AWS ホワイトペーパー

SaaS アーキテクチャの基礎

SaaS アーキテクチャの基礎: AWS ホワイトペーパー

Table of Contents

SaaS アーキテクチャの基礎

出版日:2022 年 8 月 3 日 (ドキュメントの改訂)

Software as a Service (SaaS) モデルでビジネスを運用することの範囲、目標、および性質を定義するのは困難な場合があります。SaaS を特徴付ける用語とパターンは、その起源によって異なります。このドキュメントの目標は、SaaS の基本的な要素をより明確に定義し、AWS で SaaS システムを設計およびデリバリーする際に適用されるパターン、用語、価値体系をより正確に把握することです。より大きな目標としては、SaaS デリバリーモデルの採用を検討する際に、考慮すべきオプションについてより明確に把握できるように、一連の基礎的な分析情報を提供することです。

本誌は、SaaS の導入を開始したばかりの SaaS ビルダーやアーキテクト、そして SaaS のコアコンセプトについての理解を深めたい経験豊富なビルダーを対象としています。この情報の一部は、SaaS のランドスケープに精通したい SaaS プロダクトオーナーやストラテジストにも役立ちます。

序章

Software as a Service (SaaS) という用語は、ビジネスモデルとデリバリーモデルを説明するために使用されます。ただし、課題は、SaaS であることの意味が広く理解されていないことです。

SaaS の中核となる柱のいくつかについてはある程度共通の理解がありますが、SaaS であることの意味については混乱が残っています。チームの SaaS に対する見方に多少のばらつきがあるのは当然です。しかし、それと同時に SaaS の概念や用語が明確でないと、SaaS デリバリーモデルを検討している人にとって混乱を招く可能性があります。

このドキュメントでは、SaaS の中核となる概念について説明するために使用される用語の概要を説明することに重点を置いています。これらの概念について共通の考え方を持つことで、SaaS アーキテクチャの基本的な要素が明確になり、SaaS アーキテクチャの構成要素を説明するための共通の語彙が身に付きます。これは、これらのテーマに基づいて構築される追加コンテンツについて調べるときに特に役立ちます。

このホワイトペーパーでは、マルチテナンシーのアーキテクチャの詳細ではなく、SaaS であることの意味について、その基礎をどのように定義してきたかを説明します。また、理想的には、これによって用語がより明確になり、組織が SaaS ソリューションの種類や性質についてより迅速に認識できるようになればと思います。

Well-Architected とは

AWS Well-Architected フレームワークは、クラウド内でのシステム構築に伴う意思決定の長所と短所を理解するのに役立ちます。このフレームワークの 6 つの柱により、信頼性、安全性、効率、費用対効果、持続可能性の高いシステムを設計および運用するための、アーキテクチャのベストプラクティスを確認できます。AWS マネジメントコンソール に無料で提供されている AWS Well-Architected Tool を使用すると、各柱に関する一連の質問に答えることで、これらのベストプラクティスに照らしてワークロードをレビューできます。

SaaS レンズでは、AWS で Software as a Service (SaaS) ワークロードを構築するためのベストプラクティスに重点を置いています。

クラウドアーキテクチャに関する専門的なガイダンスやベストプラクティス (リファレンスアーキテクチャのデプロイ、図、ホワイトペーパー) については、AWS アーキテクチャセンターを参照してください。

SaaS はビジネスモデルである

SaaS とは何かを定義するには、まず SaaS がビジネスモデルであるという重要な原則を理解しておくことから始めます。とりわけ、SaaS デリバリーモデルの採用は、一連のビジネス目標によって直接的に影響を受けます。確かに、こうした目標の実現にはテクノロジーが使用されるでしょうが、SaaS は、特定のビジネス目標を対象とする考え方とモデルを構築します。

SaaS デリバリーモデルの採用に関連する主要なビジネス目標について、いくつか詳しく見てみましょう。

- 俊敏性 — この用語は、SaaS のより大きな目標を要約したものです。成功している SaaS 企業は、市場、カスタマー、競争のダイナミクスに継続的に適応する準備が整っている必要がある、という考えに基づいて構築されています。優れた SaaS 企業は、新しい価格モデル、新しい市場セグメント、新しいカスタマーニーズを継続的に取り入れるように組織されています。
- 運用効率 — SaaS 企業は、規模と俊敏性を促進するために運用効率を重視しています。つまり、新機能の頻繁かつ迅速なリリースを円滑化する運用基盤の構築に重点を置いた文化とツールを整備する必要があります。また、すべてのカスタマー環境をまとめて管理、運用、デプロイできる、統一されたエクスペリエンスが得られる必要もあります。1 回限りのバージョンやカスタマイズをサポートするという考えはもはやありません。SaaS ビジネスは、ビジネスの成長と拡大を成功させるうえで重要な柱として、運用効率を重視しています。
- スムーズなオンボーディング — 俊敏性を高め、成長を受け入れる一環として、テナントカスタマーのオンボーディングプロセスにおける摩擦を減らすことも重視する必要があります。これは、企業間 (B2B) および企業対カスタマー (B2C) のカスタマーに広く当てはまります。どのセグメントやタイプのカスタマーをサポートしていても、カスタマーにとっての価値の実現までにかかる時間を重視する必要があります。サービス中心モデルに移行するには、SaaS ビジネスはカスタマーエクスペリエンスのあらゆる側面に注力し、特にオンボーディングライフサイクル全体の再現性と効率性に重点を置く必要があります。
- イノベーション — SaaS への移行は、単に現在のカスタマーのニーズに対応することではなく、イノベーションを可能にする基盤となる要素を整えることでもあります。SaaS モデルでは、カスタマーニーズに反応し、対応したいと考えるでしょう。しかし、この俊敏性を活用して将来的なイノベーションを推進し、カスタマーに新しい市場や機会、効率性を提供したいとも考えるでしょう。
- 市場の反応 — SaaS では、四半期ごとのリリースや 2 年計画といった従来の概念はありません。組織が市場のダイナミクスにほぼリアルタイムで対応し、それに対応できるようにするには、俊敏性が欠かせません。SaaS の組織的、技術的、文化的要素への投資は、新たなカスタマーと市場のダイナミクスに基づいて、ビジネス戦略を転換する機会を生み出します。

- 成長 — SaaS は成長を中心としたビジネス戦略です。組織のすべての可動部分を俊敏性と効率性に基づいて調整することで、SaaS 組織は成長モデルを目標にすることができます。つまり、提供する SaaS サービスの迅速な採用を歓迎する仕組みを整えるということです。

これらの項目では、それぞれビジネスの成果に焦点を当てていることに気付くでしょう。SaaS システムの構築に使用できる技術戦略とパターンは多岐にわたります。ただし、これらの技術戦略についても、ビジネス全体を変化させることはありません。

SaaS の導入の一環として何を達成しようとしているのかについて組織と話し合い、まずはこのビジネスに焦点を当てた議論から開始します。技術の選択は重要ですが、これらのビジネス目標を念頭に置いて実現する必要があります。例えば、俊敏性、運用効率、スムーズなオンボーディングを実現せずにマルチテナントになると、SaaS ビジネスの成功にはつながりません。

これを背景にして、これを前述の原則に従った SaaS のより簡潔な定義に定式化してみましょう。

SaaS は、よりスムーズで、サービス中心型モデルで組織がソリューションを提供できるようにするビジネスおよびソフトウェアデリバリーモデルであり、カスタマーやプロバイダーにとっての価値を最大化します。成長、リーチ、イノベーションを促進するビジネス戦略の柱として、俊敏性と運用効率は欠かせません。

ビジネス目標が一致していることと、すべてのカスタマーで エクスペリエンスを共有することがいかに重要であるかがわかるはずです。SaaS への移行で重要なのは、従来のソフトウェアモデルの一部となっている可能性のあるカスタマイゼーションを止めることです。カスタマーに固有の提供を行うことは、一般に、SaaS で達成しようとするコアバリューから遠ざかります。

サービスであって製品ではない

「サービス」モデルの採用は、単なるマーケティングまたは用語ではありません。サービスの考え方では、従来の製品ベースの開発アプローチから遠ざかっていることに気付くでしょう。どの製品にとっても機能は確かに重要ですが、SaaS では、カスタマーがサービスで体験できることに重点を置いています。

これは何を意味するのでしょうか? サービス中心モデルでは、カスタマーがどのようにサービスにオンボーディングするか、どれだけ早く価値を実現するか、カスタマーのニーズに対応する機能をいかに迅速に導入できるかについて、より深く考える必要があります。サービスの構築、運用、管理方法に関する詳細は、カスタマーには分かりません。

