マルチテナント SaaS 認可と API アクセスコントロール: 実装オプションとベストプラクティス

AWS 規範ガイダンス

AWS 規範ガイダンス: マルチテナント SaaS 認可と API アクセスコントロール: 実装オプションとベストプラクティス

Table of Contents

序章 1
ターゲットを絞ったビジネス成果 2
テナント分離とマルチテナント認可 3
アクセスコントロールのタイプ 4
RBAC 4
ABAC 4
RBAC-ABAC ハイブリッドアプローチ 5
アクセスコントロールモデルの比較 5
PDP の実装 6
Amazon Verified Permissions の使用 6
Cedar の概要 8
例 1: Verified Permissions と Cedar を使用した基本的な ABAC 9
例 2: Verified Permissions と Cedar を使用した基本的な RBAC 15
例 3: RBAC によるマルチテナントアクセスコントロール 18
例 4: RBAC と ABAC によるマルチテナントアクセスコントロール 23
例 5: Verified Permissions と Cedar を使用した UI フィルタリング 28
OPA の使用 30
Rego の概要 32
例 1: OPA と Rego を使用した基本的な ABAC 33
例 2: OPA と Rego を使用したマルチテナントアクセスコントロールとユーザー定義 RBAC 37
例 3: OPA と Rego を使用した RBAC と ABAC のマルチテナントアクセスコントロール 41
例 4: OPA と Rego を使用した UI フィルタリング 43
カスタムポリシーエンジンの使用 46
PEP の実装 47
認可決定のリクエスト 47
認可決定の評価 48
マルチテナント SaaS アーキテクチャの設計モデル 49
Amazon Verified Permissions の使用 49
API での PEPs での一元化された PDP の使用 APIs 49
Cedar SDK の使用 51
OPA の使用 51
API での PEPs での一元化された PDP の使用 APIs 51
API での PEPsでの分散 PDP の使用 APIs 54

分散 PDP をライブラリとして使用する 56
Amazon Verified Permissions マルチテナント設計に関する考慮事項 57
テナントオンボーディングとユーザーテナント登録 58
テナントごとのポリシーストア 58
1 つの共有マルチテナントポリシーストア 64
階層型デプロイモデル 69
OPA マルチテナント設計に関する考慮事項 72
集中型デプロイパターンと分散型デプロイパターンの比較 72
OPA ドキュメントモデルによるテナント分離 73
テナントオンボーディング 75
PDP の DevOps、モニタリング、ログ記録、およびデータの取得 78
Amazon Verified Permissions での PDP の外部データの取得 79
OPA での PDP の外部データの取得 81
OPA バンドル 81
OPA レプリケーション (データのプッシュ) 81
OPA 動的データ取得 82
OPA での実装に認可サービスを使用する 82
テナントの分離とデータのプライバシーに関する推奨事項 83
Amazon Verified Permissions 84
OPA 84
ベストプラクティス 85
アプリケーションで動作するアクセスコントロールモデルを選択する 85
PDP の実装 85
アプリケーション内のすべての API に PEPsを実装する 85
PDP のポリシーエンジンとして Amazon Verified Permissions または OPA を使用することを検討してください。 85
DevOps、モニタリング、ログ記録用の OPA のコントロールプレーンを実装する 86
Verified Permissions でログ記録とオブザーバビリティ機能を設定する 86
CI/CD パイプラインを使用して、Verified Permissions でポリシーストアとポリシーをプロビジョニングおよび更新する 86
認可の決定に外部データが必要かどうかを判断し、それに対応するモデルを選択する 87
よくある質問 88
次のステップ 92
リソース 93
ドキュメント履歴 95
用語集 96

..... 96
A 97
B 100
C 102
D 105
E 109
F 111
G 113
H 114
I 115
L 118
M 119
O 123
P 126
Q 129
R 129
S 132
T 136
U 137
V 138
W 138
Z 139
..... cxli

マルチテナント SaaS 認可と API アクセスコントロール: 実装オプションとベストプラクティス

Tabby Ward、Thomas Davis、Gideon Landeman、Tomas Riha、Amazon Web Services (AWS)

2024 年 5 月 (ドキュメント履歴)

認可と API アクセスコントロールは、多くのソフトウェアアプリケーション、特にマルチテナント Software as a Service (SaaS) アプリケーションにとって課題です。この複雑さは、保護する必要があるマイクロサービス APIs の急増と、さまざまなテナント、ユーザー特性、アプリケーションの状態から発生する多数のアクセス条件を考慮すると明らかです。これらの問題に効果的に対処するために、ソリューションはマイクロサービス、バックエンド for Frontend (BFF) レイヤー、およびマルチテナント SaaS アプリケーションのその他のコンポーネントによって提供される多くの APIs にわたってアクセスコントロールを適用する必要があります。このアプローチには、多くの要因と属性に基づいて複雑なアクセス決定を行うことができるメカニズムが伴う必要があります。

