는 영역이 분명히 있습니다. MSP가 기본적으로 모든 고객이 동일한 버전을 실행하도록 요구하고 MSP가 하나의 경험을 통해 모든 테넌트를 중앙에서 온보딩, 관리, 운영 및 청구할 수 있었다면 MSP보다 SaaS가 더 많아지기 시작할 수 있습니다.

더 넓게 보자면 환경 설치를 자동화하는 것이 SaaS 환경을 갖는 것과 동일하지 않다는 것입니다. 앞서 설명한 다른 모든 주의 사항을 추가한 경우에만 이것이 진정한 SaaS 모델에 더 가깝습니다.

이 이야기의 기술 및 운영 측면에서 다시 돌아가면 MSP와 SaaS 사이의 경계는 더욱 뚜렷해집니다. 일반적으로 SaaS 비즈니스로서 서비스의 성공 여부는 경험의 모든 움직이는 부분에 깊이 관여하는 능력에 달려 있습니다.

이는 일반적으로 온보딩 경험의 흐름을 파악하고, 운영 이벤트가 테넌트에 미치는 영향을 이해하고, 주요 지표와 분석을 추적하고, 고객과 가깝게 다가가는 것을 의미합니다. 이를 다른 사람에게 넘겨주는 MSP 모델에서는 SaaS 비즈니스 운영의 핵심인 주요 세부 정보에서 제외될 수 있습니다.

SaaS 마이그레이션

SaaS를 채택한 많은 제공업체는 이전에 설명한 기존 설치 소프트웨어 모델에서 SaaS로 마이그레이션합니다. 이러한 제공업체의 경우 SaaS의 핵심 원칙을 잘 준수하는 것이 특히 중요합니다.

여기서도 SaaS 모델로 마이그레이션한다는 것이 무엇을 의미하는지에 대해 혼란이 있을 수 있습니다. 예를 들어 클라우드로의 이동을 SaaS로 마이그레이션하는 것으로 보는 사람도 있습니다. 설치 및 프로비저닝 프로세스에 자동화를 추가하는 것을 마이그레이션을 달성하는 것으로 보는 사람도 있습니다.

조직마다 출발점이 다르고, 기존 고려 사항이 다르며, 시장 및 경쟁 압력에 직면할 가능성이 높다고 해도 과언이 아닙니다. 즉, 모든 마이그레이션은 다르게 보일 것입니다.

모든 경로는 다르지만 마이그레이션 전략을 형성하는 핵심 원칙과 관련해서는 단절된 부분이 있는 부분도 있습니다. 개념과 원칙을 잘 조정하면 SaaS 마이그레이션의 전반적인 성공에 상당한 영향을 미칠 수 있습니다.

앞서 설명한 개념을 바탕으로 SaaS로의 전환은 비즈니스 전략과 목표에서 시작된다는 점을 분명히 해야 합니다. 가능한 한 빨리 SaaS로 전환해야 한다는 압박이 있는 마이그레이션 환경에서는 이 점을 놓칠 수 있습니다.

이 모드에서, 조직에서는 마이그레이션을 주로 기술적 절차로 간주하는 경우가 많습니다. 실제로 모든 SaaS 마이그레이션은 대상 고객, 서비스 경험, 운영 목표 등을 명확하게 파악하는 것에서 시작해야 합니다. SaaS 비즈니스에 필요한 모습에 더 명확하게 초점을 맞추면 솔루션을 SaaS로 마이그레이션하기 위해 취하는 형태, 우선 순위 및 경로에 큰 영향을 미칩니다.

마이그레이션 초기부터 이러한 명확한 비전을 갖추면 SaaS로의 전환의 일환으로 기술과 비즈니스를 모두 마이그레이션하는 방법의 토대를 마련할 수 있습니다. 이 길을 걷기 시작하면서 향하고 있는 방향에 대해 가장 잘 알려줄 수 있는 질문에 집중하세요.

다음 표는 기술 및 비즈니스 마이그레이션 사고방식의 대조적인 특성을 보여줍니다.

표 1 — 기술 우선 마이그레이션과 비즈니스 우선 마이그레이션 비교

기술 우선 사고방식	비즈니스 우선 사고방식
테넌트 데이터를 분리하려면 어떻게 해야 할까요?	SaaS는 비즈니스 성장에 어떻게 도움이 될까요?