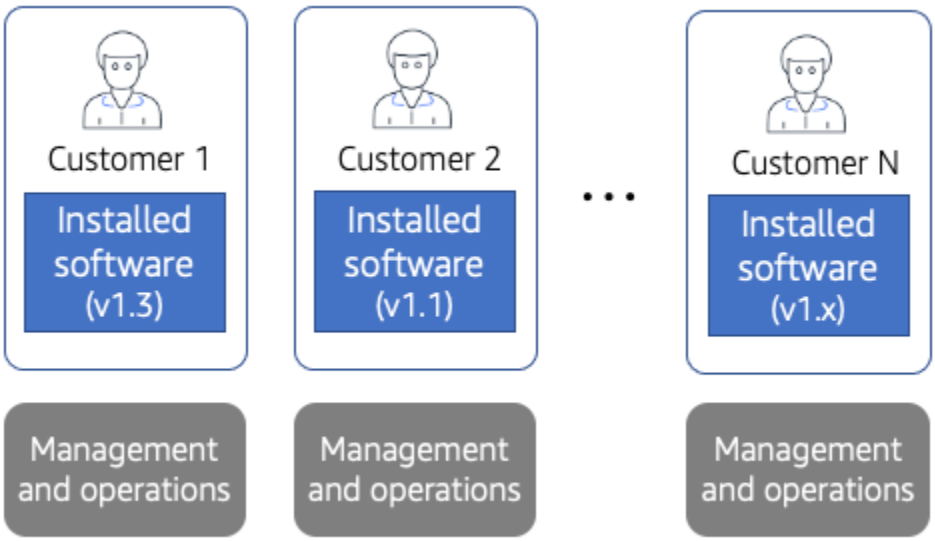
このモードでは、この SaaS サービスを他に利用する可能性のあるサービスと同じように考えます。レストランにいると、確かに料理も気になりますが、サービスも気になります。ウェイターがテーブルにどれだけ早く来るか、どのくらいの頻度で水を補充するか、料理がどれだけ速く提供されるか、これらはすべてサービスエクスペリエンスの尺度です。これは、SaaS サービスの構築についての私たちの考え方を形成するのと同じマインドセットおよび価値体系です。

このサービスとしてのモデルは、チームやサービスの構築方法に大きな影響を与えるはずです。作業のバックログにおいて、これらの体験に関する性質のものは、機能と同等またはそれより高い地位に置かれることになります。企業も、これらを SaaS サービスの長期的な成長と成功の基盤と見なします。

初期動機

SaaS を理解するために、SaaS ビジネスを構築する際に達成しようとしていることについて、非常にシンプルな概念から開始しましょう。まずは、従来の (SaaS 以外の) ソフトウェアがどのように作成、管理、運用されてきたかを調べてみましょう。

次の図は、複数のベンダーがどのようにソリューションをパッケージ化して提供してきたかを示した概念図です。



ソフトウェアソリューションのパッケージングとデリバリーのクラシックモデル

この図では、一連のカスタマー環境を示しています。これらのカスタマーは、ベンダーのソフトウェアを購入したさまざまな企業や団体です。これらのカスタマーはそれぞれ、基本的にソフトウェアプロバイダーの製品をインストールしたスタンドアロン環境で運用しています。

この場合、各カスタマーのインストールは、そのカスタマー専用のスタンドアロン環境として扱われます。つまり、カスタマーは自身をこれらの環境の所有者とみなし、必要に応じて 1 回限りのカスタマイズや独自の構成を要求する可能性があります。

この考え方によく見られる問題の 1 つは、実行する製品のバージョンをカスタマーが制御してしまうことです。この問題はさまざまな理由によって発生する可能性があります。カスタマーは、新機能に不安を感じたり、新しいバージョンの採用に伴う混乱を懸念しているかもしれません。

このような動きがソフトウェアプロバイダーの運用フットプリントにどのように影響するかは想像できるでしょう。カスタマーに 1 回限りの環境を許可すればするほど、各カスタマーの異なる構成の管理、更新、サポートが難しくなります。

このような 1 回限りの環境を必要とするとき、多くの場合、組織はカスタマー別に管理およびオペレーションエクスペリエンスを提供する専任チームを結成する必要があります。これらのリソースの一部はカスタマー間で共有される可能性がありますが、このモデルでは、新規カスタマーがオンボーディングするたびに経費が増加するのが一般的です。

各カスタマーが独自の環境 (クラウドまたはオンプレミス) でソリューションを実行することも、コストに影響します。これらの環境のスケーリングを試みることはできますが、このスケーリングは単一のカスタマーのアクティビティに限定されます。基本的に、コストの最適化は、個々のカスタマー環境の範囲内で達成できることに限定されます。また、カスタマーごとのアクティビティの差異に対応するため、個別のスケーリング戦略が必要になる場合もあります。

当初、一部のソフトウェア企業はこのモデルを強力なコンストラクトと見なすでしょう。1 回限りのカスタマイズ機能を販売ツールとして活用し、新規カスタマーがその環境に固有の要件を課せるようにします。カスタマー数の増加とビジネスの成長は大きくありませんが、このモデルは完全に持続可能に思われます。

しかし、企業の成功が大きくなるにつれ、このモデルの制約は、現実的な課題を生み出し始めます。例えば、ビジネスが急成長し、多くの新規カスタマーが急速に増えているシナリオを想像してみてください。この増加により、運用上のオーバーヘッド、管理の複雑さ、コスト、その他多くの問題が追加されます。

そして最終的には、このモデルの全体的なオーバーヘッドと影響により、ソフトウェアビジネスの成功を根本的に損なう可能性があります。運用効率が最初の課題となる場合もあります。カスタマー獲得に伴って人員とコストが増加すると、ビジネスのマージンが損なわれ始めます。

ただし、運用上の問題は課題の一部にすぎません。本当の問題は、このモデルがスケーリングするにつれ、新しい機能をリリースし、市場と歩調を合わせるビジネスの能力に直接的に影響し始めることです。各カスタマーが独自の環境を保有する場合、プロバイダーはシステムに新しい機能を導入するとき、多数の更新、移行、およびカスタマー要件の間でバランスを取る必要があります。

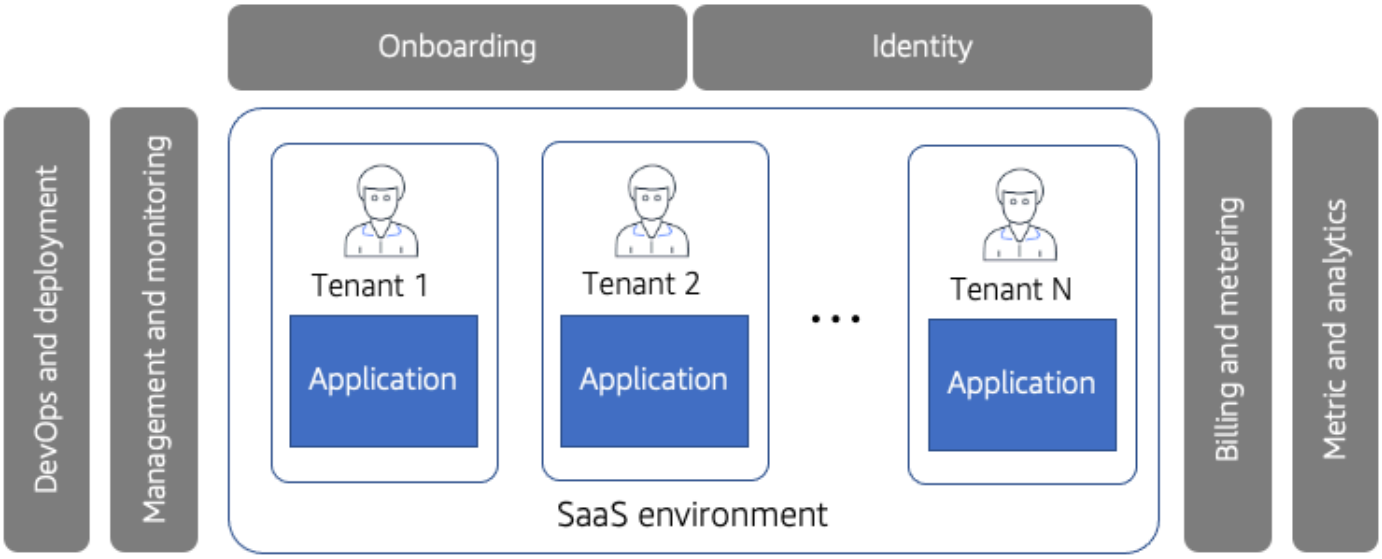
通常、これはリリースサイクルを長く複雑にし、毎年行われるリリースの数を減少させる傾向があります。さらに重要なのは、この複雑さによって、チームは、カスタマーにリリースするずっと前から各新機能の分析により多くの時間を費やさなければならないということです。チームは、デリバリーのスピードよりも、新機能の検証に重点を置くようになります。新機能のリリースのオーバーヘッドが非常に大きくなるため、チームはサービスのイノベーションを促進する新機能よりも、テストの仕組みに集中するようになります。

この遅く、より慎重なモードでは、チームのサイクル時間が長くなる傾向があり、アイデアを思いついてからカスタマーの手に渡るまでの時間がますます大きくなります。全体として、これは市場のダイナミクスや競争圧力に対応する能力を損なう可能性があります。

統一された体験への移行

この典型的なソフトウェアのジレンマのニーズに対応するため、組織はカスタマーを一元的に管理および運用できる単一かつ統一された体験を生むモデルを求めています。

次の図は、すべてのカスタマーを共有モデルで管理、オンボーディング、請求、および運用する環境の概念図を示しています。



すべてのカスタマーを共有モデルで管理、オンボーディング、請求、および運用する環境の概念図

一見すると、これは以前のモデルとそれほど変わらないように見えるかもしれません。ただし、もう少し掘り下げてみると、これら 2 つのアプローチには根本的かつ大きな違いがあることがわかります。

まず、カスタマー環境の名前がテナントに変更されていることに気付くでしょう。このテナントの概念は SaaS の基本です。基本的な考え方は、単一の SaaS 環境があり、各カスタマーがその環境のテナントと見なされ、必要なリソースを使用するというものです。テナントは、多数のユーザーがいる会社の場合もあれば、個々のユーザーに直接関連している場合もあります。