従来、API アクセスコントロールと認可はアプリケーションコードのカスタムロジックによって処理されていました。このアプローチはエラーが発生しやすく、安全ではありませんでした。このコードにアクセスできる開発者が誤って、または意図的に認可ロジックを変更し、不正アクセスにつながる可能性があるためです。アプリケーションコードのカスタムロジックによって行われた決定を監査することは困難でした。これは、監査人が特定の標準を維持する上での有効性を判断するためにカスタムロジックに専念する必要があるためです。さらに、API アクセスコントロールは、保護 APIs する API の数が少ないため、一般的に不要でした。マイクロサービスやサービス指向アーキテクチャを優先するアプリケーション設計のパラダイムシフトにより、認可とアクセスコントロールの形式を使用する必要がある APIs の数が増えました。さらに、マルチテナント SaaS アプリケーションでテナントベースのアクセスを維持する必要性には、テナントを維持するために追加の認可の課題があります。このガイドで概説されているベストプラクティスには、いくつかの利点があります。

- 認可ロジックは一元化され、プログラミング言語に固有ではない高レベルの宣言言語で記述できます。
- 認可ロジックはアプリケーションコードから抽象化され、アプリケーションのすべての APIs に繰り返し可能なパターンとして適用できます。
- 抽象化は、開発者による認可ロジックへの偶発的な変更を防ぎます。
- SaaS アプリケーションへの統合は、一貫性があり、シンプルです。
- 抽象化により、API エンドポイントごとにカスタム認可ロジックを記述する必要がなくなります。

- 監査は簡素化されます。これは、監査人がアクセス許可を決定するためにコードを確認する必要がなくなったためです。
- このガイドで概説されているアプローチは、組織の要件に応じて複数のアクセスコントロールパラダイムの使用をサポートします。
- この認可とアクセスコントロールのアプローチにより、SaaS アプリケーションの API レイヤーでテナントデータの分離を簡単に維持できます。
- ベストプラクティスは、認可に関してテナントのオンボーディングとオフボーディングに一貫したアプローチを提供します。
- このアプローチでは、このガイドで説明されているように、利点と欠点の両方を持つさまざまな認可デプロイモデル (プールまたはサイロ) を提供します。

ターゲットを絞ったビジネス成果

この規範的なガイダンスでは、マルチテナント SaaS アプリケーションに実装できる認可と API アクセスコントロールの反復可能な設計パターンについて説明します。このガイダンスは、複雑な認可要件または厳格な API アクセスコントロールのニーズを持つアプリケーションを開発するすべてのチームを対象としています。このアーキテクチャでは、ポリシー決定ポイント (PDP) またはポリシーエンジンの作成と、APIs。PDP を作成するための 2 つの具体的なオプションについて説明します。Cedar SDK での Amazon Verified Permissions の使用と、Rego ポリシー言語での Open Policy Agent (OPA) の使用です。このガイドでは、属性ベースのアクセスコントロール (ABAC) モデルまたはロールベースのアクセスコントロール (RBAC) モデル、または両方のモデルの組み合わせに基づいてアクセスを決定する方法についても説明します。このガイドに記載されている設計パターンと概念を使用して、マルチテナント SaaS アプリケーションにおける認可と API アクセスコントロールの実装を通知および標準化することをお勧めします。このガイダンスは、以下のビジネス成果を達成するのに役立ちます。

- マルチテナント SaaS アプリケーション用の標準化された API 認可アーキテクチャ – このアーキテクチャは、ポリシーが保存および管理されるポリシー管理ポイント (Splunk)、認可決定に達するためにそれらのポリシーが評価されるポリシー決定ポイント (PDP)、その決定を適用するポリシー適用ポイント (PEP) の 3 つのコンポーネントを区別します。ホストされた認可サービスである Verified Permissions は、JavaScript と PDP の両方として機能します。または、Cedar や OPA などのオープンソースエンジンを使用して PDP を自分で構築することもできます。
- アプリケーションからの認可ロジックの分離 – 認可ロジックは、アプリケーションコードに埋め込まれている場合や、アドホックな強制メカニズムを通じて実装されている場合、意図しないクロステナントデータアクセスやその他のセキュリティ違反を引き起こす偶発的または悪意のある変更

の対象となる可能性があります。これらの可能性を軽減するために、Verified Permissions などの " を使用して、アプリケーションコードとは独立して認可ポリシーを保存し、それらのポリシーの管理に強力なガバナンスを適用できます。ポリシーは高レベルの宣言言語で一元管理できるため、アプリケーションコードの複数のセクションにポリシーを埋め込む場合よりも、認可ロジックの維持がはるかにシンプルになります。このアプローチにより、更新が一貫して適用されるようになります。

- アクセスコントロールモデルへの柔軟なアプローチ – ロールベースのアクセスコントロール (RBAC)、属性ベースのアクセスコントロール (ABAC)、または両方のモデルの組み合わせはすべて、アクセスコントロールに対する有効なアプローチです。これらのモデルは、さまざまなアプローチを使用して、ビジネスの認可要件を満たそうとしています。このガイドでは、これらのモデルを比較して比較し、組織に適したモデルを選択するのに役立ちます。このガイドでは、これらのモデルが OPA/Rego や Cedar などのさまざまな認可ポリシー言語にどのように適用されるかについても説明します。このガイドで説明されているアーキテクチャにより、一方または両方のモデルが正常に採用されます。
- 厳格な API アクセスコントロール – このガイドでは、最小限の労力でアプリケーション内で APIs を一貫して広く保護する方法を提供します。これは、アプリケーション内通信を容易にするために一般的に多数の APIs を使用するサービス指向またはマイクロサービスアプリケーションアーキテクチャに特に役立ちます。厳格な API アクセスコントロールは、アプリケーションのセキュリティを強化し、攻撃や悪用に対する脆弱性を軽減します。

