SaaS ストレージ戦略

AWS でのマルチテナントストレージモデルの構築

2016 年 11 月初回発行

2021 年 5 月 6 日、技術的な正確性について更新

注意

お客様は、この文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。このドキュメントは、(a) 情報提供のみを目的としており、(b) AWS の現行製品とプラクティスを表したものであり、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義務や確約を意味するものではありません。AWS の製品やサービスは、明示または暗示を問わず、いかなる保証、表明、条件を伴うことなく「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

目次

要約

マルチテナントストレージは、Software as a Service (SaaS) ソリューションの構築と提供する上で困難な側面の 1 つです。テナントデータを分割するために使用できるさまざまな戦略があり、戦略ごとの固有のニュアンスを反映してマルチテナンシーへのアプローチを構成します。この複雑さに加えて、Amazon DynamoDB、Amazon Relational Database Service (Amazon RDS)、Amazon Redshift など、AWS が提供するさまざまなストレージモデルに、これらの各戦略をマッピングする必要があります。これらのテクノロジーに広く適用できる高レベルのテーマはありますが、各ストレージモデルには、マルチテナント環境でのデータのスコープ、管理、セキュリティに関する独自のアプローチがあります。このホワイトペーパーでは、SaaS デベロッパーに対して、データのパーティション分割に関する広範なオプションへのインサイトを提供し、SaaS 環境のニーズに最も適合する戦略とストレージテクノロジーの組み合わせを見極めることができるようにします。

はじめに

AWS は、Software as a Service (SaaS) デベロッパーに対して広範なストレージソリューションを提供します。ストレージソリューションごとに、データのスコープ、プロビジョニング、管理、保護に関する独自のアプローチがあります。サービスごとにデータの表現、インデックス作成、保存を行う方法によって、マルチテナント戦略に固有の考慮事項が追加されます。SaaS デベロッパーにとって、これらのストレージオプションの多様性は、SaaS ソリューションのストレージニーズと、ビジネスやお客様のニーズに最適なストレージテクノロジーとを適合させる機会となります。

AWS ストレージオプションを検討する場合は、SaaS ソリューションのマルチテナントモデルが、各ストレージテクノロジーとどのように適合するかを検討する必要があります。ストレージに複数の種類があるのと同様に、マルチテナントのパーティション戦略にも複数の種類があります。目標は、ストレージとテナントのパーティション分割ニーズとの最適な共通部分を見つけることです。

このホワイトペーパーでは、このパズルのすべての変動要素について解説します。マルチテナンシーを達成するために一般的に使用するモデルを調査および分類し、パーティション分割モデルごとの長所と短所を評価して最適なモデルを選択できるようにします。また、各モデルを Amazon RDS、Amazon DynamoDB、Amazon Redshift でどのようにして実現するかについても説明します。各ストレージテクノロジーを詳しく検討すると、AWS の構造を使用してマルチテナントストレージをスコープおよび管理する方法がわかります。

このホワイトペーパーでは、マルチテナントのパーティション分割戦略を選択するための一般的なガイダンスを提供します。ただし、重要な点として、各環境のビジネス面、技術面、運用面の要因がどのアプローチを選択するかに影響する場合もあります。多くの場合、SaaS 組織は、このホワイトペーパーで説明しているバリエーションを組み合わせて採用しています。

SaaS パーティション分割モデル

開始するには、明確に定義された概念モデルを通じて、さまざまな実装戦略を理解する必要があります。

図 1 は、SaaS 環境でテナントデータをパーティション分割するときに一般的に使用する 3 つの基本モデル (サイロ、ブリッジ、プール) を示しています。

パーティション分割モデルごとに、テナントデータの管理、アクセス、分離を行うアプローチは大きく異なります。以下のセクションでは、各モデルを簡単に紹介し、特定のストレージテクノロジーのコンテキストを離れて、各モデルの原則や概念を検討します。

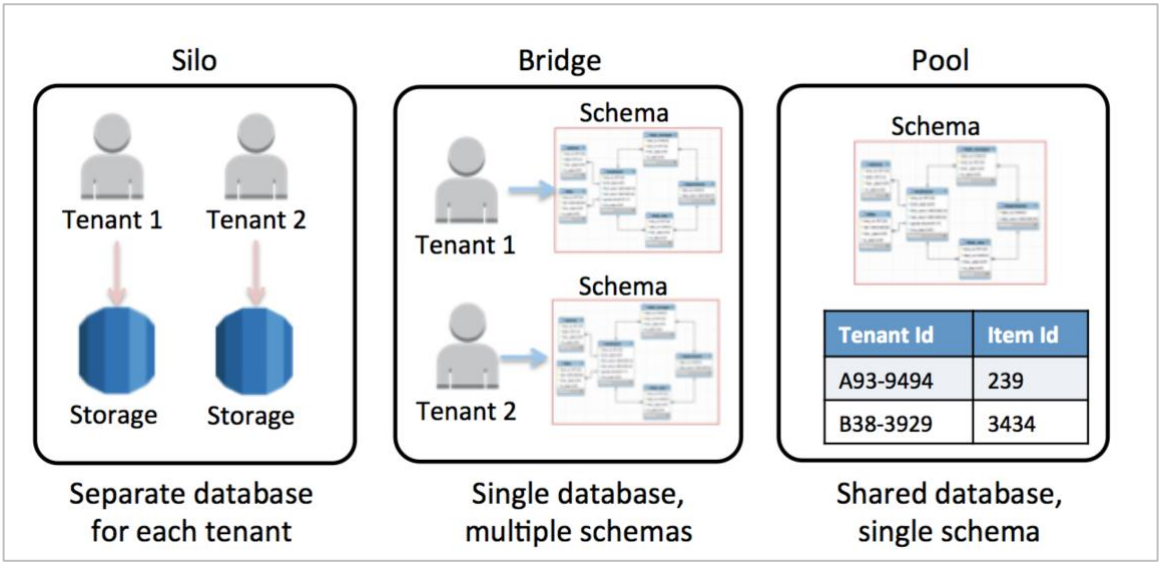

図 1 - SaaS パーティション分割モデル

サイロモデル

サイロモデルでは、テナントデータのストレージが他のテナントデータから完全に分離されます。テナントデータを表現するために使用するすべての構造は、クライアントにとって論理的に「一意」であると見なされます。つまり、多くの場合、テナントごとに表現、

モニタリング、管理、セキュリティのフットプリントを持ちます。

ブリッジモデル

通常、ブリッジモデルは、SaaS デベロッパーにとって魅力的な妥協となります。ブリッジは、すべてのテナントデータを 1 つのデータベース内に移動しますが、テナントごとのバリエーションと分離がある程度許容されます。通常、これを達成するには、テナントごとに個別のテーブルを作成します。各テーブルは、独自のデータ表現 (スキーマ) を持つことができます。

プールモデル

プールモデルは、システムのすべてのストレージ構造をテナント間で共有する包括的なマルチテナントモデルです。テナントデータは共通のデータベースに配置され、すべてのテナントが共通のデータ表現 (スキーマ) を共有します。この場合、テナントデータへスコープの定義、およびアクセス制御するためのパーティション分割キーを導入する必要があります。このモデルは、SaaS ソリューションのプロビジョニング、管理、更新を簡素化します。また、SaaS プロバイダーにとって不可欠な継続的デリバリーとアジリティの目標にも合致します。

背景の設定

サイロ、ブリッジ、プールの各モデルに基づいて、以後の議論を進めます。各 AWS ストレージテクノロジーを詳しく検討すると、これらのモデルの概念的要素を特定の AWS ストレージテクノロジーで実現する方法がわかります。これらのモデルに単純にマッピングする場合もあれば、ある程度の工夫をしてテナント分離の各タイプを実現する場合もあります。

重要な点として、これらのモデルはすべて同じように有効です。各モデルのメリットについて説明しますが、どのアプローチを最終的に選択するかは、通常、環境ごとの規制、ビジネス、レガシーの各要因に大きく左右されます。ここでの目標は、各アプローチの仕組みとトレードオフを見極めることです。

最適なモデルの特定

マルチテナントのパーティション分割ストレージモデル戦略の選択は、さまざまな要因の影響を受けます。既存のソリューションから移行する場合は、サイロモデルを使用することを検討します。このモデルを採用すると、SaaS アプリケーションを書き換えることなく、最もシンプルかつクリーンな方法でマルチテナンシーに移行できます。規制や業界の変動要因に対応するために、より一体化したモデルが必要な場合は、プールモデルの効率性とアジリティを活用して迅速かつ継続的なリリースを実現できます。ここで重要なのは、選択した戦略が、環境のビジネス面および技術面の考慮事項の組み合わせによって推進されることを認識することです。

以下のセクションでは、各モデルの長所と短所を紹介します。また、明確に定義した一連のデータポイントを提供し、より広く評価できるようにします。各モデルがアジリティの目標達成に及ぼす影響を確認します。多くの場合、アジリティは SaaS モデルを選択する際の決め手となります。SaaS 環境のアーキテクチャ戦略を選択する場合は、その戦略が、ダウンタイムのない環境でバージョンを迅速に構築、配信、デプロイする能力にどのように影響するかを検討してください。