기술 우선 사고방식	비즈니스 우선 사고방식
사용자를 테넌트와 연결하려면 어떻게 해야 할까요?	어떤 세그먼트를 타겟팅하고 있나요?
시끄러운 이웃 상황을 피하려면 어떻게 해야 할까요?	이 세그먼트의 크기와 프로필은 어떤가요?
A/B 테스트는 어떻게 하나요?	어떤 등급을 지원해야 할까요?
테넌트 부하를 기반으로 규모를 조정하려면 어떻게 해야 할까요?	어떤 서비스 경험을 목표로 하고 있나요?
어떤 청구 제공업체를 이용해야 하나요?	가격 책정 및 패키지 전략은 무엇인가요?

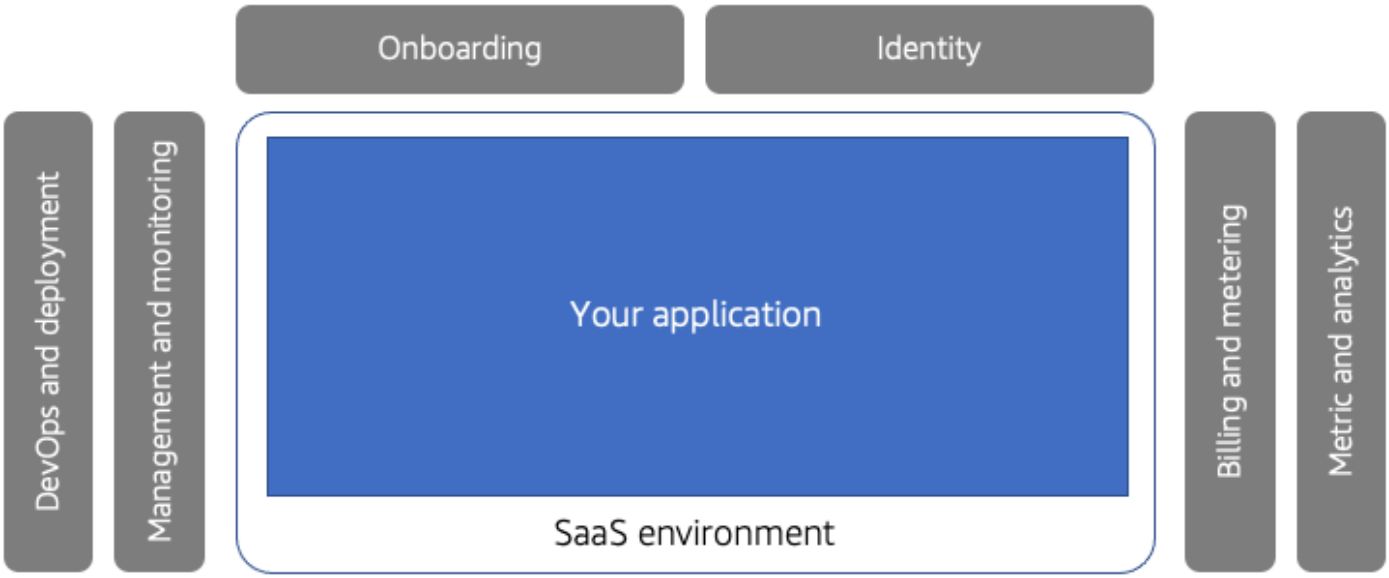
왼쪽은 기술 우선 마이그레이션 사고방식입니다. 엔지니어링 팀은 모든 SaaS 아키텍처에서 확실히 중요한 클래식 멀티테넌트 주제를 추구하는 데 집중하고 있습니다.

문제는 왼쪽에 있는 많은 질문에 대한 답변이 오른쪽 질문에 대한 답변의 직접적인 영향을 받는 경우가 많다는 것입니다. 마이그레이션을 고려하는 사람에게는 이 점이 생소할 것 같지 않습니다. 그러나 실제로 많은 조직은 비즈니스 부분이 저절로 해결될 것이라고 가정하고 운영 및 비용 효율성을 첫 단계로 추구하는 것으로 시작합니다.

이 마이그레이션 전략 내에서는 레거시 환경을 SaaS 모델에 맞게 어떻게 발전시킬지에 대한 혼란도 있을 수 있습니다. 이 분야에서도 SaaS로 마이그레이션할 수 있는 다양한 옵션이 있습니다. 하지만 모든 마이그레이션에 대해 일반적으로 권장하는 공통 가치 체계가 있습니다.