テナント分離とマルチテナント認可

このガイドでは、テナント分離とマルチテナント認可の概念について説明します。テナント分離とは、共有インフラストラクチャで運用されている場合でも、各テナントのリソースを分離するために SaaS システムで使用する明示的なメカニズムを指します。マルチテナント認可とは、インバウンドアクションを承認し、間違ったテナントに実装されないようにすることです。架空のユーザーは認証および認可され、引き続き別のテナントのリソースにアクセスできます。認証と認可はこのアクセスをブロックしません。この目標を達成するためにテナント分離を実装する必要があります。これらの 2 つの概念の違いの詳細については、ホワイトペーパーの「テナント分離」セクションを参照してください。 SaaS

アクセスコントロールのタイプ

ロールベースのアクセスコントロール (RBAC) と属性ベースのアクセスコントロール (ABAC) の 2 つの広く定義されたモデルを使用して、アクセスコントロールを実装できます。各モデルには利点と欠点があり、このセクションで簡単に説明します。使用するモデルは、特定のユースケースによって異なります。このガイドで説明するアーキテクチャは、両方のモデルをサポートしています。

RBAC

ロールベースのアクセスコントロール (RBAC) は、通常はビジネスロジックに沿ったロールに基づいてリソースへのアクセスを決定します。アクセス許可は、必要に応じてロールに関連付けられます。例えば、マーケティングロールは、制限されたシステム内でマーケティングアクティビティを実行することをユーザーに許可します。これは、認識しやすいビジネスロジックによく合うため、実装する比較的シンプルなアクセスコントロールモデルです。

RBAC モデルは、次の場合に効果が低くなります。

- 複数のロールを含む責任を持つ一意のユーザーがいます。
- ロールの定義が困難になる複雑なビジネスロジックがある。
- 大きなサイズにスケールアップするには、アクセス許可を継続的に管理し、新規および既存のロールにマッピングする必要があります。
- 認可は動的パラメータに基づいています。

ABAC

属性ベースのアクセスコントロール (ABAC) は、属性に基づいてリソースへのアクセスを決定します。属性は、ユーザー、リソース、環境、またはアプリケーションの状態に関連付けることもできます。ポリシーまたはルールは属性を参照し、基本的なブールロジックを使用して、ユーザーがアクションを実行できるかどうかを判断できます。アクセス許可の基本的な例を次に示します。

支払いシステムでは、財務部門のすべてのユーザーが営業時間*/payments*内に API エンドポイントで支払いを処理できます。

財務部門のメンバーシップは、 へのアクセスを決定するユーザー属性です/payments。API / paymentsエンドポイントに関連付けられたリソース属性もあり、営業時間内にのみアクセスを許可

します。ABAC では、ユーザーが支払いを処理できるかどうかは、財務部門のメンバーシップをユーザー属性として、時刻を のリソース属性として含むポリシーによって決まります/payments。

ABAC モデルは、動的、コンテキスト、きめ細かな認可の決定を可能にするという点で非常に柔軟です。ただし、ABAC モデルを最初に実装することは困難です。関連するすべてのアクセスベクトルのルールとポリシーを定義し、属性を列挙するには、実装にかなりの先行投資が必要です。

RBAC-ABAC ハイブリッドアプローチ

RBAC と ABAC を組み合わせると、両方のモデルの利点の一部が得られます。RBAC は、ビジネスロジックに非常に密接に連携しているため、ABAC よりも簡単に実装できます。承認の決定を行う際に、詳細度のレイヤーを追加するには、ABAC と RBAC を組み合わせることができます。このハイブリッドアプローチでは、ユーザーのロール (および割り当てられたアクセス許可) を追加の属性と組み合わせてアクセスを決定し、アクセスに関する決定を行います。両方のモデルを使用すると、権限の管理と割り当てが簡単になり、承認の決定に関する柔軟性と詳細度が向上します。

アクセスコントロールモデルの比較

次の表は、前に説明した 3 つのアクセスコントロールモデルを比較したものです。この比較は、有益で高レベルであることを意図しています。特定の状況でアクセスモデルを使用すると、この表で行った比較と必ずしも相関しない場合があります。

係数	RBAC	ABAC	ハイブリッド
柔軟性	Medium	高い	高
シンプルさ	高	低	Medium
詳細度	低	高	中程度
動的な決定とルール	いいえ	あり	あり
コンテキスト対応	いいえ	あり	ある程度
実装の労力	低	高	中程度

PDP の実装

ポリシー決定ポイント (PDP) は、ポリシーまたはルールエンジンとして特徴付けることができます。このコンポーネントは、ポリシーまたはルールを適用し、特定のアクセスを許可するかどうかの決定を返す責任があります。PDP は、ロールベースのアクセスコントロール (RBAC) モデルと属性ベースのアクセスコントロール (ABAC) モデルで機能できますが、PDP は ABAC の要件です。PDP を使用すると、アプリケーションコードの認可ロジックを別のシステムにオフロードできます。これにより、アプリケーションコードを簡素化できます。また、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認を決定するためのeasy-to-use繰り返し可能なインターフェイスも提供します。

以下のセクションでは、PDP を実装するための 3 つの方法について説明します。ただし、これは完全なリストではありません。

PDP 実装方法:

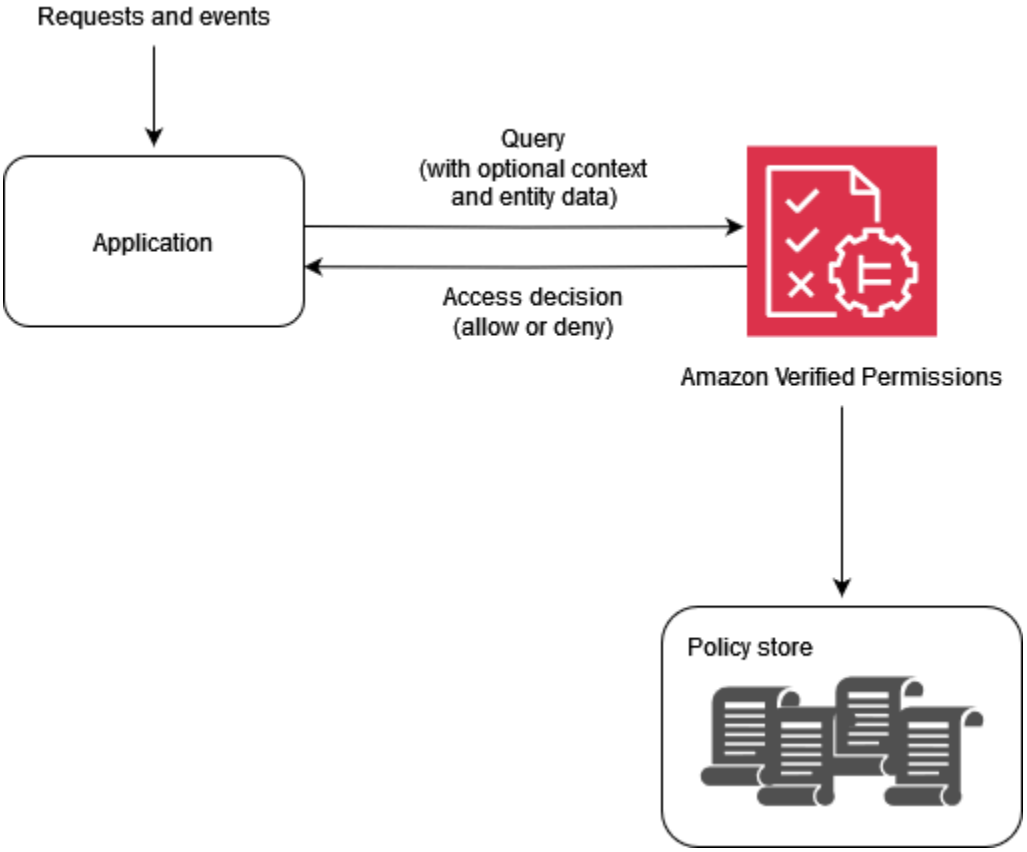
- Amazon Verified Permissions を使用した PDP の実装
- OPA を使用した PDP の実装
- カスタムポリシーエンジンの使用

Amazon Verified Permissions を使用した PDP の実装

Amazon Verified Permissions は、ポリシー決定ポイント (PDP) を実装するために使用できる、スケーラブルできめ細かなアクセス許可の管理および認可サービスです。ポリシーエンジンとして、アプリケーションがユーザーアクションをリアルタイムで検証し、過度に特権的または無効なアクセス許可を強調表示するのに役立ちます。これは、認可を外部化し、ポリシーの管理と管理を一元化することで、開発者がより安全なアプリケーションを迅速に構築するのに役立ちます。認可ロジックをアプリケーションロジックから分離することで、Verified Permissions はポリシーのデカップリングをサポートします。

Verified Permissions を使用して PDP を実装し、アプリケーション内で最小特権と継続的な検証を実装することで、開発者はアプリケーションへのアクセスをゼロトラストの原則に合わせることができます。さらに、セキュリティチームと監査チームは、アプリケーション内のどのリソースにアクセスできるかをより適切に分析および監査できます。Verified Permissions は、専用でセキュリティファーストのオープンソースポリシー言語である Cedar を使用して、よりきめ細かなコンテキスト対応アクセスコントロールのために、ロールベースのアクセスコントロール (RBAC) と属性ベースのアクセスコントロール (ABAC) に基づいてポリシーベースのアクセスコントロールを定義します。

Verified Permissions は、Amazon Cognito、Google、Facebook などの複数の ID プロバイダーを使用してマルチテナント認可を有効にする機能など、SaaS アプリケーションにいくつかの便利な機能を提供します。SaaS アプリケーションに特に役立つもう 1 つの Verified Permissions 機能は、テナントごとのカスタムロールのサポートです。顧客関係管理 (CRM) システムを設計する場合、1 つのテナントは、1 つの特定の基準セットに基づいて、販売機会ごとにアクセスの粒度を定義できます。別のテナントに別の定義がある場合があります。Verified Permissions の基盤となるアクセス許可システムは、これらのバリエーションをサポートできるため、SaaS ユースケースの優れた候補となります。Verified Permissions は、すべてのテナントに適用されるポリシーを記述する機能もサポートしているため、ガードレールポリシーを適用して SaaS プロバイダーとしての不正アクセスを防ぐことは簡単です。