テナントの概念についてより詳しく理解するには、アパートや商業ビルについて考えてみてください。これらの建物の各スペースは、個々のテナントに貸し出されています。テナントは、建物の一部の共有資源 (水道、電力など) に依存し、消費した分だけを支払います。

SaaS テナントも同様のパターンに従います。SaaS 環境のインフラストラクチャがあり、その環境のインフラストラクチャを利用するテナントがいるとします。各テナントが使用するリソースの量は異なる場合があります。しかし、これらのテナントもまとめて管理、請求、運用されています。

ここで図に戻ると、テナンシーの概念を現実的なものとして理解できるでしょう。テナントにもはや独自の環境はありません。代わりに、すべてのテナントは 1 つの集合的な SaaS 環境の壁の中に収容され、管理されています。

この図には、SaaS 環境を取り巻くさまざまな共有サービスも含まれています。これらのサービスは、SaaS 環境のすべてのテナントにグローバルに適用されます。例えばオンボーディングや ID は、この環境のすべてのテナントで共有されるということです。管理、オペレーション、デプロイ、請求、メトリクスについても同じことが言えます。

すべてのテナントに普遍的に適用される統一された一連のサービスという考えは、SaaS の基礎です。これらの概念を共有することで、前述の典型的なモデルに関わる多くの課題を解消できます。

この図でもう 1 つの重要でありながら、やや微妙な要素は、この環境のすべてのテナントが同じバージョンのアプリケーションを実行していることです。ここでは、カスタマー別に個別の 1 回限りのバージョンを実行するという考えはもはや存在しません。すべてのテナントが同じバージョンを実行していることは、SaaS 環境の基本的な特徴の 1 つです。

すべてのカスタマーで同じバージョンの製品を稼働させることにより、従来のインストール型のソフトウェアモデルで直面していた多くの課題に対応する必要がなくなります。統合モデルでは、単一の共有プロセスですべてのテナントに新しい機能をデプロイできます。

このアプローチでは、すべてのテナントを一元的に管理および運用できます。これにより、共通の運用環境でテナントを管理および監視できるため、運用上のオーバーヘッドを増やすことなく新しいテナントを追加できます。これは、SaaS の価値提案の中核部分であり、チームが運用コストを削減し、組織全体の俊敏性を向上させることができます。

このモデルで 100 人または 1,000 人の新規カスタマーを追加するとどうなるか想像してみてください。新規カスタマーによって減少する利益や追加される複雑さについて心配するのではなく、これは成長がもたらす機会ととらえることができます。

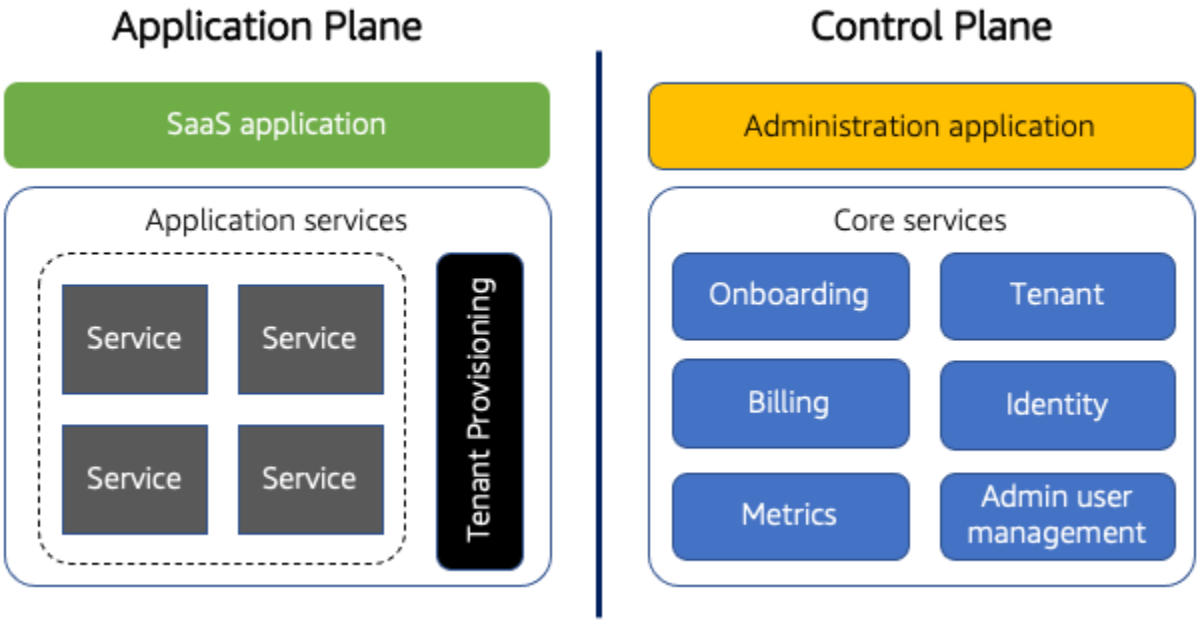
一般的に、SaaS では、このモデルの中心にあるアプリケーションをどのように実装するかが焦点です。企業は、データの保存方法やリソースの共有方法などを重視したいと考えます。こうした詳細は確かに重要ですが、現実には、アプリケーションを構築して SaaS ソリューションとしてカスタマーに提供する方法はたくさんあります。

重要なのは、テナント環境を取り巻く単一の統一された体験を実現するというより大きな目標です。このような共有の体験は、SaaS ビジネスの全体的な目標につながる成長、俊敏性、運用効率の向上につながります。

コントロールプレーンとアプリケーションプレーン

前述の図は、中核となる SaaS アーキテクチャの概念を示しています。それでは、その詳細を掘り下げて、SaaS 環境がどのように個別のレイヤーに分解されているかについてより明確に定義しましょう。SaaS の概念の境界をこのように明確に把握できれば、SaaS ソリューションの可動部分が分かりやすくなります。

次の図では、SaaS 環境を 2 つの異なるプレーンに分割しています。右側はコントロールプレーンです。図のこちら側には、マルチテナント環境のオンボーディング、認証、管理、運用、分析に使用されるすべての機能とサービスが含まれています。



コントロールプレーンとアプリケーションプレーン

このコントロールプレーンは、あらゆるマルチテナント SaaS モデルの基礎です。すべての SaaS ソリューションには、アプリケーションのデプロイや分離スキームにかかわらず、単一の統一されたエクスペリエンスを通じてテナントを管理および運用できるサービスを含める必要があります。

コントロールプレーンでは、これをさらに 2 つの異なる要素に分解しました。ここで紹介するコアサービスは、マルチテナントエクスペリエンスを調整するために使用されるサービスの集合です。コアサービスはそれぞれの SaaS ソリューションによってある程度異なる可能性があることを踏まえ、一般にコアの一部となるサービスの共通例をいくつか含めました。

また、独立した管理アプリケーションについても紹介しています。これは、SaaS プロバイダーがマルチテナント環境を管理するために使用する可能性のあるアプリケーション (Web アプリケーション、コマンドラインインターフェイス、または API) を表します。

重要な点は、コントロールプレーンとそのサービスは、実際にはマルチテナントではないということです。この機能は、SaaS アプリケーションの実際の機能属性を提供するものではありません (これはマルチテナントである必要があります)。例えば、コアサービスのどれか 1 つについて調べても、マルチテナントアプリケーション機能の一部であるテナント分離やその他の構成要素は見つかりません。これらのサービスはすべてのテナントに共通です。

図の左側は、SaaS 環境のアプリケーションプレーンです。これがアプリケーションのマルチテナント機能が存在する場所です。各ソリューションは、ドメインのニーズやテクノロジーのフットプリントなどに基づいて異なる方法でデプロイおよび分解できるため、図に示されている内容はある程度曖昧なままにしておく必要があります。

アプリケーションドメインは 2 つの要素に分かれています。ソリューションのテナントエクスペリエンスまたはアプリケーションを表す SaaS アプリケーションがあります。これは、テナントが SaaS アプリケーションを操作するときに触れる表面です。次に、SaaS ソリューションのビジネスロジックと機能要素を表すバックエンドサービスがあります。これらは、マイクロサービスでも、またはアプリケーションサービスの他のパッケージでもかまいません。

また、プロビジョニングが分割されていることにも気付くでしょう。これは、オンボーディング中にテナントにリソースをプロビジョニングすることが、このアプリケーションドメインの一部であるという事実を強調するためです。これはコントロールプレーンの機能だと主張する人もいるかもしれません。ただし、プロビジョニングと構成が必要なリソースは、アプリケーションプレーンで作成および構成されるサービスに直接的に関連しているため、アプリケーションドメインに配置されています。

このように個別のプレーンに分解すると、SaaS アーキテクチャの全体的な状況について把握するのが容易になります。さらに重要なのは、アプリケーション機能の範囲にはない一連のサービスの必要性が強調されていることです。

コアサービス

前述のコントロールプレーンでは、SaaS 環境のオンボーディング、管理、運用に使用される一般的なサービスを表す一連のコアサービスについて触れました。これらのサービスの一部の役割についてさらに焦点を当て、SaaS 環境におけるサービスの範囲と目的を強調することは、役立つ場合があります。以下に、これらの各サービスの簡単な概要について記載しています。