SaaS 원칙에 대한 이전 논의에서 SaaS 환경을 설명하는 데 사용되는 다양한 패턴과 용어에 대해 설명했습니다. 이러한 모든 솔루션을 포괄하는 공통된 주제 중 하나는 애플리케이션을 둘러싸고 서비스를 공유하자는 아이디어입니다. 자격 증명, 온보딩, 지표, 청구 등은 모두 SaaS 환경의 핵심인 공통 요소로 불립니다.

이제 마이그레이션을 살펴보면 동일한 공유 서비스가 모든 마이그레이션 사례에서 핵심적인 역할을 한다는 것을 알 수 있습니다. 다음 다이어그램은 마이그레이션 환경에 대한 개념적 관점을 제공합니다.



SaaS로 마이그레이션

이 다이어그램은 모든 마이그레이션 경로의 대상 경험을 나타냅니다. 여기에는 앞서 설명한 것과 동일한 공유 서비스가 모두 포함됩니다. 가운데에는 애플리케이션의 자리 표시자가 있습니다.

핵심 아이디어는 이러한 환경 한가운데에 원하는 수의 애플리케이션 모델을 도입할 수 있다는 것입니다. 마이그레이션의 첫 번째 단계에서는 각 테넌트가 자체 사일로에서 실행되도록 할 수 있습니다. 또는 요소가 사일로화되고 일부 기능은 현대화된 마이크로서비스 모음을 통해 처리되는 하이브리드 아키텍처를 사용할 수도 있습니다.

처음에 애플리케이션을 현대화하는 정도는 레거시 환경의 특성, 시장 요구 사항, 비용 고려 사항 등에 따라 달라집니다. 변하지 않는 것은 이러한 공유 서비스의 도입입니다.

모든 SaaS 마이그레이션은 기업이 SaaS 모델에서 운영할 수 있도록 이러한 기본 공유 서비스를 지원해야 합니다. 예를 들어, 애플리케이션 아키텍처의 모든 변형에는 SaaS 자격 증명이 필요합니다. SaaS 솔루션을 관리하고 모니터링하려면 테넌트 인식 운영이 필요합니다.

마이그레이션 초기에 이러한 공유 서비스를 도입하면 기본 애플리케이션이 여전히 각 테넌트의 전체 스택 사일로에서 실행되고 있더라도 고객에게 SaaS 경험을 제공할 수 있습니다.

일반적인 목표는 SaaS 모델에서 애플리케이션을 실행하는 것입니다. 그런 다음 애플리케이션의 추가 현대화 및 개선에 관심을 돌릴 수 있습니다. 또한 이 접근 방식을 통해 비즈니스의 다른 부분(마케팅, 영업, 지원 등)을 더 빠른 속도로 이전할 수 있습니다. 더 중요한 것은 이를 통해 고객 참여를 시작하고 고객 피드백을 수집하여 환경을 지속적으로 현대화하는 데 사용할 수 있다는 것입니다.

사용하는 공유 서비스에는 궁극적으로 필요한 모든 기능이나 메커니즘이 포함되어 있지 않을 수 있다는 점에 유의해야 합니다. 주요 목표는 마이그레이션 초기에 필요한 공유 메커니즘을 만드는 것입니다. 이를 통해 애플리케이션 아키텍처의 발전과 운영 발전에 필수적인 시스템 요소에 집중할 수 있습니다.

SaaS 자격 증명

SaaS는 애플리케이션의 자격 증명 모델에 새로운 고려 사항을 추가합니다. 각 사용자가 인증되면 특정 테넌트 컨텍스트에 연결되어야 합니다. 이 테넌트 컨텍스트는 SaaS 환경 전체에서 사용되는 테넌트에 대한 필수 정보를 제공합니다.

테넌트와 사용자의 이러한 바인딩을 애플리케이션의 SaaS 자격 증명이라고도 합니다. 각 사용자가 인증할 때 자격 증명 공급자는 일반적으로 사용자 자격 증명과 테넌트 자격 증명을 모두 포함하는 토큰을 생성합니다.

테넌트를 사용자에게 연결하는 것은 다운스트림에 많은 영향을 미치는 SaaS 아키텍처의 기본 요소입니다. 이 자격 증명 프로세스의 토큰은 애플리케이션의 마이크로서비스로 유입되며 테넌트 인식 로그를 생성하고, 지표를 기록하고, 청구를 측정하고, 테넌트 격리를 적용하는 등의 작업에 사용됩니다.