トレードオフの評価

サイロ、ブリッジ、プールの 3 つのパーティション分割モデルを並べて見ると、これらの戦略を採用する際のトレードオフがあることがわかります。1 つのモデルの長所となっている特性は、多くの場合、別のモデルでは弱点となっています。例えば、サイロモデルの原則と価値のシステムは、プールモデルのシステムとは逆になることがよくあります。

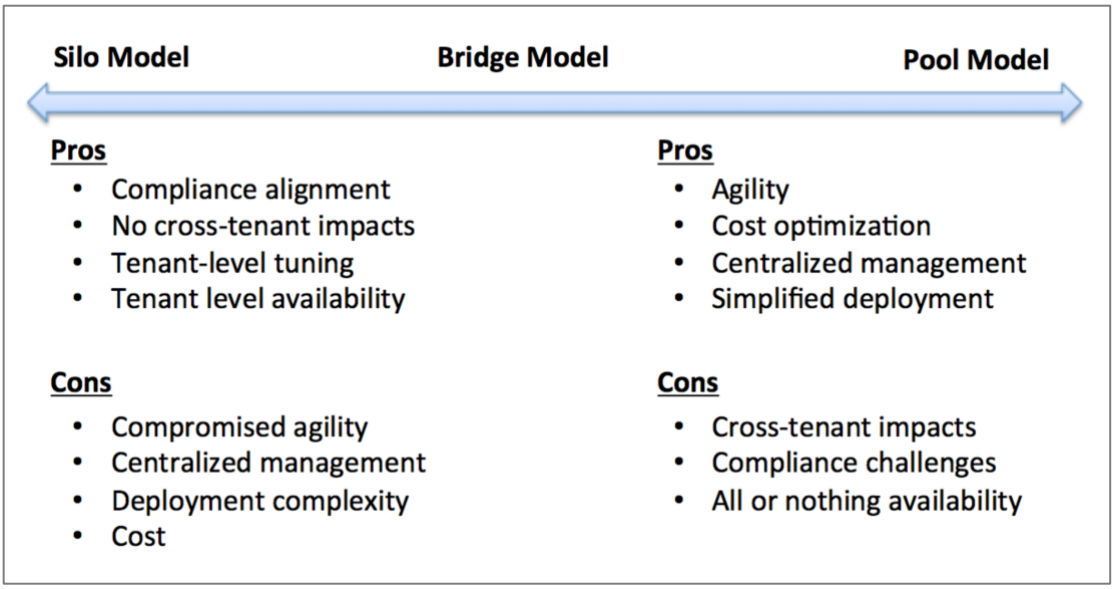

図 2 - パーティション分割モデルのトレードオフ

図 2 は、これらの競合する原則を示しています。図の上部に、3 つのパーティション分割モデルが並んでいます。左端は、サイロモデルの長所と短所を示しています。右端は、プールモデルの長所と短所を示しています。ブリッジモデルは、これら 2 つのややハイブリッドであり、両端に示している長所と短所の組み合わせになります。

サイロモデルのトレードオフ

テナントデータを複数の完全に分かれたデータベースとして表現することは魅力的です。このアプローチでは、既存のシングルテナントソリューションを簡単に移行できることに加えて、インフラストラクチャ全体を共有して運用することに懸念を示す一部のテナント

にも対処できます。

長所

サイロは厳格な規制とセキュリティの制約がある SaaS ソリューションに適している— これらの環境の場合、SaaS のお客様は、各自のデータを他のテナントからどのように分離すべきかについて非常に具体的な期待を持っています。サイロモデルを使用すると、テナントデータ間の境界を明確化するオプションをテナントに提供し、お客様に対してデータは専用モデルに保存されるという安心感を与えることができます。

クロステナントへの影響を制限できる — サイロモデルを使用して分離することで、特定のテナントのアクティビティが別のテナントに影響を与えないことを保証できます。このモデルでは、テナント別のチューニングが可能です。システムのデータベースパフォーマンス SLA を、特定のテナントのニーズに合わせて調整できます。データベースのチューニングに使用する手段により、多くの場合、サイロモデルへのマッピングはより自然なものとなり、テナント中心のエクスペリエンスを設定しやすくなります。

可用性をテナントレベルで管理し、テナントに機能停止をもたらす可能性を最小限に抑える — テナント別に独自のデータベースを使用することで、1 つのデータベースの機能停止がすべてのテナントに波及することを心配する必要がなくなります。あるテナントのデータに問題が発生しても、システムの他のテナントに悪影響が及ぶ可能性は低くなります。

短所

プロビジョニングと管理が複雑になる — テナントごとのインフラストラクチャを導入するたびに、テナントごとにプロビジョニングおよび管理する必要がある変動要素が増えます。例えば、サイロ化されたデータベースソリューションがシステムのテナントのオンボーディングエクスペリエンスにどのように影響するかを考えてください。サインアッププロセスでは、オンボーディングプロセス中にデータベースを作成および設定するオートメーションが必要になります。これは確かに実現可能ですが、SaaS 環境に複雑さの層と潜在的な障害点が追加されます。

テナントのアクティビティを確認して対応する能力が弱まる — SaaS では、すべてのテナントにわたってシステムのヘルスを確認できる管理とモニタリングが必要になる場合があ

ります。データベースのパフォーマンス問題を未然に予測し、ポリシーを広範囲に適用して対応したい場合もあります。しかし、サイロモデルを使用すると、すべてのテナントを網羅してシステム全体のヘルスビューを作成するためのツールを見つけて導入することが難しくなります。

サイロモデルの分散性は、テナント全体のパフォーマンス傾向を効果的に分析して評価する能力に影響を与える ― 各テナントが独自のサイロにデータを保存すると、サービス負荷の管理と調整はテナント中心モデルでのみ行うことができます。これは、主に 1 回限りの設定やポリシーのセットを導入することにつながり、これらを個別に管理および調整する必要があります。これは非効率的であるとともにオーバーヘッドの増加をもたらし、お客様のニーズに迅速に対応する能力を低下させる可能性があります。

サイロはコストの最適化を制限する ― 最も重大な欠点として、サイロモデルの 1 回限りの性質は、ストレージリソースの消費を調整する能力を制限する傾向があることです。

プールモデルのトレードオフ

プールモデルは、SaaS ライフスタイルに対する最終的なオールインコミットメントを表します。プールモデルでは、テナントストレージのプロビジョニング、移行、運用管理、モニタリングを合理化できる統一アプローチをテナントにもたらすことに重点があります。

長所

アジリティ ― すべてのテナントデータを 1 つのストレージ構造に一元化すると、合理的かつ普遍的なアプローチをサポートするツールやライフサイクルをより簡単に作成し、すべてのテナントにストレージソリューションを迅速にデプロイできます。このアジリティは、オンボーディングプロセスにも及びます。プールモデルでは、SaaS サービスにサインアップするテナントごとに個別のストレージインフラストラクチャをプロビジョニングする必要がありません。新しいテナントをプロビジョニングして、そのテナントの ID をインデックスとして使用するだけで、すべてのテナントが共有するストレージモデルからテナントのデータにアクセスできます。

ストレージのモニタリングと管理が容易になる ― プールモデルでは、ツールと集計分析を活用してテナントのストレージアクティビティを集約する方が自然です。1 つのストレ

ージモデルを管理するための日常的なツールを活用して、システムのヘルスに関する包括的なクロステナントビューを構築できます。プールモデルを使用すると、システムイベントに未然に対応するためのグローバルポリシーをより簡単に導入できます。一般的に、データを単一のデータベースと共有の表現にまとめると、マルチテナントのストレージ、デプロイ、管理の多くが容易になります。

追加のオプションにより、SaaS ソリューションのコストフットプリントを最適化できる ― コストの削減は、多くの場合、パフォーマンスのチューニングで達成できます。例えば、スループットの最適化は 1 つのポリシーとしてすべてのテナントに適用できます。テナントごとに別々のポリシーを管理する必要はありません。

プールはデプロイのオートメーションと運用のアジリティを向上させる ― プールモデルの共有の特性により、通常、データベースデプロイのオートメーション全体の複雑さが軽減されます。これは、製品の新機能を継続的および頻繁にリリースする SaaS の需要によく適合します。

短所

アジリティはスケーリングと可用性の管理を難しくする ― プールされたマルチテナント環境でストレージの機能停止が発生した場合の影響を考えてください。1 人のお客様がダウンするのではなく、**すべて**のお客様がダウンします。そのため、プールモデルを採用する組織は、環境のオートメーションとテストにも多額の投資を行う傾向があります。プールされたソリューションには、事前モニタリングソリューションと、堅牢なバージョニング、データ、スキーマの移行が必要です。リリースをスムーズに進める必要があり、テナントの問題を効率的にキャプチャして顕在化させる必要があります。

プールはテナントデータの配布を管理しにくくする ― プールされたストレージでは、テナントデータのサイズと配布が課題になることがあります。テナントは、多様な負荷をシステムにもたらし、これらの負荷がストレージパフォーマンスを低下させる可能性があります。プールモデルでは、テナントの多様な負荷に対応するためのメカニズムをよく検討する必要があります。データのサイズと配布は、データ移行へのアプローチ方法にも影響します。これらの問題は、通常、ストレージテクノロジーごとに異なるため、ケースバイケースで対処する必要があります。