- オンボーディング — どの SaaS ソリューションも、SaaS 環境に新しいテナントを導入するため、円滑なメカニズムを提供する必要があります。これは、セルフサービスのサインアップページでも、社内で管理されたエクスペリエンスでもかまいません。いずれにしても、SaaS ソリューションでは、このようなエクスペリエンスから内部と外部の摩擦を取り除き、このプロセスの安定性、効率性、再現性を確保するため、できる限りのことを行う必要があります。SaaS ビジネスの成長と規模をサポートする上で重要な役割を果たします。通常、このサービスは他のサービスをオーケストレーションして、ユーザー、テナント、分離ポリシー、プロビジョニング、およびテナント別のリソースを作成します。
- テナント — テナントサービスは、テナントのポリシー、属性、および状態を一元管理する方法を提供します。重要なのは、テナントは個々のユーザーではないということです。実際、テナントは多くのユーザーと関連している可能性があります。
- アイデンティティ — SaaS システムには、ソリューションの認証および認可エクスペリエンスにテナントコンテキストを与える、ユーザーをテナントに結びつける明確な方法が必要です。これは、オンボーディングエクスペリエンスとユーザープロファイルの全体管理の両方に影響します。
- 請求 — SaaS の導入の一環として、組織は新しい請求モデルを採用することがよくあります。また、サードパーティの請求プロバイダーとの統合を検討する場合もあります。このコアサービスは、主に新規テナントのオンボーディングをサポートし、テナントの請求書作成に使用される使用量データやアクティビティデータを収集することに重点を置いています。
- メトリクス — SaaS チームにとって、テナントがシステムをどのように使用し、どのようにリソースを消費し、どのようにシステムを利用しているかをより明確に把握できるように、豊富なメトリクスデータを収集して分析できる能力は非常に重要です。このデータは、運用、製品、およびビジネス戦略の策定に使用されます。
- 管理者ユーザー管理 — SaaS システムは、テナントユーザーと管理者ユーザーの両方をサポートする必要があります。管理者ユーザーは SaaS プロバイダーの管理者を意味します。彼らは、カスタマーの運用エクスペリエンスにログインして、SaaS 環境の監視と管理を行います。

マルチテナンシーの再定義

マルチテナンシーと SaaS という用語は多くの場合で密接に関連しています。組織によっては、SaaS とマルチテナンシーを同じものとして説明している場合があります。これは当然のように思えるかもしれませんが、SaaS とマルチテナンシーを同一視すると、実際には SaaS はアーキテクチャ戦略というよりもビジネスモデルであるのに対し、チームは SaaS について純粋に技術的なものとして検討するようになります。

この概念についてより詳しく理解するため、まずはマルチテナンシーの従来の考え方について確認しましょう。インフラストラクチャのみに焦点を当てたこの考え方では、マルチテナンシーは、俊敏性とコスト効率の向上のため、テナントがリソースをどのように共有するかについて説明するのに使用されます。

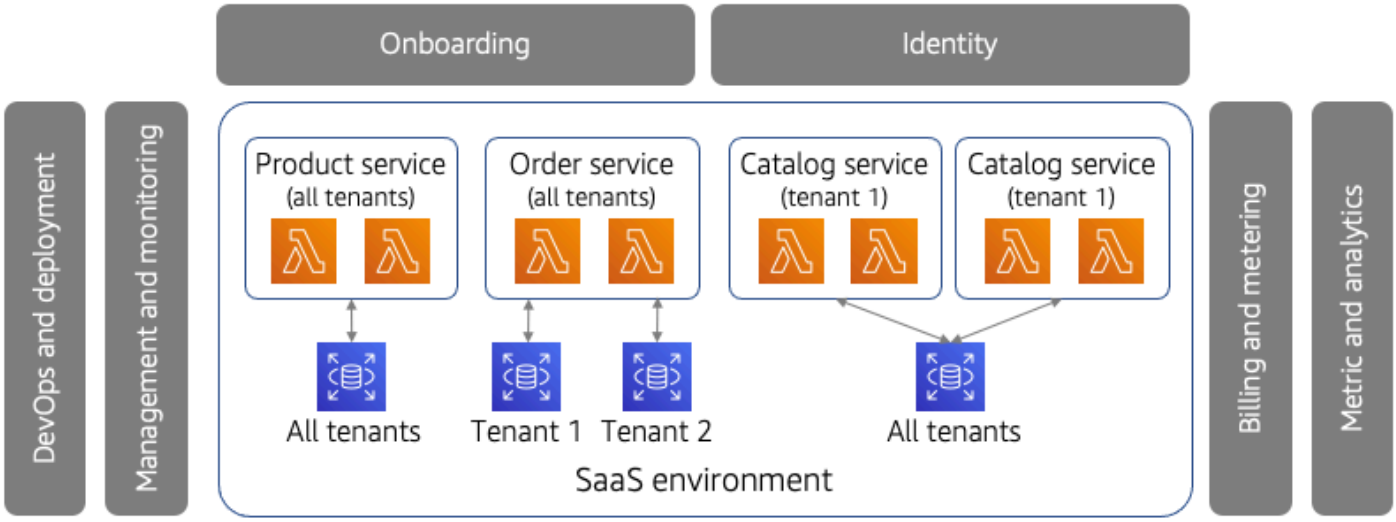
例えば、SaaS システムの複数のテナントによって消費されるマイクロサービスまたは Amazon Elastic Compute Cloud (Amazon EC2) インスタンスが存在するとします。テナントはこのサービスを実行するインフラストラクチャを共有しているため、このサービスはマルチテナントモデルで実行されていると見なされます。

この定義の課題は、マルチテナンシーの技術的な概念をあまりにも直接的に SaaS と結び付ける点にあります。SaaS の最大の特徴は、共有のマルチテナントインフラストラクチャを備えていることだと想定しています。SaaS に対するこの考え方は、さまざまな環境で SaaS が実現される多くの方法について検討したとき、確立できなくなります。

次の図は、マルチテナンシーを定義する上で直面するいくつかの課題について示した SaaS システムの図です。

ここでは、前述の典型的な SaaS モデルを示しています。一連のアプリケーションサービスが、テナントをまとめて管理および運用できる共有サービスに囲まれています。

新しく追加されたのはマイクロサービスです。この図には、製品、注文、カタログの 3 つのサンプルマイクロサービスが含まれています。これらの各サービスのテナンシーモデルをよく見ると、すべてわずかに異なるテナントパターンを採用していることがわかります。



SaaS とマルチテナンシー

製品サービスは、すべてのリソース (コンピューティングとストレージ) をすべてのテナントと共有します。これは、従来のマルチテナンシーの定義と一致しています。ただし、注文サービスを見てみると、コンピューティングは共有されているものの、テナントごとにストレージが分かれていることがわかります。

カタログサービスには、コンピューティングはテナントごとに個別であり (テナントごとに個別のマイクロサービスのデプロイ)、ストレージはすべてのテナントで共有という別のバリエーションが追加されています。

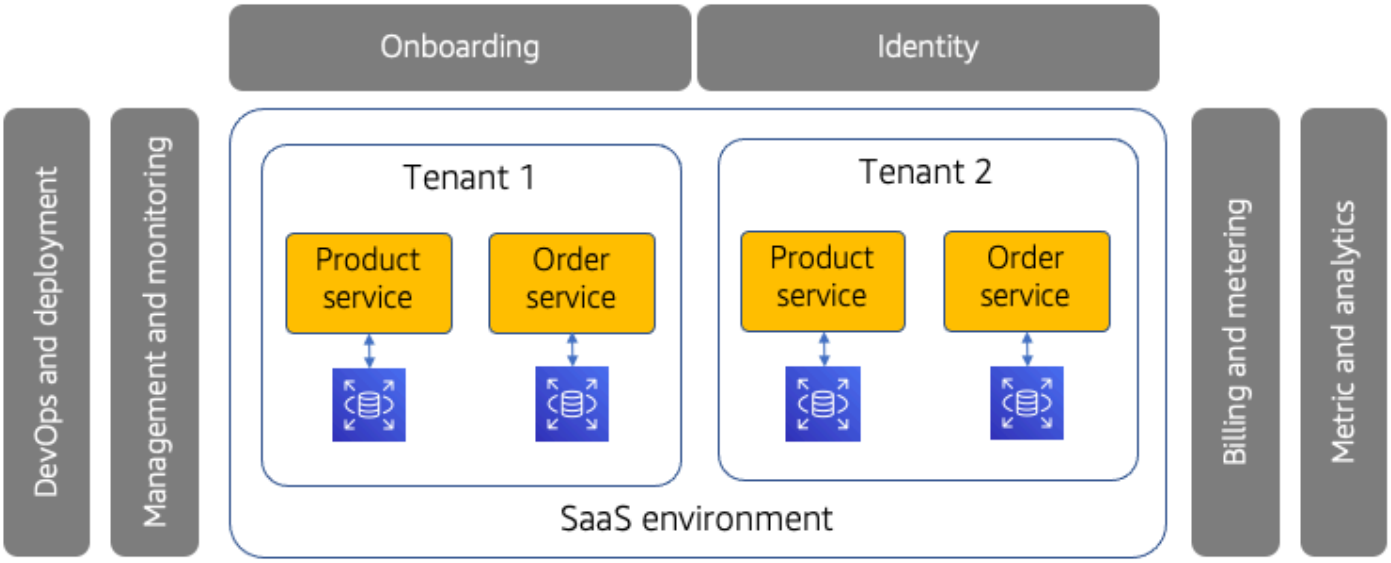
この種のバリエーションは SaaS 環境では一般的です。SaaS ソリューションの一部を選択的に共有したり、サイロ化したりする理由には、ノイジーネイバー、階層化モデル、分離の必要性などがあります。