사용자를 테넌트에 매핑하는 별도의 독립형 메커니즘에 의존하는 시나리오를 피하는 것이 중요합니다. 이로 인해 시스템 보안이 약화될 수 있으며 아키텍처에 병목 현상이 발생하는 경우가 많습니다.

테넌트 격리

고객을 멀티 테넌트 모델로 전환시킬수록 고객은 한 테넌트가 다른 테넌트의 리소스에 액세스할 가능성에 대해 더 우려하게 됩니다. SaaS 시스템에는 공유 인프라에서 실행되는 경우에도 각 테넌트의 리소스가 격리되도록 하는 명시적인 메커니즘이 포함되어 있습니다.

이를 테넌트 격리라고 합니다. 테넌트 격리의 기본 개념은 SaaS 아키텍처가 리소스에 대한 액세스를 엄격하게 제어하고 다른 테넌트의 리소스에 액세스하려는 시도를 차단하는 구성을 도입한다는 것입니다.

단, 테넌트 격리는 일반 보안 메커니즘과는 별개입니다. 시스템은 인증 및 권한 부여를 지원하지만 테넌트 사용자가 인증되었다고 해서 시스템이 격리된 것은 아닙니다. 격리는 응용 프로그램의 일부일 수 있는 기본 인증 및 권한 부여와는 별도로 적용됩니다.

이를 더 잘 이해하기 위해 자격 증명 공급자를 사용하여 SaaS 시스템에 대한 액세스를 인증했다고 가정해 보겠습니다. 이 인증 경험의 토큰에는 특정 애플리케이션 기능에 대한 사용자 액세스를 제어하는 데 사용할 수 있는 사용자 역할에 대한 정보도 포함될 수 있습니다. 이러한 구조는 보안을 제공하지만 격리는 제공하지 않습니다. 실제로 사용자는 인증을 받고 권한을 부여받은 후에도 다른 테넌트의 리소스에 계속 액세스할 수 있습니다. 인증 및 권한 부여에 관한 어떤 것도 이러한 액세스를 반드시 차단할 수는 없습니다.

테넌트 격리는 테넌트 컨텍스트를 사용하여 리소스에 대한 액세스를 제한하는 데만 중점을 둡니다. 현재 테넌트의 컨텍스트를 평가하고 해당 컨텍스트를 사용하여 해당 테넌트가 액세스할 수 있는 리소스를 결정합니다. 이 격리는 해당 테넌트 내의 모든 사용자에게 적용됩니다.

다양한 SaaS 아키텍처 패턴에서 테넌트 격리가 실현되는 방식을 살펴보면 이는 더욱 어려워집니다. 네트워크(또는 좀 더 세부적인) 정책으로 테넌트 간 액세스를 허용하지 않는 경우 전체 리소스 스택을 테넌트 전용으로 사용하여 격리를 달성할 수도 있습니다. 다른 시나리오에서는 리소스 액세스를 제어하기 위해 보다 세분화된 정책이 필요한 풀링된 리소스(Amazon DynamoDB 테이블의 항목)가 있을 수 있습니다.

테넌트 리소스에 액세스하려는 모든 시도는 해당 테넌트에 속하는 리소스로만 범위를 지정해야 합니다. 특정 애플리케이션의 격리 요구 사항을 지원할 도구 및 기술 조합을 결정하는 것은 SaaS 개발자와 설계자의 임무입니다.

데이터 파티셔닝

데이터 파티셔닝은 멀티테넌트 환경에서 데이터를 나타내는 데 사용되는 다양한 전략을 설명하는 데 사용됩니다. 이 용어는 다양한 데이터 구성을 개별 테넌트와 연결하는 데 사용할 수 있는 다양한 접근 방식과 모델을 포괄하는 데 광범위하게 사용됩니다.

데이터 파티셔닝과 테넌트 격리를 서로 바꿔서 사용할 수 있는 것으로 볼 수도 있습니다. 하지만 이 두 개념은 동일하지 않습니다. 데이터 파티셔닝에 대해 이야기할 때는, 개별 테넌트의 테넌트 데이터가 저장되는 방식에 대해 설명하는 것입니다. 데이터를 파티셔닝한다고 해서 데이터가 격리되는 것은 아닙니다. 한 테넌트가 다른 테넌트의 리소스에 액세스할 수 없도록 하려면 여전히 격리를 별도로 적용해야 합니다.