プールされた環境の共有特性により、一部のドメインで抵抗に遭う場合がある ― SaaS 製

品によっては、規制および内部のデータの保護要件を満たすためにサイロモデルを要求するお客様がいます。

ハイブリッド: ビジネス上の妥協策

多くの組織にとって、戦略の選択は、サイロ、ブリッジ、プールのいずれかのモデルを選択するだけでは済まない場合があります。テナントとビジネスは、どのストレージ戦略を選択するかに大きく影響します。

一部のテナントがサイロモデルまたはブリッジモデルを必要としているとします。これらのテナントは、いずれかのモデルを選択すると、このモデルを使用してすべてのストレージを実装する必要があると思い込みます。このため、プールモデルを利用できるテナントを受け入れることが制限されます。実際、サイロモデルやブリッジモデルの属性を必要としていないテナントの階層にとっては、コストや複雑さが増す可能性があります。

ここで 1 つの妥協策として、プールされたストレージを完全にサポートするソリューションを基盤として構築できます。次に、サイロ化されたストレージソリューションを必要としているテナント用に別のデータベースを構築できます。図 3 は、このアプローチの実際の例を示しています。

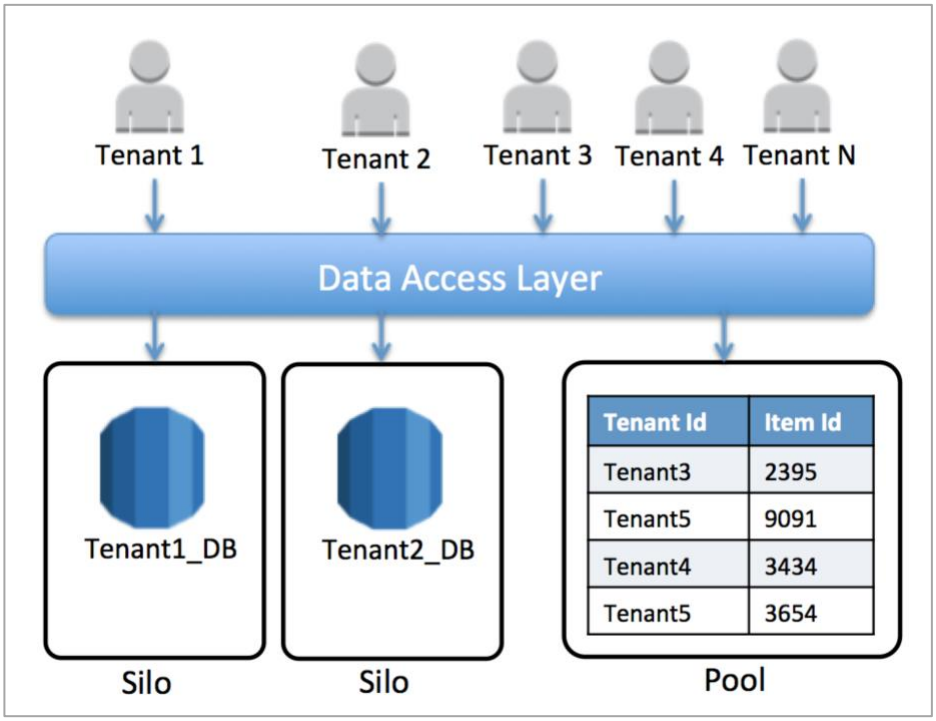

図 3 - ハイブリッドのサイロ/プールストレージ

ここでは、2 つのテナント (Tenant1 と Tenant2) がサイロモデルを利用し、残りのテナントはプールされたストレージモデルで実行しています。これは、テナントの基盤となるストレージからデベロッパーを隠すデータアクセスレイヤーによって抽象化されています。これにより、データアクセスレイヤーと管理プロファイルの複雑さが増しますが、サービスを階層化して両方のモデルを最大限に活用する方法をビジネスに提供できます。

データ移行

データ移行は、競合する SaaS ストレージモデルの評価から見逃される傾向があります。ただし、SaaS では、アーキテクチャの選択が、新しい特徴や機能を継続的にデプロイする能力にどのような影響するかを検討する必要があります。パフォーマンスおよび一般的なテナントエクスペリエンスに注目することは重要ですが、基盤となるデータ表現の継続

的な変更をストレージソリューションに反映する方法を検討することも必要です。

移行とマルチテナンシー

マルチテナントストレージのモデルごとに、データ移行に取り組む独自のアプローチが必要です。サイロモデルとブリッジモデルでは、テナントごとにデータを移行できます。このアプローチは、すべてのテナントを移行エラーにさらすことなく、各 SaaS テナントを慎重に移行できるという点で、組織にとって魅力的かもしれません。ただし、このアプローチでは、デプロイライフサイクルのオーケストレーション全体がより複雑になる可能性があります。

プールモデルでのデータ移行は、魅力的ですが困難を伴います。プールモデルでの移行では、単一のポイントを通じて、すべてのテナントを新しいデータモデルに正常に移行できます。一方、プールの移行中に問題が発生した場合は、すべてのテナントに影響する可能性があります。

最初に、データ移行がマルチテナント SaaS 戦略全体にどのように適合するかを検討する必要があります。この移行オーケストレーションを配信パイプラインに早期に組み込むと、リリースプロセスのアジリティを高めることができます。

侵襲的な変更の最小化

経験則として、システム内のデータをどのように進化させるかを検討する際には、明確なポリシーと原則に従う必要があります。可能な限り、チームは以前のバージョンとの下位互換性があるデータ変更を重視する必要があります。アプリケーションのデータ表現に対する変更を最小限に抑える方法を見つけることができれば、データを新しい表現に変換する際の高いオーバーヘッドを制限できます。

一般的に使用されているツールや手法を活用して、移行プロセスをオーケストレートできます。実際には、侵襲的な変更を最小限に抑えることは、SaaS デベロッパーにとって重要ではありますが、SaaS ドメインに固有のものではありません。したがって、このホワイトペーパーでは取り上げていません。

セキュリティに関する考慮事項

SaaS プロバイダーは、データのセキュリティを最優先事項にする必要があります。マルチテナント戦略を採用する場合、組織は、テナントデータを不正アクセスから効果的に保護するために堅牢なセキュリティ戦略を必要とします。SaaS のお客様の信頼を得るには、テナントデータを保護し、システムで適切なセキュリティ対策を採用していることを伝える必要があります。

選択したストレージ戦略は、AWS でサポートしている一般的なセキュリティパターンを使用している可能性があります。例えば、保管中のデータの暗号化は、すべてのモデルに例外なく適用できる水平戦略です。これにより、データに不正にアクセスしても、情報の復号に必要なキーなしでは悪用できないため、基本的なセキュリティレベルが確保されます。

ここで、サイロ、ブリッジ、プールの各モデルのセキュリティプロファイルを確認すると、各モデルでのセキュリティの実現方法に追加のバリエーションがあることに気付きます。例えば、AWS Identity and Access Management (AWS IAM) には、テナントデータへのアクセスをスコープおよび制御する方法に微妙な差異があることがわかります。一般に、サイロモデルとブリッジモデルは、データベース全体またはテーブル全体へのアクセスを制限するために適用できるため、IAM ポリシーにより自然に適合します。プールモデルに切り替えると、IAM を活用してデータへのアクセスをスコープできなくなる場合があります。代わりに、アプリケーションのサービスの認可モデルがより多くの責任を肩代わりすることになります。これらのサービスでは、共有表現のデータに対するスコープと制御を解決するために、ユーザーのアイデンティティを使用する必要があります。

分離とセキュリティ

一部の組織やドメインでは、テナントの分離をサポートすることが基本です。SaaS プロバイダーに規制上またはセキュリティ上の特定の要件がある場合、仮想環境であってもデータを分離することが不可欠になると考えられます。

各 AWS ストレージソリューションを検討する場合は、AWS ストレージサービスごとに分離を実現する方法に注目してください。分離を実現する方法は、RDS と DynamoDB では

大きく異なります。ストレージ戦略を選択し、お客様のセキュリティ上の考慮事項を評価する際は、これらの差異を考慮してください。

管理とモニタリング

どのアプローチをマルチテナントストレージに採用するかは、SaaS ソリューションの管理およびモニタリングのプロファイルに大きな影響を与える可能性があります。実際、システムのヘルスの集約および分析に伴う複雑さとアプローチは、ストレージモデルや AWS テクノロジーごとに大きく異なる場合があります。

ストレージ傾向の集約

SaaS ストレージの効果的な運用ビューを構築するには、メトリクスとダッシュボードを通じてテナントアクティビティの集約ビューを取得する必要があります。すべてのテナントのエクスペリエンスに影響する可能性があるストレージ傾向を事前に特定できることが必要です。この集約ビューを作成するために必要なメカニズムは、サイロモデルとプールモデルでは大きく異なります。サイロ化されたストレージでは、分離された各データベースからデータを収集し、その情報を集約モデルに表示するためのツールを使用する必要があります。対照的に、プールモデルの場合は、その性質上、テナントアクティビティの集約ビューが既に存在します。