これらのバリエーションやその他多くの可能性を考慮すると、この環境を特徴づけるためにマルチテナントという用語をどのように使用すべきかを判断するのがさらに難しくなります。全体的に見て、カスタマーから見ると、これはマルチテナント環境です。ただし、最も技術的な定義を使用すると、この環境の一部はマルチテナントであり、別の一部はマルチテナントではありません。

これが、マルチテナントという用語を使用して SaaS 環境を特徴づけるのをやめるべき理由です。アプリケーション内でマルチテナンシーを実装する方法について説明することはできますが、ソリューションを SaaS として特徴づけるために使用することは避けてください。マルチテナントという用語は、アーキテクチャの一部が共有される場合とそうでない場合があることは分かっているため、SaaS 環境全体をマルチテナントとして説明する場合に使用する方が理にかなっています。概して、それでもこの環境はマルチテナントモデルで運用および管理されています。

極端なケース

テナンシーという概念をよりわかりやすくするために、テナントがリソースを共有しない SaaS モデルについて検討してみましょう。次の図は、一部の SaaS プロバイダーで採用されている SaaS 環境のサンプルを示しています。



テナントごとのスタック

この図を見ると、これらのテナントの周りには共通の環境がまだ存在することがわかります。ただし、各テナントには専用のリソースのコレクションが配置されます。このモデルでは、テナント間で共有されるものはありません。

この例は、マルチテナントであることの意味に疑問を投げかけています。どのリソースも共有されていないのに、これはマルチテナント環境なのでしょうか。このシステムを利用するテナントは、リソースを共有する SaaS 環境と同じ内容を期待しています。実際、SaaS 環境内でリソースがどのようにデプロイされているかについてらない可能性があります。

これらのテナントはサイロ化されたインフラストラクチャで運営されていますが、それでも一元的に管理および運営されています。オンボーディング、ID、メトリクス、請求、運用に関して統一されたエクスペリエンスを共有します。また、新しいバージョンがリリースされると、すべてのテナントにデプロイされます。これを実行するため、個々のテナントに 1 回限りのカスタマイズを許可することはできません。

ここで示すのは極端な例ですが、マルチテナント SaaS の概念をテストする適切なモデルです。共有インフラストラクチャの効率性をすべて実現しているわけではありませんが、完全に有効なマル

チテナント SaaS 環境です。一部のカスタマーでは、ドメインによって、一部のカスタマーまたはすべてのカスタマーがこのモデルで実行することが規定されている場合があります。だからといって、SaaS ではない、というわけではありません。これらの共有サービスを使用し、すべてのテナントが同じバージョンを実行している場合は、SaaS の基本原則に沿っています。

こうしたパラメータと SaaS のより広い定義について考えると、マルチテナントという用語の使い方を進化させる必要があることがわかります。集約して管理および運用されている SaaS システムをマルチテナントと呼ぶ方が理にかなっています。そうすれば、SaaS ソリューションの実装におけるリソースの共有化または専用化の方法については、より詳細な技術用語で説明できます。

シングルテナントという用語の削除

マルチテナントという用語の使用の一環として、SaaS 環境を説明するためにシングルテナントという用語を使用したいと思うのは当然のことです。ただし、前述の背景を考えると、シングルテナントという用語は混乱を招きます。

前のトピックの図 (テナントごとのスタック) はシングルテナント環境でしょうか。各テナントには技術的には独自のスタックがありますが、これらのテナントは依然としてマルチテナントモデルで運用および管理されています。これが、一般的にシングルテナントという用語が避けられる理由です。すべての環境は、マルチテナントとして特徴付けられます。これは、リソースの一部またはすべてが共有化または専用化されている、一部のテナンシーのバリエーションを実装しているだけだからです。

サイロとプールの紹介

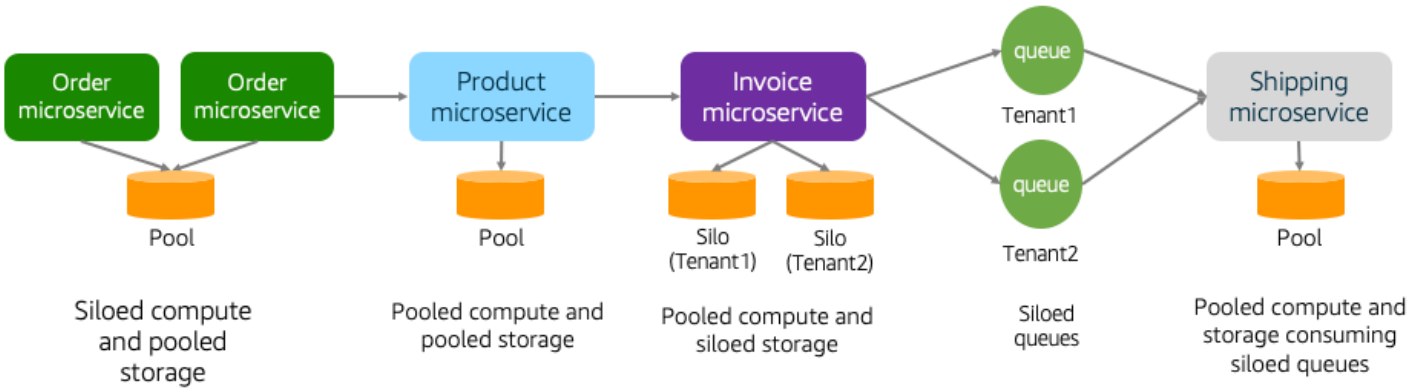
これらすべてのモデルのバリエーションと、マルチテナンシーという用語に関する課題を考慮し、SaaS を構築する際に使用されるさまざまなモデルをより正確に把握し、説明するための用語をいくつか紹介しました。

SaaS 環境でのリソースの使用を特徴付ける用語には、サイロとプールの 2 つがあります。これらの用語を使って SaaS 環境の性質にラベルを付けることができます。マルチテナントは、基礎となるあらゆる数のモデルに適用できる包括的な説明として使用します。

最も基本的なレベルでは、サイロという用語は、リソースが特定のテナントに専用化されるシナリオを指します。逆に、プールモデルは、テナントがリソースを共有するシナリオを表すために使用されます。

サイロとプールという用語がどのように使用されるかについて検討するとき、サイロとプールは絶対的な概念ではないことを明確にておくことが重要です。サイロとプールは、テナントのリソーススタック全体に適用することも、SaaS 環境全体の一部に選択的に適用することもできます。したがって、あるリソースがサイロモデルを使用していると言っても、その環境のすべてのリソースがサイロ化されているわけではありません。プールという用語の使い方についても同じことが言えます。

次の図は、SaaS 環境でサイロ化されたモデルとプールされたモデルをより詳細なレベルで使用する方法の例を示しています。



サイロモデルとプールモデル

この図には、サイロモデルとプールモデルの性質について、より的を絞って説明するための一連のサンプルが含まれています。これを左から右にたどると、注文マイクロサービスから開始されていることがわかります。このマイクロサービスでは、コンピューティングはサイロ化し、ストレージはプール化しています。これはコンピューティングとストレージがプール化された製品サービスと相互作用します。

次に、製品サービスは、プール化されたコンピューティングとサイロ化されたストレージを備えた請求書マイクロサービスとやり取りします。このサービスは、キューを介して配送サービスにメッセージを送信します。キューはサイロ化されたモデルで展開されます。

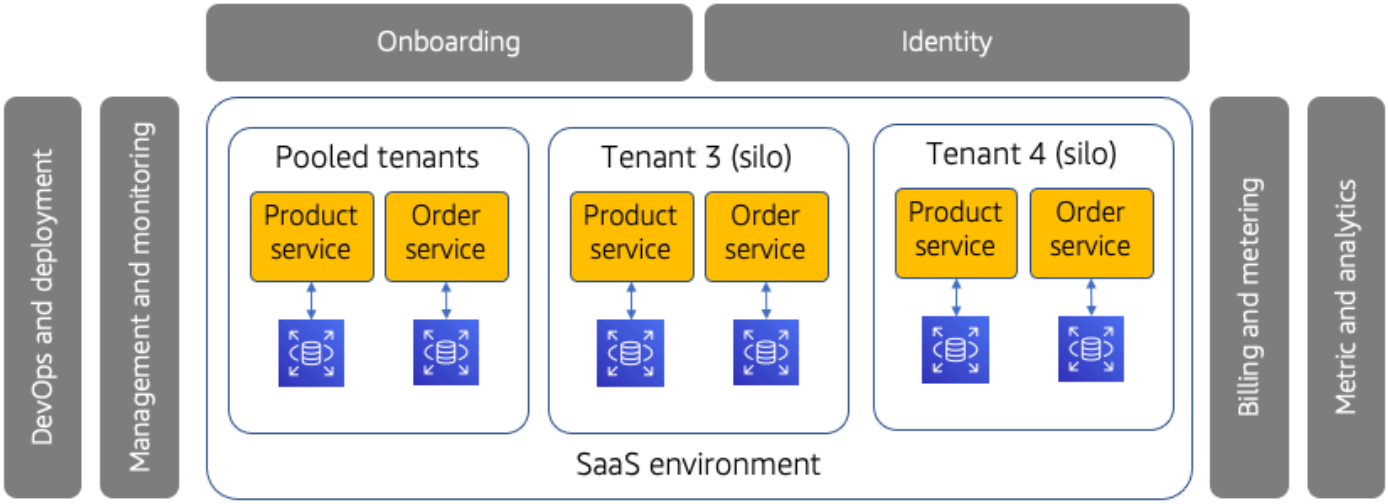
最後に、配送マイクロサービスはサイロ化されたキューからメッセージを取得します。これはプール化されたコンピューティングとストレージを使用します。