Verified Permissions を使用する理由

Amazon Cognito などの ID プロバイダーで Verified Permissions を使用すると、アプリケーション用のより動的なポリシーベースのアクセス管理ソリューションを実現できます。データのセキュリティ、機密性、プライバシーを維持しながら、ユーザーによる情報の共有とコラボレーションを支援するアプリケーションを構築できます。Verified Permissions は、アイデンティティとリソースのロールと属性に基づいてアクセスを強制するきめ細かな認可システムを提供することで、運用コスト

を削減します。ポリシーモデルを定義し、ポリシーを作成して一元的に保存し、アクセスリクエストをミリ秒単位で評価できます。

Verified Permissions では、Cedar というシンプルで人間が読める宣言言語を使用してアクセス許可を表現できます。Cedar で記述されたポリシーは、各チームのアプリケーションで使用されるプログラミング言語に関係なく、チーム間で共有できます。

Verified Permissions を使用する際の考慮事項

Verified Permissions では、プロビジョニングの一環としてポリシーを作成し、自動化できます。アプリケーションロジックの一部として、実行時にポリシーを作成することもできます。ベストプラクティスとして、テナントのオンボーディングとプロビジョニングの一部としてポリシーを作成するときは、継続的インテグレーションと継続的デプロイ (CI/CD) パイプラインを使用してポリシーバージョンを管理、変更、追跡する必要があります。または、アプリケーションはポリシーバージョンを管理、変更、追跡できますが、アプリケーションロジックは本質的にこの機能を実行しません。アプリケーションでこれらの機能をサポートするには、この機能を実装するようにアプリケーションを明示的に設計する必要があります。

認可決定に到達するために他のソースから外部データを提供する必要がある場合は、このデータを取得し、認可リクエストの一部として Verified Permissions に提供する必要があります。このサービスでは、追加のコンテキスト、エンティティ、属性はデフォルトでは取得されません。

Cedar の概要

Cedar は、柔軟で拡張可能でスケーラブルなポリシーベースのアクセスコントロール言語であり、開発者がアプリケーションのアクセス許可をポリシーとして表現するのに役立ちます。管理者と開発者は、ユーザーがアプリケーションリソースを操作することを許可または禁止するポリシーを定義できます。1 つのリソースに複数のポリシーをアタッチできます。アプリケーションのユーザーがリソースに対してアクションを実行しようとすると、アプリケーションは Cedar ポリシーエンジンに認可をリクエストします。Cedar は該当するポリシーを評価し、 `ALLOW`または `DENY`の決定を返します。Cedar は、任意のタイプのプリンシパルとリソースの認可ルールをサポートし、ロールベースのアクセスコントロール (RBAC) と属性ベースのアクセスコントロール (ABAC) を許可し、自動推論ツールによる分析をサポートします。

Cedar では、ビジネスロジックと認可ロジックを分離できます。アプリケーションのコードからリクエストを行うときは、Cedar の認可エンジンを呼び出して、リクエストが承認されているかどうかを判断します。承認されている場合 (決定は `ALLOW`)、アプリケーションはリクエストされたオペレーションを実行できます。承認されていない場合 (決定は `DENY`)、アプリケーションはエラーメッセージを返す可能性があります。Cedar の主な機能は次のとおりです。

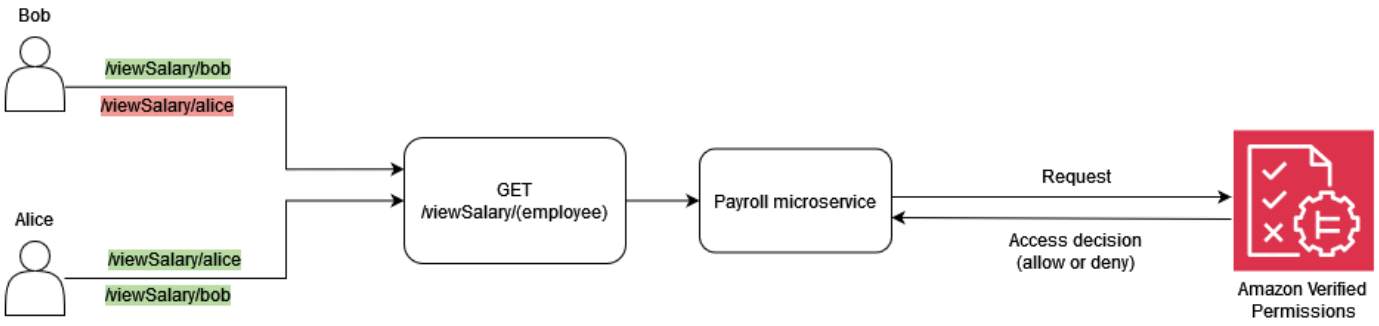
- 表現性 – Cedar は認可のユースケースをサポートするために特別に設計されており、人間の読みやすさを念頭に置いて開発されました。
- パフォーマンス – Cedar は、迅速な取得のためのインデックス作成ポリシーをサポートし、制限されたレイテンシーで高速でスケーラブルなリアルタイム評価を提供します。
- 分析 – Cedar は、ポリシーを最適化し、セキュリティモデルを検証できる分析ツールをサポートしています。

詳細については、Cedar のウェブサイトを参照してください。

例 1: Verified Permissions と Cedar を使用した基本的な ABAC

このシナリオ例では、Amazon Verified Permissions を使用して、架空のペイロールマイクロサービス内の情報にアクセスできるユーザーを決定します。このセクションでは、Cedar を使用してアクセスコントロールの決定をレンダリングする方法を示す Cedar コードスニペットについて説明します。これらの例は、Cedar および Verified Permissions が提供する機能を完全に探索することを目的としたものではありません。Cedar の詳細な概要については、Cedar のドキュメントを参照してください。