テナント中心のアクティビティビュー

管理およびモニタリングのストレージソリューションでは、ストレージのアクティビティをテナント中心に表示する方法を提供する必要があります。特定のテナントでストレージの問題が発生している場合は、ストレージのメトリクスとプロファイルデータをドリルダウンして、その特定テナントで発生している問題の原因を確認できるようにしておく必要があります。この場合、サイロモデルは、ストレージアクティビティのテナント中心のビューの構築とより自然に適合します。プールされたストレージ戦略では、特定のテナントのストレージアクティビティを抽出するために、テナントのフィルタリングメカニズムが

必要になります。

ポリシーとアラーム

各 AWS ストレージサービスには、アプリケーションのストレージパフォーマンスを評価および調整するための独自のメカニズムがあります。ストレージはシステムの重大なボトルネックになる場合があるため、モニタリングのポリシーとアラームを導入して、アプリケーションのストレージに関するヘルスの変化を特定して対応する必要があります。

選択したパーティション分割モデルは、ストレージのモニタリング戦略の複雑さと管理の容易性にも影響します。ソリューションをサイロ化するほど、テナントごとの管理と維持に伴う変動要素が増えます。これとは対照的に、プールされたストレージ戦略は、その共有の特性により、すべてのテナントにわたるポリシーとアラームの一元的な収集が容易になります。

これらのストレージポリシー全体の目的は、ヘルスイベントの予測と対応に役立つ一連のプロアクティブなルールを適用することです。マルチテナントストレージモデルを選択する場合は、各アプローチがシステムのストレージポリシーやアラームの実装方法にどのように影響するかを検討してください。

階層化ストレージモデル

AWS は幅広いストレージサービスをデベロッパーに提供しています。各サービスを組み合わせて適用することで、SaaS テナントのさまざまなコストとパフォーマンス要件に対応できます。ここで重要な点は、ストレージ戦略を AWS の特定のサービスやストレージテクノロジーに限定しないことです。

アプリケーションのストレージニーズをプロファイリングする場合は、きめ細かいアプローチを通じて、特定のストレージサービスの強みとアプリケーションのコンポーネントごとの要件を適合させます。例えば、DynamoDB は特定のアプリケーションサービスに適し、RDS は別のアプリケーションサービスに適している場合があります。ソリューションにマイクロサービスアーキテクチャを使用する場合、サービスごとにストレージビューが異なるときは、各サービスのプロファイルにどのストレージテクノロジーが最適であるか

を検討してください。アプリケーションを構成するマイクロサービスごとに異なるストレージソリューションを使用することは珍しくありません。

この戦略では、SaaS ソリューションを階層化する別の方法として、ストレージを使用することもできます。階層ごとに異なるストレージ戦略を使用し、ソリューションの各階層の価値提案を区別するさまざまなレベルのパフォーマンスと SLA を提供できます。このアプローチを使用することで、テナントの階層と、これらの階層がインフラストラクチャにもたらすコストおよび負荷とを適切に調整できます。

デベロッパーエクスペリエンス

一般的なアーキテクチャの原則として、通常、デベロッパーはアプリケーションの水平的な側面を一元化して抽象化するレイヤーやフレームワークを導入しようとします。この場合の目標は、ポリシーとテナント解決戦略を一元化して標準化することです。例えば、データアクセスレイヤーを導入して、データアクセスリクエストにテナントコンテキストを挿入できます。これにより、開発が簡素化され、テナントのアイデンティティがシステムをどのように通過するかについて、デベロッパーが意識する範囲が限定されます。

このレイヤーを配置すると、テナントごとに異なるポリシーや戦略に関するオプションも増えます。また、これに伴ってストレージアクティビティの設定と追跡を一元化する機会も生まれます。

連結アカウントサイロモデル

各ストレージサービスの詳細を確認する前に、AWS 連結アカウントを使用して AWS ストレージソリューション上にサイロモデルを実装する方法を考えてみます。このアプローチでサイロを実現するには、ソリューションでテナントごとに異なる連結アカウントをプロビジョニングする必要があります。これにより、テナントのインフラストラクチャ全体が他のテナントから完全に分離されるため、サイロを確実に達成できます。

連結アカウントのアプローチは、一括請求 (コンソリデーティッドビリング) 機能に依存しており、お客様は子アカウントを支払者アカウント全体に関連付けることができます。こ

の場合、テナントごとに連結アカウントが異なるとしても、これらのテナントの請求は依然として集約され、1 つの請求書の一部として支払者アカウントに表示されます。
図 4 は、連結アカウントを使用してサイロモデルを実装する方法の概念的なビューを示しています。ここでは、2 つのテナントに別々のアカウントがあり、各テナントは支払者アカウントに関連付けられています。このような分離により、利用可能な AWS ストレージテクノロジーのうち、どれでも自由に選択してテナントのデータを保管できます。

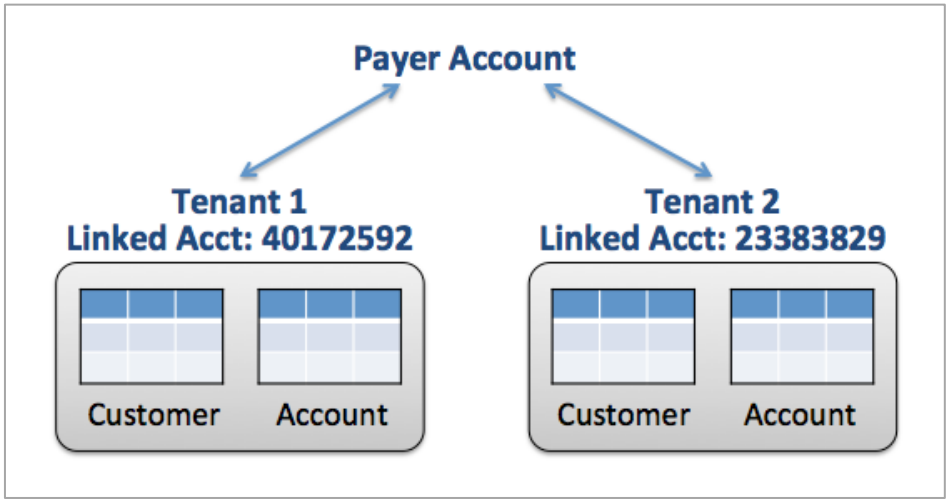

図 4 - 連結アカウントを使用したサイロモデル

これを見ると、サイロ環境を必要としている SaaS プロバイダーには非常に魅力的な戦略であるように思えます。個々のテナントの管理と移行は、部分的に確実に簡素化されます。また、AWS の経費を連結アカウントレベルで要約できるため、テナントのコストも確認しやすくなります。
ただし、これらの利点があっても、連結アカウントのサイロモデルには重要な制限があります。例えば、プロビジョニングは確実に複雑になります。テナントのインフラストラクチャの作成に加えて、各連結アカウントの作成を自動化し、それに伴う制限を調整する必要があります。さらに大きな課題はスケーリングです。AWS では、作成できる連結アカウントの数に制約があります。これらの制約は、多数の新しい SaaS テナントを作成する環境に適合しない可能性があります。

DynamoDB でのマルチテナンシー

DynamoDB でデータをスコープおよび管理する方法には、マルチテナンシーへのアプローチを複雑にするいくつかの新しい要素があります。一部のストレージサービスは従来のデータパーティション分割戦略と適合しますが、DynamoDB ではサイロ、ブリッジ、プールの各モデルへの直接マッピングがわずかに少なくなります。DynamoDB では、マルチテナント戦略を選択する際に、いくつかの追加の要素を考慮する必要があります。
以下のセクションでは、DynamoDB でマルチテナントパーティション分割の各スキームを実現するために一般的に使用される AWS のメカニズムについて説明します。

サイロモデル

DynamoDB でのサイロモデルの実装方法を確認する前に、サービスがデータへのアクセスをどのようにスコープおよび制御するかを検討する必要があります。RDS とは異なり、DynamoDB にはデータベースインスタンスの概念はありません。代わりに、DynamoDB で作成したすべてのテーブルは、リージョン内のアカウントに対してグローバルになります。つまり、そのリージョンの各テーブルの名前は、アカウントごとに一意である必要があります。