これは少し複雑に思えるかもしれませんが、サイロとプールの概念のより詳しい性質について明らかにすることを目的としています。SaaS ソリューションの設計と構築を検討する際には、ドメインとカスタマーのニーズに基づいて、このようなサイロ化やプール化に関する意思決定を行うことが予想されます。

サイロモデルやプールモデルを適用する方法やタイミングは、ノイジーネイバー、分離、階層化、その他さまざまな理由に影響する可能性があります。

フルスタックのサイロとプール

サイロとプールという用語を使用して SaaS スタック全体を記述することもできます。このアプローチでは、テナントのすべてのリソースを専用または共有の方法でデプロイします。次の図は、これが SaaS 環境にどのように適用されるかを示す例です。



フルスタックのサイロモデルとプールモデル

この図から、フルスタックのテナントデプロイには 3 つの異なるモデルが存在することがわかります。まず、フルスタックのプール環境が存在することがわかります。このプール内のテナントは、すべてのリソース (コンピューティング、ストレージなど) を共有します。

表示されている他の 2 つのスタックは、フルスタックのサイロ化されたテナント環境を表しています。この場合、テナント 3 とテナント 4 はそれぞれ専用のスタックを持ち、他のテナントとリソースを共有していません。

同じ SaaS 環境でサイロモデルとプールモデルが混在していることは、それほど珍しいことではありません。例えば、システムを使用するのに適度な料金を支払う基本階層のテナントがあるとします。これらのテナントはプール化された環境に置かれます。

一方、サイロで運用される特権にさらに料金を支払っても構わないと思っているプレミアム層のテナントが存在する場合もあります。これらのカスタマーは、(図のように) 別々のスタックでデプロイされます。

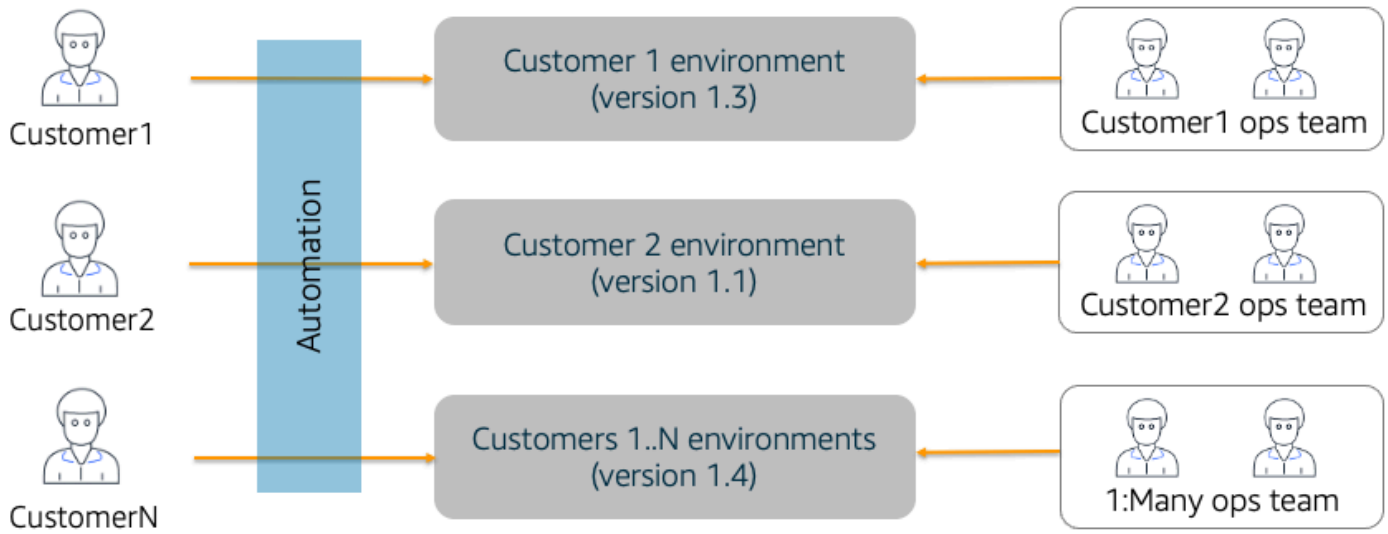
テナントが独自のフルスタックのサイロで運用されることを許可しているこのモデルでも、これらのサイロでは、これらのテナントに対して一回限りの変更やカスタマイズを許可しないことが不可欠で

す。すべにおいて、これらのスタックでは、同じバージョンのソフトウェアを使用して、同じ構成のスタックを実行している必要があります。新しいバージョンがリリースされると、プールされたテナント環境とサイロ化された各環境にデプロイされます。

SaaS とマネージドサービスプロバイダー (MSP) の比較

また、SaaS モデルとマネージドサービスプロバイダー (MSP) モデルの境界については、一部混乱が生じています。MSP モデルについて検討するとき、SaaS モデルと似たような目標があるように思えるかもしれません。

しかし、MSP についてもう少し掘り下げてみると、MSP と SaaS は実際には異なることがわかります。次の図は、MSP 環境の概念図を示しています。



マネージドサービスプロバイダー (MSP) モデル

この図は、MSP モデルへのアプローチの 1 つを示しています。左側には、MSP モデルを利用しているカスタマーが表示されます。一般的に、ここでのアプローチは、利用可能なオートメーションをすべて使用して各カスタマーに対して環境をプロビジョニングし、ソフトウェアをインストールするというものです。

右側は、MSP がこれらのカスタマー環境をサポートするために提供する運用上のフットプリントの概算です。

注意すべき重要な点は、MSP は多くの場合、特定のカスタマーが実行したいバージョンの製品をインストールして管理しているということです。すべてのカスタマーが同じバージョンを実行している可能性がありますが、これは通常 MSP モデルでは必要ありません。

一般的には、これらの環境のインストールと管理を担当することで、ソフトウェアプロバイダーの業務を簡素化することを戦略とします。これによってプロバイダーの業務はシンプルになりますが、SaaS サービスに不可欠な価値観や考え方にこれが直接当てはまるわけではありません。

焦点は管理責任の軽減にあります。この移行を行うことと、すべてのカスタマーに同じバージョンで統一された管理と運用エクスペリエンスを提供することは同じではありません。多くの場合、MSP は、個別のバージョンを許可し、これらの環境をそれぞれ運用上別個のものとして扱うことがあります。

確かに、MSP が SaaS と重複し始める可能性のある分野は存在します。MSP が基本的にすべてのカスタマーに同じバージョンを実行することを要求し、MSP が単一のエクスペリエンスですべてのテナントのオンボーディング、管理、運用、請求を一元的に行うことができれば、これは MSP と言うよりも SaaS に近くなります。

より大きなテーマとして言うならば、環境のインストールの自動化と SaaS 環境の保有はイコールではないということです。前に説明した他の注意点をすべて追加することで、初めてこれは真の SaaS モデルとなります。

この話をテクノロジーおよびオペレーションの側面から戻すと、MSP と SaaS の境界線はさらに明確になります。一般的に、SaaS ビジネスにとって、提供するサービスの成功は、エクスペリエンスのあらゆる可動部分に深く関与できるかどうかにかかっています。

これは通常、オンボーディングエクスペリエンスの状態を把握し、運用イベントがテナントに与える影響を理解し、主要なメトリクスや分析情報を追跡し、カスタマーとの距離を縮めることです。MSP モデルでは、これを他の人に引き継ぐため、SaaS ビジネス運用の中核となる重要な詳細部分から少し離れてしまう可能性があります。

SaaS への移行

SaaS を採用するプロバイダーの多くは、(前述の) 従来のインストール済みソフトウェアモデルから SaaS に移行します。これらのプロバイダーにとって、SaaS の中核となる原則をしっかりと理解することは特に重要です。

ここでもやはり、SaaS モデルへの移行が意味することについて混乱が生じる可能性があります。例えば、クラウドへの移行を SaaS への移行と見なす人もいるでしょう。また、インストールとプロビジョニングのプロセスを自動化することで移行を実現できると考える人もいます。

組織によってスタート地点が異なり、レガシーに関する考慮事項も異なり、市場や競争圧力も異なる可能性が高いと言っても過言ではありません。つまり、それぞれの移行は見え方が異なります。

それでも、それぞれのパスは異なるものの、移行戦略を形成する中核的な原則をめぐって、一部の工リアでは話がつながらない部分があります。概念と原則をうまく調整することは、SaaS への移行の全体としての成功に大きな影響を与える可能性があります。

前に概説した概念から、SaaS への移行は、ビジネス戦略と目標から始めることが明らかです。できるだけ早期に SaaS に移行しなければならないというプレッシャーがある移行環境では、この点を見失う可能性があります。