각 AWS 스토리지 기술에는 데이터 파티셔닝 전략에 대한 고유한 고려 사항이 있습니다. 예를 들어, Amazon DynamoDB에서 데이터를 격리하는 것은 Amazon Relational Database Service(RDS)를 사용하여 데이터를 분리하는 것과 매우 다르게 보입니다.

일반적으로 데이터 파티셔닝을 고려할 때는 먼저 데이터를 사일로화할지 아니면 풀링할지 고민합니다. 사일로 모델에서는 데이터를 혼합하지 않고도 각 테넌트마다 고유한 스토리지 구조를 갖게 됩니다. 풀 파티셔닝의 경우 각 테넌트와 관련된 데이터를 결정하는 테넌트 식별자를 기반으로 데이터가 혼합 및 파티셔닝됩니다.

예를 들어, Amazon DynamoDB의 경우 사일로 모델은 각 테넌트에 대해 별도의 테이블을 사용합니다. Amazon DynamoDB에서 데이터를 풀링하려면 모든 테넌트의 데이터를 관리하는 각 Amazon DynamoDB 테이블의 파티션 키에 테넌트 식별자를 저장해야 합니다.

각 서비스마다 사일로 및 풀링된 스토리지 모델을 구현하기 위해 서로 다른 접근 방식이 필요할 수 있는 자체 구조를 도입하는 등 AWS 서비스 범위에 따라 이러한 상황이 어떻게 달라질지 상상할 수 있습니다.

데이터 파티셔닝과 테넌트 격리는 별개의 주제이지만 선택하는 데이터 파티셔닝 전략은 데이터 격리 모델의 영향을 받을 가능성이 높습니다. 예를 들어 도메인이나 고객의 요구 사항에 가장 잘 맞는 접근 방식이 있기 때문에 일부 스토리지를 사일로화할 수 있습니다. 또는 풀 모델에서는 솔루션에 필요한 세분화 수준으로 격리를 적용할 수 없기 때문에 사일로를 선택할 수도 있습니다.

시끄러운 이웃도 격리 방식에 영향을 미칠 수 있습니다. 애플리케이션의 일부 워크로드 또는 사용 사례는 다른 테넌트의 영향을 제한하거나 서비스 수준에 관한 계약(SLA)을 충족하기 위해 별도로 보관해야 할 수도 있습니다.

측정, 지표 및 청구

SaaS에 대한 논의에는 측정, 지표 및 청구에 대한 개념도 포함되는 경향이 있습니다. 이러한 개념은 종종 하나의 개념으로 통합됩니다. 그러나 SaaS 환경에서 측정, 지표 및 청구가 수행하는 다양한 역할을 구분하는 것이 중요합니다.

이러한 개념의 문제점은 같은 단어를 겹쳐서 사용하는 경우가 많다는 것입니다. 예를 들어 청구서 생성에 사용되는 측정에 대해 설명해 보겠습니다. 이와 동시에 청구와 관련이 없는 리소스의 내부 사용량을 추적하는 데 사용되는 측정에 대해서도 설명해 드릴 수 있습니다. 또한 이 논의에 뒤섞일 수 있는 다양한 상황에서의 지표와 SaaS에 대해서도 설명합니다.

이 문제를 해결하는 데 도움이 되도록 몇 가지 구체적인 개념을 각 용어에 연결해 보겠습니다. 여기에는 절대적인 개념이 없다는 점을 알고 있습니다.

- 측정 — 이 개념은 여러 정의가 있지만 SaaS 청구 도메인에 가장 적합합니다. 테넌트의 활동이나 리소스 사용량을 측정하여 청구서 생성에 필요한 데이터를 수집하는 것이 좋습니다.
- 지표 — 지표는 비즈니스, 운영 및 기술 영역 전반의 추세를 분석하기 위해 캡처하는 모든 데이터를 나타냅니다. 이 데이터는 SaaS 팀 내의 다양한 상황과 역할에서 사용됩니다.

이러한 구분은 중요하지 않지만 SaaS 환경에서 측정 및 지표의 역할에 대한 생각을 단순화하는 데 도움이 됩니다.

이제 이 두 개념을 예제에 연결하면 청구서 생성에 필요한 데이터를 표시하는 데 사용되는 특정 측정 이벤트를 애플리케이션에 적용하는 것을 생각할 수 있습니다. 이는 요청 수, 활성 사용자 수일 수도 있고 고객이 이해할 수 있는 특정 단위와 관련된 일부 사용량 집계(요청, CPU, 메모리)와 매핑될 수도 있습니다.