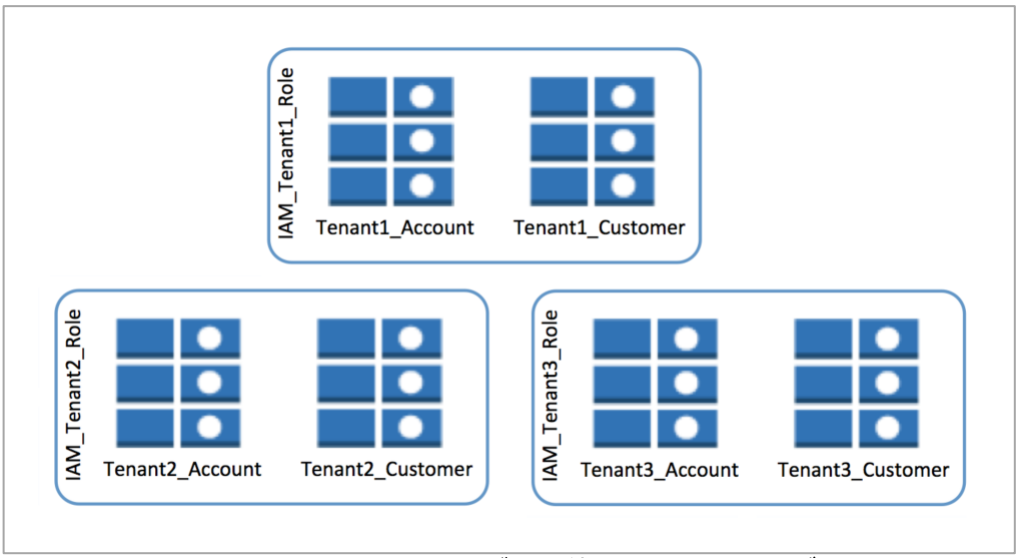

図 5 - DynamoDB テーブルを使用したサイロモデル

DynamoDB でサイロモデルを実装する場合、特定のテナントに関連する 1 つ以上のテーブルをグループ化する方法を見つける必要があります。また、このアプローチでは、これらのテーブルの安全な制御されたビューを作成してサイロのお客様のセキュリティ要件を満たすとともに、複数のテナントをまたいだデータアクセスを回避する必要があります。
図 5 は、このテナント別にテーブルをグループ化する方法の一例を示しています。テナントごとに 2 つのテーブル (`Account` と `Customer`) が作成されていることに注意してください。これらのテーブルには、テーブル名の先頭にテナント ID も付けられています。これにより、DynamoDB のテーブル命名要件が解決され、テーブルおよび関連するテナントの間に必要なバインドが作成されます。
これらのテーブルへのアクセスは、IAM ポリシーを導入して達成することもできます。プロビジョニングプロセスでは、テナントごとにポリシーの作成を自動化し、このポリシーを特定のテナントが所有するテーブルに適用する必要があります。
このアプローチは、サイロモデルの基本的な分離目標を達成し、各テナントのデータ間に明確な境界を定義します。また、テナントごとに調整および最適化することもできます。次の 2 つの特定の領域を調整できます。

- Amazon CloudWatch のメトリクスは、テーブルレベルでキャプチャできるため、ストレージアクティビティのテナントメトリクスを簡単に集約できます。
- テーブルの書き込み/読み取り容量は、1 秒あたりの入/出力 (IOPS) として測定され、テーブルレベルで適用されるため、テナントごとに異なるスケーリングポリシーを作成できます。

このモデルの欠点は、運用面と管理面で顕著になる傾向があります。このアプローチの場合、テナントの運用ビューでは、テナントのテーブル命名スキームを認識し、テナント中心のコンテキストで情報をフィルタリングして表示する必要があります。このアプローチでは、これらのテーブルを操作する必要があるコードに対して、間接的なレイヤーを追加します。DynamoDB テーブルとやり取りするたびに、テナントコンテキストを挿入して、各リクエストを適切なテナントテーブルにマッピングする必要があります。
マイクロサービスベースのアーキテクチャを採用する SaaS プロバイダーには、別の一連の考慮事項もあります。マイクロサービスの場合、通常、チームはストレージの責任を個々のサービスに分散します。各サービスでは、データを保存および管理する方法を自由

に決定できます。これに伴って、DynamoDB での分離が複雑になり、各サービスのニーズに合わせてテーブルの数を増やす必要があります。また、スコーピングに別の面が追加され、各サービスのテーブルごとにサービスへのバインドを識別する必要があります。
これらの課題を軽減し、DynamoDB のベストプラクティスを合わせるには、1 つのテーブルで**すべて**のテナントデータに対応することを検討してください。このアプローチは、いくつかの点で効率化をもたらし、ソリューションのプロビジョニング、運用管理、移行のプロファイルを簡素化します。
ほとんどの場合、個別の DynamoDB テーブルと IAM ポリシーを使用してテナントデータを分離することで、サイロモデルのニーズに対処できます。別の唯一のオプションは、前述の連結アカウントのサイロモデルを検討することです。ただし、前に説明したように、連結アカウントの分離モデルには追加の制限と考慮事項があります。

ブリッジモデル

DynamoDB では、ブリッジモデルとサイロモデルの境界が非常にあいまいです。ブリッジモデルを使用する目的が、基本的に、1 つのアカウントでクライアントごとに 1 回限りのスキーマバリエーションを持つことである場合、この目的は前述のサイロモデルで達成できることがわかります。
ブリッジの場合、唯一の問題は、サイロモデルで説明した分離の要件の一部を緩和できるかどうかです。テーブルレベルの IAM ポリシーを導入しないことで、これを実現することができます。テナントに完全な分離が必要なければ、IAM ポリシーを削除してプロビジョニングスキームを簡素化できることになります。ただし、ブリッジの場合でも分離のメリットはあります。したがって、IAM の分離をなくすことは魅力的ですが、クロステナントアクセスを制限できる構造とポリシーを活用することは、依然として SaaS の優れたプラクティスです。

プールモデル

DynamoDB でプールモデルを実装するには、サービスでどのようにデータを管理するかを冷静に検討する必要があります。データは DynamoDB に保存されるため、サービスでは、データを継続的に評価およびパーティション分割することで、スケーリングを達成す

る必要があります。また、データのプロファイルが均等に分散されている場合は、この基盤となるパーティション分割スキームを活用するだけで、SaaS テナントのパフォーマンスとコストのプロファイルを最適化できます。

ここでの課題は、マルチテナントの SaaS 環境内のデータは、通常、均一に分散されていないことです。SaaS テナントは形とサイズがさまざまであるため、データは決して均一にはなりません。SaaS ベンダーでは、少数のテナントがデータフットプリントの大部分を消費するようになるのが一般的です。

このため、DynamoDB 上でのプールモデルの実装に伴って問題が生じることがわかります。テナント ID を 1 つの DynamoDB パーティションキーに単にマッピングすると、パーティションの「ホットスポット」が生じることもすぐにわかります。1 つの大規模なテナントのせいで DynamoDB がデータを効果的にパーティション分割できなくなることが考えられます。これらのホットスポットは、ソリューションのコストとパフォーマンスに影響を与える可能性があります。キーの分散が最適ではない場合、IOPS を増やしてホットパーティションの影響を相殺する必要があります。IOPS の増加は、ただちにソリューションのコスト増となります。

この問題を解決するには、テナントデータの分散をより適切に制御するためのメカニズムを導入する必要があります。シングルテナント ID に依存したデータのパーティション分割をしないアプローチが必要になります。これらすべての要因は 1 つのパスにつながります。セカンダリシャーディングモデルを作成して、各テナントを複数のパーティションキーに関連付ける必要があります。

このようなソリューションを実現する方法の例を次に示します。まず、別のテーブル (これを「テナントルックアップテーブル」と呼びます) が必要です。このテーブルによって、対応する DynamoDB パーティションキーへのテナントのマッピングをキャプチャして管理します。図 6 は、テナントルックアップテーブルを構成する方法の例を示しています。

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”]
		}

図 6 - テナントルックアップテーブルの導入

このテーブルには、2 つのテナントのマッピングが含まれています。これらのテナントに関連する項目には属性があり、テナントに関連するテーブルごとのシャーディング情報が含まれています。ここでは、どちらのテナントにも [Customer] テーブルと [Account] テーブルのシャーディング情報があります。また、テナントとテーブルの組み合わせごとに、テーブルの現在のシャーディングプロファイルを表す 3 つの情報があることに注意してください。これらの情報は以下のとおりです。

- **ShardCount** - テーブルに現在関連付けられているシャードの数を示します。
- **ShardSize** - 各シャードの現在のサイズ
- **ShardId** - テナントにマッピングされたパーティションキーのリスト (テーブル別)

このメカニズムを使用すると、テーブルごとにデータの分散方法を制御できます。ルックアップテーブルの間接参照を使用すると、格納するデータの量に基づいてテナントのシャーディングスキームを動的に調整できます。データフットプリントが特に大きいテナントには、シャード数を増やします。モデルではテーブルごとにシャーディングを設定するため、特定のシャーディング設定に対するテナントのデータニーズのマッピングをより細かく制御できます。これにより、テナントのデータプロファイルに表示されることが多い自

然なバリエーションに合致したパーティション分割を行うことができます。
テナントルックアップテーブルを導入すると、テナントデータの分散に対処できますが、同時にコストが伴います。このモデルでは、ソリューションのデータアクセスレイヤーで対処する必要がある間接参照のレベルが導入されます。テナント ID を使用してデータに直接アクセスする代わりに、まず該当するテナントのシャードマッピングを参照し、これらの ID の和集合を使用してテナントデータにアクセスします。図 7 のサンプルの [Customer] テーブルは、このモデルでのデータの表現方法を示しています。

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

図 7 - [Customer] テーブルのシャード ID

この例で、ShardID は図 6 に示したテーブルからの直接マッピングです。そのテナントルックアップテーブルでは、Tenant1 と Tenant2 の [Customer] テーブルが分かれており、それぞれに個別のシャード ID のリストが含まれています。これらのシャード ID は、このサンプルの [Customer] テーブルに表示されている値に直接相関します。実際のテナント ID は、この [Customer] テーブルに決して表示されないことに注意してください。