次の図では、 `viewSalaryGET`メソッドに関連付けられた 2 つの一般的なビジネスルールを適用します。従業員は自分の給与を表示でき、従業員は自分の直属の従業員の給与を表示できます。Verified Permissions ポリシーを使用して、これらのビジネスルールを適用できます。



従業員は自分の給与を表示できます。

Cedar では、基本的なコンストラクトは、プリンシパル、アクション、またはリソースを表すエンティティです。認可リクエストを行い、Verified Permissions ポリシーを使用して評価を開始するには、プリンシパル、アクション、リソース、エンティティのリストを指定する必要があります。

- プリンシパル (`principal`) は、ログインしているユーザーまたはロールです。
- アクション (`action`) は、リクエストによって評価されるオペレーションです。

- リソース (resource) は、アクションがアクセスするコンポーネントです。
- エンティティのリスト (entityList) には、リクエストの評価に必要なすべてのエンティティが含まれます。

従業員が自分の給与を表示できるビジネスルールを満たすために、次のような Verified Permissions ポリシーを指定できます。

```
permit (
    principal,
    action == Action::"viewSalary",
    resource
)
when {
    principal == resource.owner
};
```

このポリシーは、 Actionが ALLOWでありviewSalary、リクエスト内のリソースにプリンシパルと等しい属性所有者があるかどうかを に評価します。たとえば、Bob が給与レポートをリクエストしたログインユーザーであり、給与レポートの所有者でもある場合、ポリシーは に評価されますALLOW。

次の認可リクエストが Verified Permissions に送信され、サンプルポリシーによって評価されます。この例では、Bob はviewSalaryリクエストを行うログインユーザーです。したがって、Bob はエンティティタイプ のプリンシパルですEmployee。Bob が実行しようとしているアクションは viewSalary,でviewSalary、表示されるリソースはタイプ Salary-BobですSalary。BobがSalary-Bobリソースを表示できるかどうかを評価するには、 タイプを の値 Bob (プリンシパル) Employee と タイプを持つリソースの所有者属性にリンクするエンティティ構造を指定する必要がありますSalary。この構造は で指定します。ここでentityList、 に関連付けられた属性Salaryには、タイプ Employeeと値 entityIdentifierを含む を指定する所有者が含まれますBob。Verified Permissions は、認可リクエストでprincipal提供された を、決定を行うためにSalaryリソースに関連付けられているowner属性と比較します。

```
{
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Bob"
  },
  "action": {
```

```
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        }
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {}
      }
    ]
  }
}
```

Verified Permissions への認可リクエストは出力として以下を返します。 属性`decision`は `ALLOW`または ですDENY。

```
{
    "determiningPolicies":
        [
            {
                "determiningPolicyId": "PAYROLLAPP_POLICYSTOREID"
            }
```

```
        ],
    "decision": "ALLOW",
    "errors": []
}
```

この場合、Bob は自分の給与を表示しようとしていたため、Verified Permissions に送信された認可リクエストは に評価されますALLOW。ただし、目的は Verified Permissions を使用して 2 つのビジネスルールを適用することでした。以下を示すビジネスルールも当てはまります。

従業員は、自分の部下の給与を表示できます。

このビジネスルールを満たすには、別のポリシーを指定できます。次のポリシーは、アクションがALLOWでありviewSalary、リクエストのリソースにプリンシパルとowner.manager等しい属性があるかどうかを に評価します。たとえば、Alice が給与レポートをリクエストしたログインユーザーで、Alice がレポートの所有者のマネージャーである場合、ポリシーは に評価されますALLOW。

```
permit (
    principal,
    action == Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager
};
```

次の認可リクエストが Verified Permissions に送信され、サンプルポリシーによって評価されます。この例では、Alice はviewSalaryリクエストを行うログインユーザーです。したがって、Alice はプリンシパルであり、エンティティはタイプ ですEmployee。Alice が実行しようとしているアクションは でviewSalary、viewSalary表示されるリソースは の値Salaryを持つ タイプですSalary-Bob。Alice がSalary-Bobリソースを表示できるかどうかを評価するには、 型のEmployee値を Alice manager 属性にリンクするエンティティ構造を指定する必要があります。このエンティティ構造は、 型の owner 属性を の値Salaryに関連付ける必要がありますSalary-Bob。この構造をで指定します。ここでentityList、 に関連付けられた属性Salaryには、タイプ Employeeと値 entityIdentifierを含む を指定する所有者が含まれますBob。Verified Permissions はまず owner 属性をチェックし、 属性は タイプEmployeeと値 に評価されますBob。次に、Verified Permissions は、関連付けられたmanager属性を評価しEmployee、提供されたプリンシパルと比較して認可の決定を行います。この場合、 principal と resource.owner.manager 属性は同等ALLOWであるため、決定は です。

```
{
```

```
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Alice"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "None"
            }
          }
        },
        "parents": []
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        },
        "parents": []
```

```
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Alice"
            }
          }
        },
       "parents": []
      }
    ]
  }
}
```