この状態になると、多くの場合、組織は移行を主に技術的な作業と見なしてしまいます。現実には、SaaS への移行は、どれも対象となるカスタマー、サービスエクスペリエンス、運用目標などを明確に把握することから始める必要があります。SaaS ビジネスがどうあるべきかをより明確に把握することは、ソリューションを SaaS に移行するための形状、優先順位、および道筋に大きな影響を与えます。

移行の初期段階からこの明確なビジョンを持つことで、SaaS への移行の一環としてテクノロジーとビジネスの両方の移行について、それらの基礎を築くことができます。この道筋を歩むときは、自分が向かう先について最も詳しい情報を提供してくれる質問を重視します。

次の表は、テクノロジーの移行とビジネスの移行の考え方の対照的な性質を示しています。

表1 — テクノロジー優先の移行とビジネス優先の移行

テクノロジー優先の考え方	ビジネス優先の考え方
テナントデータをどのように分離するか。	SaaS はビジネスの成長にどのように役立つか。

テクノロジー優先の考え方	ビジネス優先の考え方
ユーザーをテナントにつなげるにはどうすればよいか。	どのセグメントをターゲットにしているか。
ノイジーネイバーの状況をどのように回避するか。	これらのセグメントのサイズとプロファイルは。
A/B テストはどうすればよいか。	どの階層をサポートする必要があるか。
テナントの負荷に基づいてどのようにスケーリングすればよいか。	どのようなサービスエクスペリエンスをターゲットにしているか。
どの請求プロバイダーを使うべきか。	価格設定とパッケージ戦略は何か。

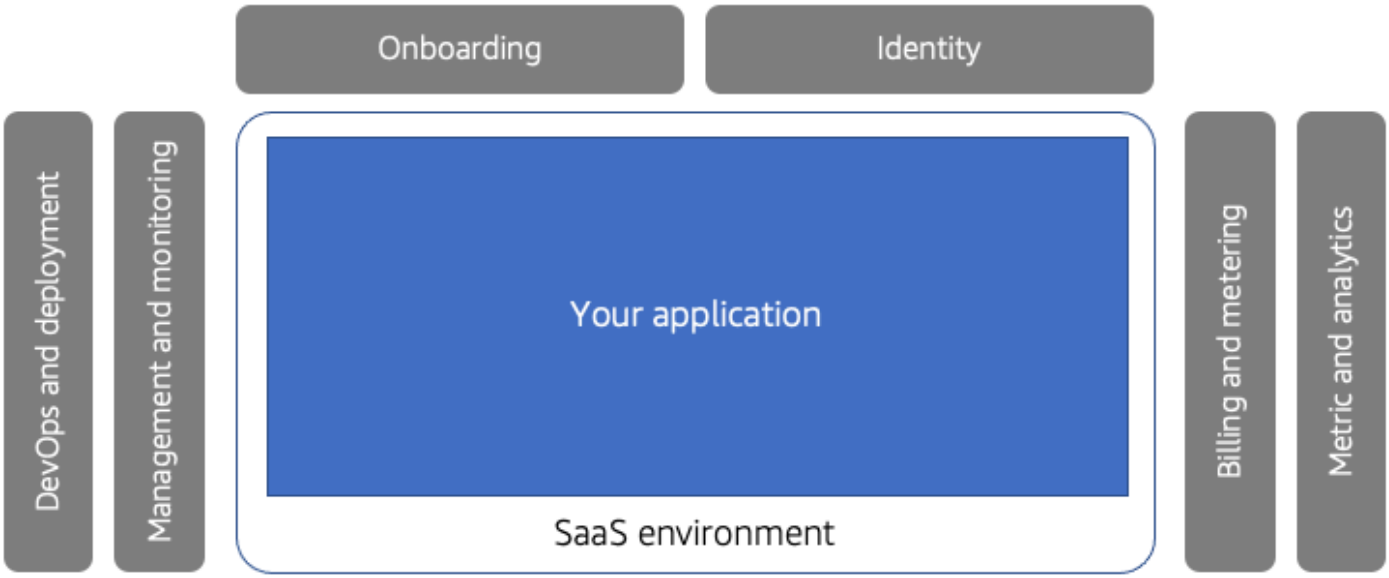
左側に示しているのは、テクノロジー優先の移行の具体的な例です。エンジニアリングチームは、従来のマルチテナントのトピックを追求するのに全力を注ぎます。これは確かにどの SaaS アーキテクチャにとっても重要です。

ただし問題なのは、左側の質問に対する多くの回答は、右側の質問に対する回答に直接影響されることが多いことです。移行を検討している人にとって、これは目新しいことではないでしょう。しかし、現実として、多くの組織では、ビジネスの部分は機能するだろうと想定し、最初のステップとして運用効率とコスト効率の追求から始めています。

この移行戦略では、レガシー環境をどのように進化させて SaaS モデルに適合するかについても混乱が生じる可能性があります。この分野でも、SaaS への移行にはさまざまな選択肢があります。しかし、どの移行に対しても一般的に提唱している共通の価値体系があります。

SaaS の原則についての前述の説明では、SaaS 環境の説明に使用されるさまざまなパターンと用語について概説しました。これらのソリューションすべてに共通するテーマの 1 つは、アプリケーションを囲む共有サービスを持つという考えです。ID、オンボーディング、メトリクス、請求など、これらはすべて SaaS 環境の中核となる共通要素として挙げられています。

さて、移行について検討するとき、これらと同じ共有サービスは、あらゆる移行シナリオで重要な役割を果たしていることがわかります。次の図は、移行先環境の概念図を示しています。



SaaS への移行

この図は、あらゆる移行パスのターゲットエクスペリエンスを表しています。これには、前に説明したのと同じ共有サービスがすべて含まれています。中央にはアプリケーションのプレースホルダーがあります。

重要なのは、この環境の真ん中には、アプリケーションモデルをいくつでも配置できるということです。移行の最初のステップでは、各テナントがそれぞれ独自のサイロで運用している場合があります。あるいは、何らかのハイブリッドアーキテクチャがあり、要素がサイロ化され、他の機能は最新のマイクロサービスのコレクションを通じて処理されている場合もあります。

最初にアプリケーションをどの程度モダナイズするかは、レガシー環境の性質、市場のニーズ、コストに関する考慮事項などによって異なります。変わらないのは、これらの共有サービスの導入です。

どのような SaaS への移行でも、ビジネスが SaaS モデルで運用できるように、これらの基礎的な共有サービスをサポートする必要があります。例えば、アプリケーションアーキテクチャのすべてのバリエーションで SaaS ID が必要です。SaaS ソリューションを管理および監視するには、テナント対応のオペレーションが必要です。

これらの共有サービスを移行の最初に導入することで、基盤となるアプリケーションがテナントごとにフルスタックのサイロで運用されている場合でも、カスタマーに SaaS エクスペリエンスを提供できます。

一般的な目標は、アプリケーションを SaaS モデルで実行することです。そうすれば、アプリケーションのさらなるモダナイゼーションと改良に目を向けることができます。また、このアプローチで

は、ビジネスの他の部分 (マーケティング、販売、サポートなど) をより速いペースで移動させることも可能です。さらに重要なのは、これによってカスタマーからのフィードバックをエンゲージさせて収集し、それを活用して環境の継続的なモダナイゼーションを形成できることです。

導入する共有サービスには、最終的に必要となる機能やメカニズムがすべて含まれていない場合があることに注意してください。主な目標は、移行の初期段階で必要な共有メカニズムを作成することです。これにより、アプリケーションアーキテクチャの進化および運用の進化にとって不可欠なシステムの要素に集中できるようになります。

SaaS の ID

SaaS では、アプリケーションの ID モデルについて新たに考慮する必要があります。各ユーザーは認証されたら、特定のテナントコンテキストに関連付ける必要があります。このテナントコンテキストは、SaaS 環境全体で使用されるテナントに関する重要な情報を提供します。

このテナントとユーザーのバインディングは、アプリケーションの SaaS ID と呼ばれることがよくあります。各ユーザーが認証されると、通常、ID プロバイダーはユーザー ID とテナント ID の両方を含むトークンを生成します。

テナントとユーザーをつなぐことは、SaaS アーキテクチャの基本的な側面であり、ダウンストリームに多くの影響を及ぼします。この ID プロセスのトークンはアプリケーションのマイクロサービスに送られ、テナントごとに識別可能なログの作成、メトリクスの記録、課金の計測、テナント分離の実施などに使用されます。

ユーザーをテナントにマッピングする個別のスタンドアロンメカニズムに依存するシナリオは、回避することが不可欠です。これは、システムのセキュリティを損ない、多くの場合、アーキテクチャにボトルネックを形成します。

テナント分離

カスタマーをマルチテナントモデルに移行すればするほど、あるテナントが別のテナントのリソースにアクセスする可能性について懸念が増すようになります。SaaS システムには、たとえ共有インフラストラクチャ上で運用されていても、各テナントのリソースが確実に分離される明確なメカニズムが搭載されています。

これをテナント分離と呼びます。テナント分離の背景にある考え方は、SaaS アーキテクチャにリソースへのアクセスを厳密に制御する仕組みを導入し、別のテナントのリソースにアクセスしようとする試みをブロックするというものです。

テナント分離は一般的なセキュリティメカニズムとは別のものであることに注意してください。システムは認証と認可をサポートしますが、テナントユーザーが認証されたとしても、システムが分離されたわけではありません。分離は、アプリケーションの一部となっている可能性のある基本認証や認可とは別に適用されます。