シャード分散の管理

このモデルの仕組みは特に複雑というわけではありません。データを効果的に分散する戦略をどのように実装するかを考えるときに、この問題に注目する必要があります。テナントに追加のシャードが必要になったときに、どのように検出するか。 このプロセスを自動化するために、どのようなメトリクスと基準を収集できるか。 データとドメインの特性は、データプロファイルにどのように影響するか。 どのソリューションでも、これらの質問のすべてを解決する単一のアプローチはありません。一部の SaaS 組織は、お客様の正しい情報に基づいて、これを手動で調整しています。他の組織は、より自然な基準に従ったアプローチを採用しています。

ここで示すアプローチは、データの分散を処理するために選択できる 1 つの方法です。最終的に、ここで説明している原則のうち、環境のニーズに最も合致する原則のハイブリッドを採用することになります。重要な点は、プールモデルを採用する場合、DynamoDB がデータをどのようにパーティション分割するかを認識することです。データの分散方法を考慮せずにデータをむやみに移動すると、SaaS ソリューションのパフォーマンスとコストのプロファイルを損なう可能性があります。

IOPS の動的な最適化

SaaS 環境の IOPS ニーズの管理は、困難を伴う場合があります。テナントがシステムにもたらす負荷は、大きく異なる場合があります。IOPS を最悪のケースに備えて設定すると、最大レベルに達するまでは実際の負荷に基づくコストの最適化が見送られます。

代わりに、動的なモデルを実装し、テーブルの IOPS をアプリケーションの負荷プロファイルに基づいてリアルタイムで調整することを検討してください。Dynamic DynamoDB は、この問題に対処するために使用できる、設定可能なオープンソースソリューションの 1 つです。

複数の環境のサポート

DynamoDB に関する戦略について考える場合、これらの各モデルを複数の環境 (QA、開発、本番など) にわたってどのように実現するかを検討してください。複数の環境を使用

する必要がある場合は、AWS でストレージ戦略を環境別に分離して使い分ける方法に影響が生じます。例えば、ブリッジモデルとプールモデルでは、テーブル名に修飾子を追加して環境コンテキストを指定することになります。これには多少の指定ミスが伴いますが、これを織り込んでテーブル名のプロビジョニングとランタイム解決を行う必要があります。

移行の効率性

DynamoDB のスキーマレスの性質は、SaaS プロバイダーにとって重要な利点となります。これにより、新しいテーブルやレプリケーションを導入することなく、アプリケーションに更新を適用し、テナントデータを移行できます。DynamoDB は、SaaS のバージョン間でテナントを移行するプロセスを簡素化します。また、SaaS ソリューションの最新バージョンで複数のアジャイルテナントを同時にホストしながら、他のテナントが以前のバージョンを引き続き使用できるようにします。

トレードオフの検討

各モデルには、ビジネスニーズに最適なモデルを決定する際に考慮すべきトレードオフがあります。サイロパターンは魅力的に見えますが、プロビジョニングと管理の複雑さが増すため、ソリューションのアジリティが低下します。個別の環境をサポートし、環境ごとに個別のテーブルのグループを作成すると、間違いなく、自動デプロイの複雑さに影響します。ブリッジは、DynamoDB ではサイロモデルのわずかなバリエーションです。したがって、サイロモデルと共通する部分がほとんどです。

DynamoDB のプールモデルには、いくつかの大きな利点があります。データのフットプリントの統合により、プロビジョニング、移行、管理、モニタリングのエクスペリエンスが簡素化されます。また、マルチテナントアプローチを採用して、クロステナントベースで読み取り/書き込みの IOPS を調整することで、消費とテナントのエクスペリエンスを最適化できます。これにより、パフォーマンスの問題により広範に対応でき、コストを最小限に抑える機会が生まれます。これらの要因により、プールモデルは SaaS 組織にとって非常に魅力的なものになります。

RDS でのマルチテナンシー

初期の SaaS システムはリレーショナルデータベースで提供されることが多かったため、デベロッパーコミュニティは、これらの環境でマルチテナンシーに対処するためのいくつかの一般的なパターンを確立しています。実際、RDS にはサイロ、ブリッジ、プールの各モデルへのより自然なマッピングがあります。

RDS でのデータの構造と表現は、非マネージドリレーショナル環境の拡張と言えます。例えば、MySQL で使用できる基本的なメカニズムは RDS でも使用できます。これにより、すべての RDS の状況でマルチテナンシーを実現することが比較的簡単になります。

以下のセクションでは、RDS でパーティション分割モデルを実現するために一般的に使用するさまざまな戦略について説明します。

サイロモデル

AWS では、複数の方法でサイロパターンを実現できます。ただし、分離を実現する最も一般的で簡単なアプローチは、テナントごとに個別のデータベースインスタンスを作成することです。個別のインスタンスにより、お客様のコンプライアンスニーズを一般的に満たす分離レベルを実現できます。この場合、完全に分離したアカウントのプロビジョニングにはオーバーヘッドが生じません。

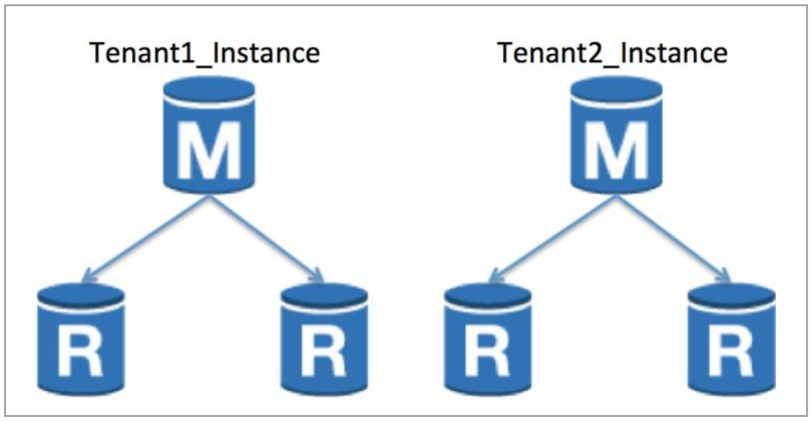

図 8 - サイロとしての RDS インスタンス

図 8 は、RDS 上で実現できる基本的なサイロモデルを示しています。ここでは、2 つのテナントに 2 つの個別のインスタンスをプロビジョニングしています。
この図は、テナントインスタンスごとに 1 つのマスターデータベースと 2 つのリードレプリカを示しています。これは、このアプローチを使用してテナントごとに最適化した高可用性の戦略をセットアップおよび設定する方法を示すためのオプションの概念です。

ブリッジモデル

RDS でブリッジモデルを実現することは、すべてのストレージモデルに共通するテーマと適合します。基本的なアプローチは、すべてのテナントで 1 つのインスタンスを利用するとともに、そのデータベース内のテナントごとに個別の表現を作成することです。これにより、各テーブルを特定のテナントにマッピングするためのプロビジョニングとランタイムテーブルの解決が必要になります。
ブリッジモデルは、テナントデータの移行時にテナントごとにスキーマを変更する機会とともに、ある程度の柔軟性も提供します。例えば、複数の異なるテナントで複数の異なるバージョンの製品を同時に実行し、スキーマの変更をテナントごとに徐々に移行させることができます。
図 9 は、RDS でブリッジモデルを実装する 1 つの方法を例として示しています。この図では、1 つの RDS データベースインスタンス内に、`Tenant1` の [`Customer`] テーブルと `Tenant2` の [`Customer`] テーブルが分かれています。

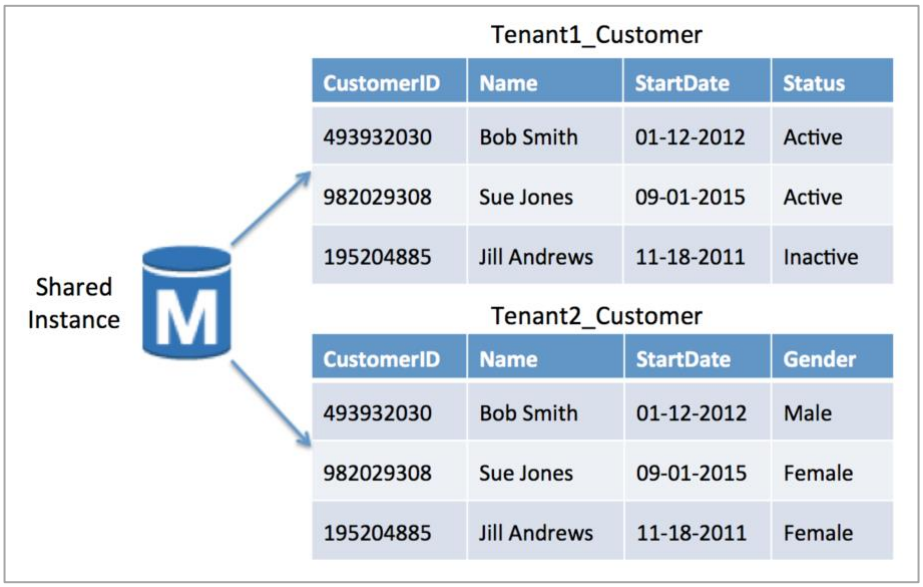

図 9 - RDS でのブリッジモデルの例

この例は、テナントレベルでスキーマを変更できることを示しています。`Tenant1` のスキーマには [`Status`] 列がありますが、この列は `Tenant2` では削除され、[`Gender`] 列に置き換えられています。