この例のこれまでのところ、 viewSalaryメソッドに関連する 2 つのビジネスルールが提供されました。従業員は自分の給与を表示でき、従業員は自分の部下の給与を表示でき、Verified Permissions は各ビジネスルールの条件を個別に満たすポリシーとして確認できます。単一の Verified Permissions ポリシーを使用して、両方のビジネスルールの条件を満たすこともできます。

従業員は、自分の給与と、自分の部下の給与を表示できます。

前の認可リクエストを使用する場合、次のポリシーは、アクションALLOWが viewSalaryであり、リクエスト内のリソースに owner.managerと等しい属性があるかprincipal、 とowner等しい属性があるかを に評価しますprincipal。

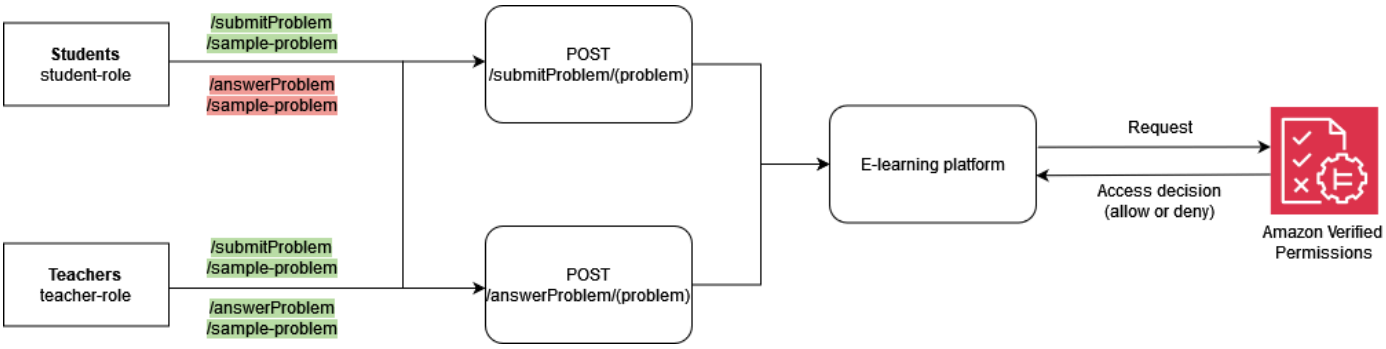
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

たとえば、Alice が給与レポートをリクエストするログインユーザーであり、Alice が所有者のマネージャーまたはレポートの所有者である場合、ポリシーは に評価されますALLOW。

Cedar ポリシーで論理演算子を使用する方法の詳細については、Cedar ドキュメントを参照してください。

例 2: Verified Permissions と Cedar を使用した基本的な RBAC

この例では、Verified Permissions と Cedar を使用して基本的な RBAC を示します。前述のように、Cedar の基本コンストラクトはエンティティです。開発者は独自のエンティティを定義し、オプションでエンティティ間の関係を作成できます。次の例には、Users、、 の 3 種類のエンティティが含まれていますProblems。 Students および は、 タイプのエンティティと見なRole,すTeachersことができRoles、それぞれを 0 または のいずれかに関連付けるUserことができますRoles。



Cedar では、これらの関係は RoleStudentを親UserBobとして にリンクすることで表現されます。この関連付けは、すべての学生ユーザーを 1 つのグループに論理的にグループ化します。Cedar でのグループ化の詳細については、Cedar ドキュメントを参照してください。

次のポリシーは、タイプ Studentsの論理グループにリンクされているsubmitProblem,すべてのプリンシパルALLOWのアクションの決定を評価しますRole。

```
permit (
    principal in ElearningApp::Role::"Students",
    action == ElearningApp::Action::"submitProblem",
    resource
);
```

次のポリシーは、タイプ Teachersの論理グループにリンクされているすべてのプリンシパルについてanswerProblem、ALLOWアクション submitProblemまたは の決定を評価しますRole。

```
permit (
```

```
    principal in ElearningApp::Role::"Teachers",
    action in [
        ElearningApp::Action::"submitProblem",
        ElearningApp::Action::"answerProblem"
    ],
    resource
);
```

これらのポリシーを使用してリクエストを評価するには、評価エンジンは、認可リクエスト内で参照されるプリンシパルが適切なグループのメンバーであるかどうかを知る必要があります。したがって、アプリケーションは認可リクエストの一部として関連するグループメンバーシップ情報を評価エンジンに渡す必要があります。これは entitiesプロパティを通じて行われます。これにより、認可呼び出しに関連するプリンシパルとリソースの属性とグループのメンバーシップデータを Cedar 評価エンジンに提供できます。次のコードでは、グループメンバーシップは、 User::"Bob"を という親として定義することで示されますRole::"Students"。

```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Bob"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Bob"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
                    "entityId": "Students"
```

```
                }
            ]
        },
        {
          "identifier": {
            "entityType": "ElearningApp::Problem",
            "entityId": "SomeProblem"
          },
          "attributes": {},
          "parents": []
        }
      ]
  }
}
```

この例では、Bob はanswerProblemリクエストを行うログインユーザーです。したがって、Bob はプリンシパルであり、エンティティはタイプ ですUser。Bob が実行しようとしているアクションは ですanswerProblem。Bob がanswerProblemアクションを実行できるかどうかを評価するには、エンティティを の値Userにリンクし、親エンティティを としてリストしてグループメンバーシップを割り当てるエンティティ構造を指定する必要がありますRole::"Students"。ユーザーグループのエンティティRole::"Students"はアクションを実行することしか許可されないためsubmitProblem、この認可リクエストは に評価されますDENY。