これをより詳しく理解するために、ID プロバイダーを使用して SaaS システムへのアクセスを認証した場合を想像してみてください。この認証で得られたトークンには、特定のアプリケーション機能へのユーザーアクセスを制御するのに使用できるユーザーロールに関する情報も含まれる場合があります。これらが提供するのは、セキュリティであり、分離ではありません。実際、ユーザーは認証と認可を受けても、別のテナントのリソースにアクセスすることができます。認証や認可についても、必ずしもこのアクセスをブロックするものはありません。

テナント分離は、テナントコンテキストを使用してリソースへのアクセスを制限することにのみ重点を置いています。現在のテナントコンテキストを評価し、そのコンテキストを使用して、そのテナントがどのリソースにアクセスできるかを判断します。この分離は、そのテナント内のすべてのユーザーに適用されます。

さまざまな SaaS アーキテクチャパターンでテナント分離がどのように実現されているかについて調べると、これはさらに困難になります。場合によっては、リソースのスタック全体をテナント専用にして、ネットワーク (またはより粗い粒度の) ポリシーによりテナント間のアクセスを阻止することで、分離を実現できます。他のシナリオでは、リソースへのアクセスを制御するためのより細かい粒度のポリシーを必要とするリソース (Amazon DynamoDB テーブル内のアイテム) をプール化している場合があります。

テナントリソースにアクセスしようとする試みはすべて、そのテナントに属するリソースのみを対象とする必要があります。特定のアプリケーションの分離要件をサポートするツールとテクノロジーの組み合わせを決定するのは、SaaS 開発者とアーキテクトの仕事です。

データのパーティション化

データのパーティション化は、マルチテナント環境でデータを表現するために使用されるさまざまな戦略について説明するのに使用されます。この用語は、さまざまなデータ構造を個々のテナントに関連付けるために使用できる幅広いアプローチやモデルを表すために広く使用されています。

注意すべきなのは、データのパーティション化とテナント分離は交換可能な用語とみなしがちなことです。しかし、これら 2 つの概念は同等ではありません。データのパーティション化とは、個々のテナントに対してテナントデータがどのように保存されるかということです。データをパーティション化しても、データが確実に分離されるわけではありません。ただし、あるテナントが別のテナントのリソースにアクセスできないようにするには、分離を個別に適用する必要があります。

各 AWS ストレージ技術には、データのパーティション化戦略に関する独自の考慮事項があります。例えば、Amazon DynamoDB でデータを分離することは、Amazon Relational Database Service (Amazon RDS) でデータを分離することとは大きく異なります。

一般的に、データのパーティション化について考えるときは、まずデータをサイロ化するのか、プールするのかについて考えることから始めます。サイロ化されたモデルでは、データが混在することなく、テナントごとに異なるストレージコンストラクトがあります。プール化されたパーティショニングでは、どのデータを各テナントに関連付けるかを決定するテナント識別子に基づいて、データは混在してパーティション化されます。

一例として、Amazon DynamoDB では、サイロ化されたモデルではテナントごとに個別のテーブルを使用します。Amazon DynamoDB にデータをプールするには、すべてのテナントのデータを管理する各 Amazon DynamoDB テーブルのパーティションキーにテナント識別子を格納します。

これが各種 AWS サービスによってどのように異なるかは想像できるでしょう。各サービスが独自のコンストラクトを導入しているため、サービスごとにサイロやプール化ストレージモデルを実現するためのアプローチが異なる場合があります。

データのパーティション化とテナント分離は異なるトピックですが、選択するデータのパーティション化戦略は、多くの場合データの分離モデルによる影響を受けます。例えば、そのアプローチがドメインやカスタマーの要件に最も適合するため、一部のストレージをサイロ化することがあります。また、プールモデルでは、ソリューションに必要な詳細度レベルで分離を実施できない場合があるため、サイロを選択することがあります。

また、分離へのアプローチは、ノイジーネイバーによる影響を受ける可能性があります。他のテナントからの影響を抑えるため、またはサービスレベルアグリーメント (SLA) を満たすために、アプリケーションで一部のワークロードやユースケースの分離が必要な場合があります。

計測、メトリクス、および請求

SaaS を議論するとき、計測、メトリクス、請求という概念も含まれる傾向があります。これらの概念は、多くの場合 1 つの概念にまとめられます。ただし、SaaS 環境で計測、メトリクス、請求が果たすさまざまな役割を区別することは重要です。

これらの概念の難点は、同じ単語が重複して使用されることが多いことです。例えば、請求書の生成に使用される計測などです。また、同時に、請求とは関係のないリソースの内部使用量を追跡するために使用される計測もそうです。また、メトリクスと SaaS についてもさまざまな文脈で取り上げられていますが、これらがこの議論に混同される可能性もあります。

こうした問題を解決するため、これらの各用語に対していくつかの特定の概念を関連付けてみましょう (ただしこれについて絶対的なものはありません)。

- 計測 — この概念には多くの定義がありますが、SaaS の請求ドメインに最も適しています。テナントのアクティビティやリソースの使用量を計測して、請求書の作成に必要なデータを収集するという考え方です。
- メトリクス — メトリクスとは、ビジネス、オペレーション、技術分野の傾向を分析するために収集したすべてのデータを表します。このデータは、SaaS チーム内のさまざまなコンテキストやロールで使用されます。

この区別は重要ではありませんが、SaaS 環境における計測とメトリクスの役割についての考え方をシンプルにするうえで役立ちます。

さて、これら 2 つの概念を例につなげると、請求書の作成に必要なデータを明確にする特定の計測イベントを使ってアプリケーションを計装化することを検討できます。これには、リクエスト数やアクティブユーザー数、またはカスタマーにとって合理的な単位と相関する使用量 (リクエスト、CPU、メモリ) の集計値などが含まれます。

SaaS 環境では、これらの請求イベントをアプリケーションから公開し、SaaS システムで採用されている請求コンストラクトに取り込まれ、適用されます。これは、サードパーティの請求システムまたはカスタムシステムです。

対照的に、メトリクスは、さまざまなテナントがシステムに課している健全性や運用上のフットプリントを評価するために不可欠なアクション、アクティビティ、消費パターンなどを把握することに基づいています。ここで発行して集計するメトリクスは、さまざまなペルソナ (運用チーム、プロダクトオーナー、アーキテクトなど) のニーズによって決定されます。ここでは、このメトリクスデータが発行され、いくつかの分析ツールに集約されます。これにより、さまざまなユーザーがシステムア

クティビティのビューを構築して、自分のペルソナに最も合ったシステムの側面を分析できます。プロダクトオーナーは、さまざまなテナントがどのように機能を利用しているかを知りたいと考えるかもしれません。また、アーキテクトは、テナントがインフラストラクチャリソースをどのように使用しているかなどを理解するのに役立つ見解が必要な場合があります。

B2B および B2C SaaS

SaaS サービスは、B2B 市場と B2C 市場の両方に向けて作成されています。市場とカスタマーのダイナミクスが異なることは確かですが、SaaS の全体的な原則がこれらの各市場を何らかの形で変化させることはありません。

例えば、オンボーディングについて検討するとき、B2B と B2C のカスタマーではオンボーディングの経験が異なる場合があります。B2C システムでは、確かにセルフサービスのオンボーディングフローに重点を置いているかもしれません (ただし、B2B システムでこれがサポートされている場合もあります)。

カスタマーへのオンボーディングの提供方法には違いがあるかもしれませんが、オンボーディングの基本的な価値はほぼ同じです。B2B ソリューションが社内のオンボーディングプロセスに依存している場合でも、そのプロセスは可能な限りスムーズで自動化されていることが期待されます。B2B であるからといって、カスタマーの価値に対するタイムトゥバリューについて、期待する内容が変わることはありません。

結論

このドキュメントの目的は、SaaS のパターンと戦略を特徴づけるために使用されるモデルと用語を明確にし、基本的な SaaS アーキテクチャの概念を概説することです。これにより、組織が SaaS 全体の状況をより正確に把握できるようになることが期待されます。

ここで説明する内容の多くは、SaaS であることが何を意味するのかに焦点を当てており、統一された体験を通じてすべての SaaS テナントを管理および運用できる環境の作成に重点が置かれています。これは、SaaS が何よりもまずビジネスモデルであるという核となる概念につながっています。構築する SaaS アーキテクチャは、これらの基本的なビジネス目標を促進することを目的としています。

詳細情報

ここで説明するパターンに準拠する SaaS アーキテクチャパターンについて、多くのリソースでさらに詳しく説明されています。

詳細については、次を参照してください。

- SaaS テナント分離戦略 (AWS ホワイトペーパー)
- SaaS ストレージ戦略 (AWS ホワイトペーパー)
- Well-Architected SaaS レンズ (AWS ホワイトペーパー)

寄稿者

このドキュメントには、次の個人および組織が貢献しました。

- Tod Golding、Principal Partner Solutions Architect、AWS SaaS Factory

ドキュメントの改訂

このホワイトペーパーの更新に関する通知を受け取るには、RSS フィードにサブスクライブしてください。

変更	説明	日付
初版発行	ホワイトペーパーの発行。	2022 年 8 月 3 日

注意

お客様は、本書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的とし、(b) AWS の現行製品と慣行について説明しており、これらは予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤー、またはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または黙示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は AWS 契約によって規定されます。本書は、AWS とお客様との間で締結されるいかなる契約の一部でもなく、その内容を修正するものでもありません。

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。