別のオプションとしては、インスタンス内のテナントごとに個別のデータベースの概念を導入します。RDS の状況ごとに用語は異なります。RDS ストレージコンテナによっては、これをデータベースと呼ぶ場合もあれば、スキーマと呼ぶ場合もあります。

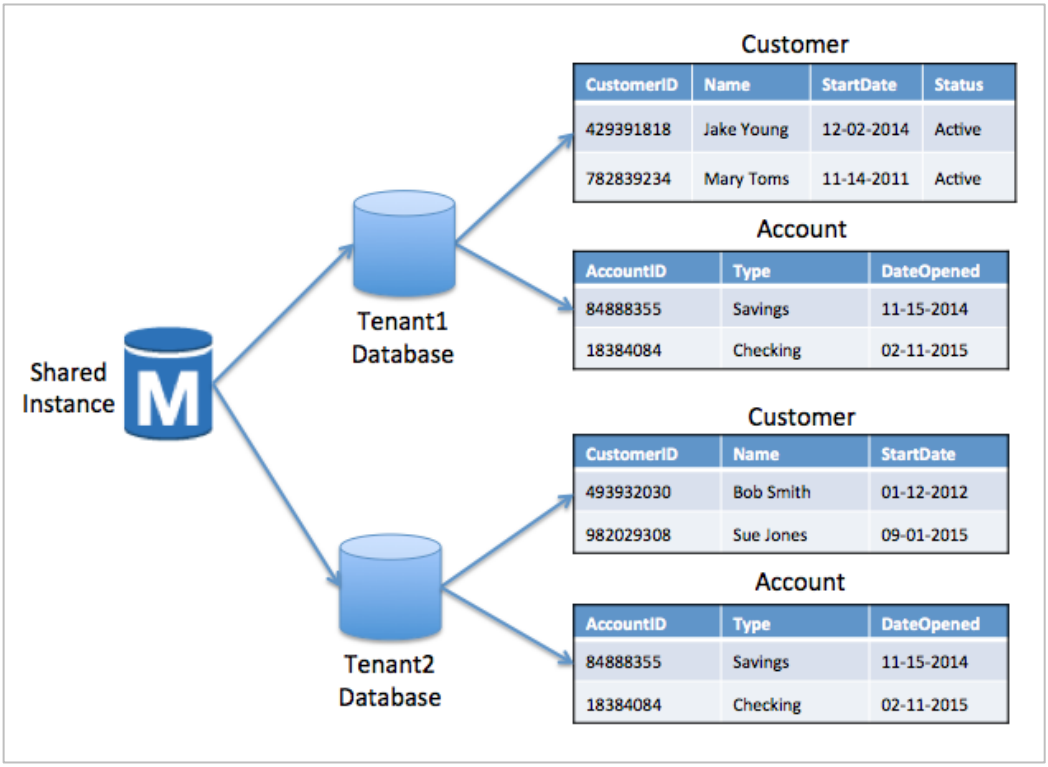

図 10 - 個別のテーブル/スキーマを持つ RDS のブリッジ

図 10 は、この代替ブリッジモデルの例を示しています。テナントごとにデータベースを作成し、テナントごとに独自のテーブルのコレクションを持たせていることに注目してください。一部の SaaS 組織では、これにより、テナントデータの管理がより自然にスコープされ、名前付けを個別のテーブルに伝播する必要がなくなります。

このモデルは魅力的ですが、RDS のすべての状況に最適であるとは限りません。一部の RDS コンテナでは、インスタンスに作成できるデータベース/スキーマの数が制限されています。例えば、SQL Server コンテナでは、インスタンスあたり 30 個のデータベースのみが許可されます。これは、ほとんどの SaaS 環境では受け入れられない可能性があります。

ブリッジモデルではテナント間のバリエーションが可能ですが、重要な点として、スキーマの変更をポリシーで制限することは依然として必要です。スキーマの変更を導入するた

びに、ダウンタイムを生じることなく、SaaS テナントを新しいモデルに正常に移行することが課題となります。したがって、このモデルは、これらの移行を簡素化しますが、1 回限りのテナントのスキーマや、テナントのデータの表現に対する定期的な変更は促進しません。

プールモデル

RDS のプールモデルは、従来のリレーショナルインデックス作成スキームに依存してテナントデータをパーティション分割します。すべてのテナントデータを共有インフラストラクチャモデルに移動する一環として、テナントデータを単一の RDS インスタンスに保存し、テナント間で共通のテーブルを共有します。これらのテーブルには、一意のテナント ID をインデックスとして付け、この ID を使用して各テナントのデータにアクセスして管理します。

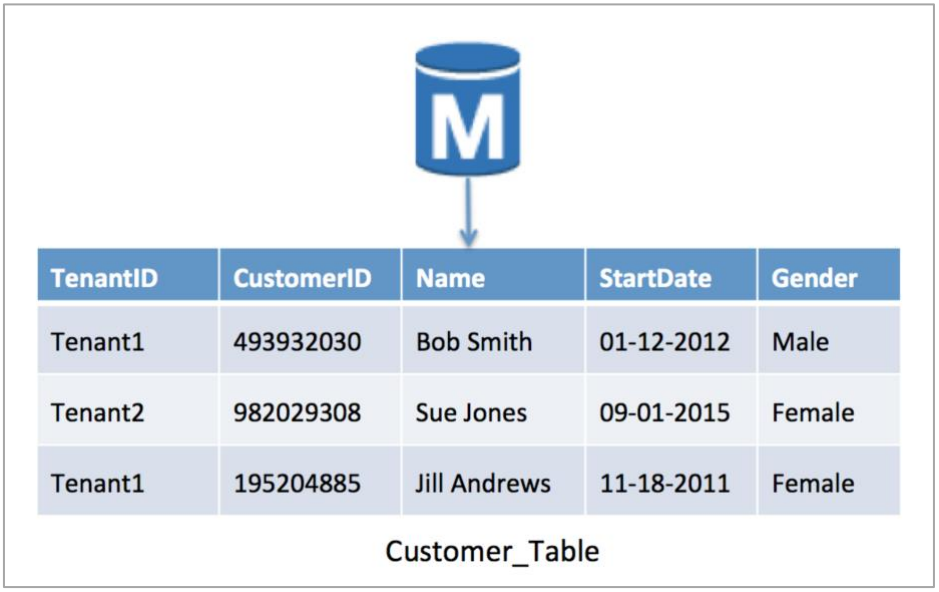

図 11 - 共有スキーマを持つ RDS プールモデル

図 11 に、実際のプールモデルの例を示します。ここでは、RDS インスタンスに **1 つ**の [`Customer`] テーブルがあり、このテーブル内にアプリケーションのすべてのテナントのデータを保持しています。RDS は RDBMS であるため、すべてのテナントが同じスキー

マバージョンを使用する必要があります。RDS は DynamoDB とは異なります。DynamoDB では、柔軟なスキーマにより、各テナントが単一のテーブル内に一意のスキーマを持つことができます。

単一のインスタンス制限を考慮する

上で説明したモデルの多くでは、単一のインスタンスにデータを保存し、そのインスタンス内でデータをパーティション分割することに重点を置いています。SaaS 環境のサイズとパフォーマンスのニーズによっては、単一のインスタンスを使用することが、テナントデータのプロファイルに適合しない場合があります。RDS では、単一のインスタンスに保存できるデータの量に制限があります。この制限の内訳は以下のとおりです。

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

さらに、単一のインスタンスの使用に伴ってリソースの競合問題 (CPU、メモリ、I/O) が発生します。
単一のインスタンスが実用的でないシナリオでは、自然な拡張として、テナントデータを複数のインスタンスに分散するシャーディングスキームを導入します。このアプローチでは、シャード化したインスタンスの小さなコレクションから始めます。次に、テナントデータのプロファイルを継続的に観察し、単一のインスタンスが制限に達したり、ボトルネックにならないよう、インスタンスの数を増やしていきます。

トレードオフの検討

RDS を使用する場合のトレードオフはかなり単純です。通常は、管理およびプロビジョニングの複雑さとアジリティとのトレードオフになります。全体として、RDS のサイロモデルでは、プロビジョニングの自動化の問題は少なくなります。ただし、通常、プールモデルに関連するコストと管理効率は大きな問題です。これは、これらのモデルが継続的デリバリー環境にどのように適合するかを考えたときに、特に重要になります。

Amazon Redshift でのマルチテナンシー

Amazon Redshift では、マルチテナントの考え方がさらに複雑になります。Amazon Redshift は、大規模なデータウェアハウスを格納する高性能クラスターの構築に重点を置いています。また、Amazon Redshift では、各クラスター内に作成できる構造にいくつかの制限があります。以下の制限を考慮します。

- クラスターあたり 60 件のデータベース
- データベースあたり 256 件のスキーマ
- データベースあたり 500 件の同時接続数
- 50 件の同時クエリ数
- クラスターにアクセスすることで、クラスター内のすべてのデータベースにアクセスできます。