SaaS 환경에서는 애플리케이션에서 이러한 청구 이벤트를 게시하고 SaaS 시스템에서 사용하는 청구 구성에 의해 수집 및 적용됩니다. 이는 타사 청구 시스템일 수도 있고 사용자 지정 시스템일 수도 있습니다.

이와는 대조적으로, 측정 이면의 사고 방식은 다양한 테넌트가 시스템에 부과하는 상태 및 운영 공간을 평가하는 데 필수적인 행동, 활동, 소비 패턴 등을 파악하는 것입니다. 여기에 게시하고 집계하는 지표는 운영팀, 제품 소유자, 설계자 등 다양한 페르소나의 필요에 따라 더 많이 좌우됩니다. 여기서는 이 지표 데이터를 몇 가지 분석 도구에 게시하고 집계하여 각 사용자가 자신의 특성에 가장 잘 맞는 시스템 측면을 분석하는 시스템 활동 뷰를 구축할 수 있도록 합니다. 제품 소유자는 다양한 테넌트가 기능을 어떻게 사용하고 있는지 알고 싶어할 수 있습니다. 건축가에게는 테넌트가 인프라 리소스 등을 소비하는 방식을 이해하는 데 도움이 되는 뷰가 필요할 수 있습니다.

B2B 및 B2C SaaS

SaaS 상품은 B2B 및 B2C 시장 모두를 위해 만들어졌습니다. 시장과 고객의 동태는 확실히 다르지만, SaaS의 전반적인 원칙이 각 시장을 변화시키지는 않습니다.

예를 들어 온보딩을 살펴보면 B2B 고객과 B2C 고객의 온보딩 경험이 다를 수 있습니다. B2C 시스템이 셀프 서비스 온보딩 흐름에 더 초점을 맞출 수 있는 것은 사실입니다(B2B 시스템에서도 이를 지원할 수도 있음).

온보딩이 고객에게 제공되는 방식에는 차이가 있을 수 있지만 온보딩의 기본 가치는 거의 동일합니다. B2B 솔루션이 내부 온보딩 프로세스에 의존하더라도, 해당 프로세스는 최대한 원활하고 자동화될 것으로 예상됩니다. B2B가 된다고 해서 시간이 지나면서 기대치를 변화시켜 고객을 위한 가치를 창출하고 있다는 뜻은 아닙니다.

결론

이 문서의 목적은 SaaS 패턴 및 전략을 특성화하는 데 사용되는 모델 및 용어에 대한 설명을 제공하여 기본적인 SaaS 아키텍처 개념을 개괄적으로 설명하는 것입니다. 이를 통해 조직이 전체 SaaS 환경을 더 명확하게 볼 수 있습니다.

여기서는 SaaS가 무엇을 의미하는지에 초점을 맞추고 있으며, 통합 경험을 통해 모든 SaaS 테넌트를 관리하고 운영할 수 있는 환경을 만드는 데 중점을 둡니다. 이는 SaaS가 무엇보다도 비즈니스 모델이라는 핵심 아이디어와 연결됩니다. 구축하는 SaaS 아키텍처는 이러한 기본 비즈니스 목표를 촉진하기 위한 것입니다.

참조 자료

여기에 설명된 패턴을 준수하는 SaaS 아키텍처 패턴에 대해 더 자세히 설명하는 많은 리소스가 있습니다.

자세한 내용은 다음을 참조하세요.

- SaaS 테넌트 격리 전략(AWS 백서)
- SaaS 스토리지 전략(AWS 백서)
- Well-Architected SaaS 렌즈(AWS 백서)

기여자

다음 개인과 조직이 이 문서에 기여했습니다.

- Tod Golding, 수석 파트너 솔루션 아키텍트, AWS SaaS Factory

문서 수정

이 백서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
최초 게시	백서가 게시되었습니다.	2022년 8월 3일

고지 사항

고객은 본 문서의 정보를 독립적으로 평가할 책임이 있습니다. 본 문서는 (a) 정보 제공의 목적으로만 제공되고, (b) 사전 통지 없이 변경될 수 있는 현재 AWS 제품 및 관행을 나타내고, (c) AWS 및 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 약속이나 보증도 하지 않습니다. AWS 제품 또는 서비스는 명시적이든 묵시적이든 어떠한 종류의 보증, 진술 또는 조건 없이 '있는 그대로' 제공됩니다. 고객에 대한 AWS의 책임 및 채무는 AWS 계약에 준거합니다. 본 문서는 AWS와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하세요.