上記の制限が Amazon Redshift へのスケーリングとパフォーマンスにどのように影響するかは想像できます。また、これらの制限が Amazon Redshift でのマルチテナンシーへのアプローチに与える影響も想像できます。適度なテナント数が対象である場合、これらの制限がソリューションに与える影響はほとんどありません。ただし、多数のテナントが対象である場合は、これらの制限を全体的な戦略に織り込む必要があります。
以下のセクションでは、Amazon Redshift で各マルチテナントストレージモデルを実現するために一般的に使用する戦略に注目します。

サイロモデル

Amazon Redshift でテナントを真に分離するサイロモデルを実現するには、テナントごとに個別のクラスターをプロビジョニングする必要があります。クラスターを使用することで、テナント間に明確に定義された境界を作成できます。この境界は、お客様のデータがクロステナントアクセスから正常に隔離されていることを保証するために一般的に必要です。このアプローチでは、Amazon RedShift の自然なセキュリティメカニズムを最大限に活用します。したがって、IAM ポリシーとデータベース権限の組み合わせを使用して、

クラスターへのテナントアクセスを制御および制限できます。IAM ではクラスター全体の管理を制御し、データベース権限ではクラスター内のデータへのアクセスを制御します。サイロモデルを使用すると、テナントごとに調整したエクスペリエンスを作成できます。Amazon Redshift では、クラスター内のノードの数とタイプを設定できるため、テナントごとの負荷プロファイルを対象とした環境を作成できます。また、コストを最適化するための戦略として、これを使用することもできます。

他のサイロモデルで見てきたように、このモデルの課題は、各テナントのクラスターをオンボーディングプロセスの一環としてプロビジョニングする必要があることです。このプロセスを自動化してプロビジョニングプロセスに伴う余分な時間とオーバーヘッドを吸収することは、デプロイのフットプリントに複雑さを加えることになります。また、新しいテナントを割り当てる速度にも多少の影響があります。

ブリッジモデル

Amazon Redshift でのブリッジモデルには自然マッピングがありません。技術的には、テナントごとに個別のスキーマを作成できます。ただし、Amazon Redshift ではスキーマ数が 256 件に制限されているため、問題が発生する可能性があります。テナント数が多い環境では、単にスケールされません。ブリッジモデルでは、セキュリティも Amazon Redshift の課題となります。Amazon Redshift クラスターのユーザーとして承認されている場合、そのクラスター内のすべてのデータベースへのアクセスが許可されます。このため、きめ細かなアクセスコントロールを適用する責任は、SaaS アプリケーションが負うことになります。

ブリッジモデルを使用する動機とこれらの技術的な考慮事項を勘案すると、大半の SaaS プロバイダーにとって、このアプローチを Amazon Redshift で使用することを検討しても現実的ではないようです。ソリューションで制限に対処できたとしても、分離プロファイルはお客様にとって受け入れられない可能性があります。最終的に、分離を必要とするテナントごとにサイロモデルを使用することが最善の答えとなります。

プールモデル

Amazon Redshift でのプールモデルの構築は、これまでに説明した他のストレージモデル

と非常によく似ています。基本的な考え方は、すべてのテナントのデータを、共有データベースおよびテーブルを持つ 1 つの Amazon Redshift クラスターに保存することです。このアプローチでは、一意のテナント ID を示す列を導入して、テナントデータをパーティション分割します。

このアプローチでは、他のプールモデルで見た利点のほとんどを活用できます。すべてのテナントデータを単一の Amazon Redshift クラスターに格納することで、全体的な管理、モニタリング、アジリティが向上します。

同時接続数の制限は、Amazon Redshift でプールモデルを実装する際に、ある程度の困難をもたらします。同時接続数の上限は 500 であるため、多くのマルチテナント SaaS 環境は、この制限をすぐに超えてしまいます。ただし、これでプールモデルが無用になるわけではありません。代わりに、これらの接続を消費および解放する方法とタイミングを管理する効果的な戦略を策定する責任は、SaaS デベロッパーが負うことになります。

接続の管理に対処するための一般的な方法がいくつかあります。デベロッパーは、多くの場合、クライアントベースのキャッシュを活用して、Amazon Redshift への実際の接続の必要性を制限します。このモデルでは、接続プールも適用できます。デベロッパーは、Amazon Redshift の接続制限を超えずに、アプリケーションのデータアクセスパターンを効果的に満たすことができる戦略を選択する必要があります。

プールモデルを採用した場合は、共有インフラストラクチャ内での操作中に発生し得る一般的な問題に常に注意する必要があります。例えば、データのセキュリティを確保するには、アプリケーションレベルのポリシーを適用して、クロステナントアクセスを制限する必要があります。また、環境のパフォーマンスを継続的にチューニングおよび改善して、特定のテナントが他のテナントのエクスペリエンスを阻害するのを防ぐ必要があります。

アジリティに注目する

マルチテナントストレージオプションのマトリックスは困難を伴う場合があります。柔軟性、分離、管理可能性の最適な組み合わせを表すソリューションを特定するのは難しい場合があります。すべてのオプションを検討することは重要ですが、マルチテナントストレージの考え方に絶えずアジリティを組み込むことも重要です。SaaS 組織の成功は、ソリューションに組み込まれたアジリティの量に左右されることがよくあります。

選択したストレージテクノロジーと分離モデルは、新しい特徴や機能を簡単にデプロイする能力に直接影響します。データの構造とコンテンツの形状は、多くの場合、新機能をサポートするために変更されます。つまり、基盤となるストレージモデルは、ダウンタイムなしでこれらの変更に対応する必要があります。このシームレスな移行をサポートすることに関しては、分離モデルごとに長所と短所があります。オプションを検討する場合は、これらの要因に適切な重みを付けます。

サイロ、ブリッジ、プールの各モデルにはアジリティのフットプリントがありますが、共通の原則を適用することで、アジリティを引き続き維持できます。重要な原則は、テナントデータの 1 回限りのバリエーションを最小限に抑えるという自明のことですが、これが守られない場合があります。例えば、サイロモデルとブリッジモデルでは、ストレージのバリエーションが生じ、1 つの自動化されたイベントの一環として、新しい機能をすべての SaaS のお客様客にプッシュすることが複雑になる場合があります。チームは、マルチテナントストレージ戦略に伴う摩擦を制限しようとして、オートメーションや継続的なデプロイを使用する傾向があります。

ストレージ戦略を決定する場合、ストレージ要件は絶えず変わるという現実を受け入れる必要があります。SaaS のお客様のニーズは動くターゲットであり、今日選択したストレージモデルが明日は通用しない場合があります。また、AWS は絶えず新しい機能やサービスを導入し、ストレージのアプローチを強化する新しい機会として提供しています。

まとめ

SaaS のお客様のストレージニーズは単純ではありません。SaaS の現実として、ビジネスのドメイン、お客様、レガシーの考慮事項は、ビジネスのニーズに最適なマルチテナントストレージオプションの組み合わせを決定する方法に影響します。

すべての環境に例外なく適合する 1 つの戦略はありませんが、SaaS 配信モデルのコア原則により適合するモデルがあることは明らかです。一般的に、プールベースのストレージへのアプローチは、AWS のストレージテクノロジーを問わず、マルチテナント環境の管理と運用に必要な統一アプローチに適合します。すべてのテナントを 1 つの共有リポジトリおよび表現にまとめることで、アプローチの運用とデプロイのフットプリントを合理化および統一し、すべてのテナントにわたるヘルスとパフォーマンスを確認できます。

サイロモデルとブリッジモデルは、間違いなく有用であり、一部の SaaS プロバイダーにとっては必要不可欠です。ここで重要な点は、特に対処しない場合、アジリティが複雑になりがちなことです。一部の AWS ストレージテクノロジーは、分離されたテナントストレージスキームをより適切にサポートできます。例えば、RDS でのサイロモデルの構築は、DynamoDB での構築よりも複雑ではありません。通常、パーティション分割モデルとして連結アカウントを使用するたびに、プロビジョニング、管理、スケーリングの課題が増えます。

マルチテナンシーを達成する仕組みとは別に、各 AWS ストレージテクノロジーのプロファイルが、マルチテナントアプリケーションの機能のさまざまなニーズにどのように適合するかを検討してください。テナントがデータにアクセスする方法と、テナントのニーズを満たすためにデータの形を進化させる方法を検討します。アプリケーションを分解して複数の自律型サービスにするほど、サービスごとに個別のストレージ戦略を選択して採用することが容易になります。

これらのサービスと分割スキームを検討すると、パターンと転換点をよりよく理解し、マルチテナントのストレージ戦略を選択しやすくなります。AWS は、SaaS プロバイダーに対してサービスおよび構造の豊富な選択肢を提供します。これらを組み合わせることで、あらゆるマルチテナントストレージニーズに対応できます。

寄稿者

本書の執筆に当たり、次の人物および組織が寄稿しました。

- Tod Golding、AWS パートナープログラム、パートナーソリューションアーキテクト
- Clinton Ford、DynamoDB、シニアプロダクトマーケティングマネージャー
- Zach Christopherson、Amazon Redshift、データベースエンジニア
- Brian Welker、RDS MySQL および MariaDB、プリンシパルプロダクトオーナー

ドキュメントの改訂

日付	説明
2021 年 5 月 6 日	技術的な正確性についてレビュー
2016 年 11 月 1 日	初版発行