一方、 の値を持ち、 グループの一部Userである タイプがanswerProblemアクションを実行Role::"Teachers" しようとするAliceと、ポリシーはグループ内のプリンシパルがすべてのリソースanswerProblemに対してアクションを実行することを許可するように指示するためALLOW、認可リクエストRole::"Teachers"は に評価されます。次のコードは、 に評価されるこのタイプの 認可リクエストを示していますALLOW。

```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
```

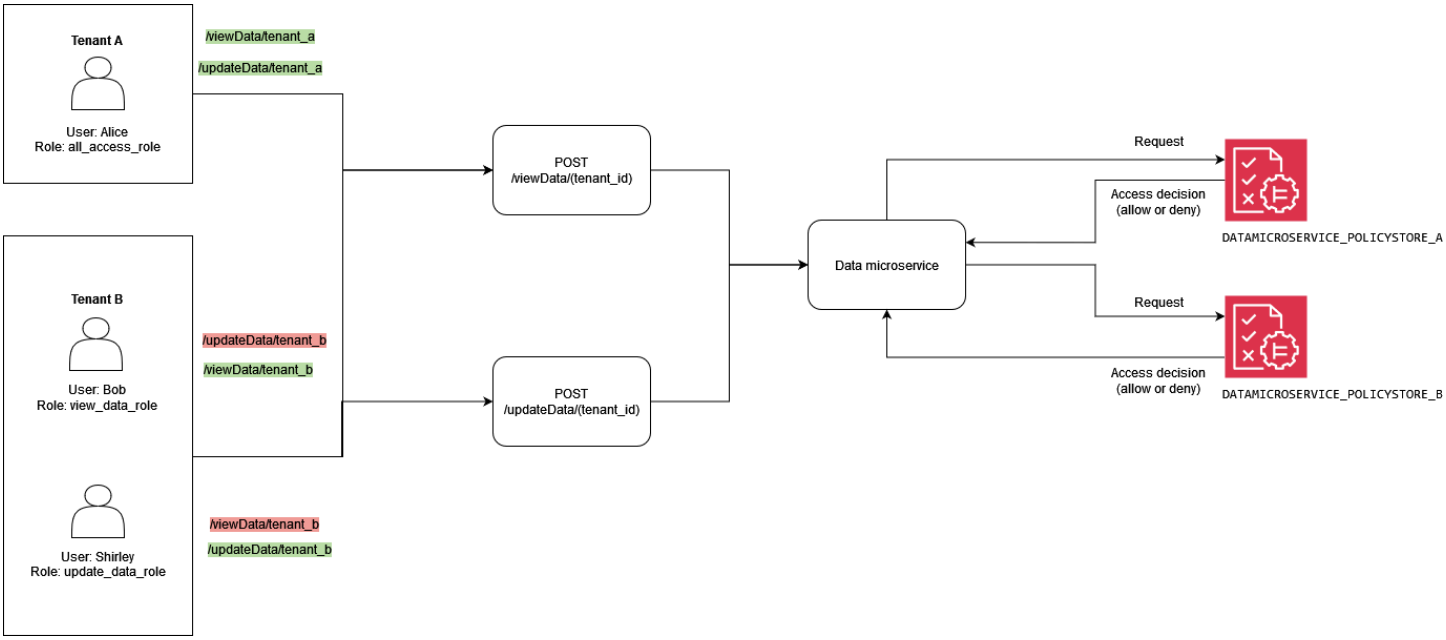
```
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Alice"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
                    "entityId": "Teachers"
                }
            ]
        },
        {
            "identifier": {
                "entityType": "ElearningApp::Problem",
                "entityId": "SomeProblem"
            },
            "attributes": {},
            "parents": []
        }
      ]
  }
}
```

例 3: RBAC によるマルチテナントアクセスコントロール

前の RBAC の例について詳しく説明するために、SaaS プロバイダーの一般的な要件である SaaS マルチテナンシーを含めるように要件を拡張できます。マルチテナントソリューションでは、リソースアクセスは常に特定のテナントに代わって提供されます。つまり、テナント A のユーザーは、データがシステムに論理的または物理的にコロケーションされている場合でも、テナント B のデータを表示できません。次の例は、複数の Verified Permissions ポリシーストアを使用してテナント分離を実装する方法と、ユーザーロールを使用してテナント内のアクセス許可を定義する方法を示しています。

テナントごとのポリシーストアの設計パターンを使用することは、Verified Permissions によるアクセス制御を実装しながら、テナントの分離を維持するためのベストプラクティスです。このシナリオでは、テナント A とテナント B のユーザーリクエ

ストは、DATAMICROSERVICE_POLICYSTORE_Bそれぞれ個別のポリシーストア DATAMICROSERVICE_POLICYSTORE_Aおよび に対して検証されます。マルチテナント SaaS アプリケーションの Verified Permissions 設計上の考慮事項の詳細については、「Verified Permissions マルチテナント設計上の考慮事項」セクションを参照してください。



次のポリシーは、DATAMICROSERVICE_POLICYSTORE_Aポリシーストアにあります。プリンシパルがタイプ allAccessRoleのグループの一部であることを確認しますRole。この場合、プリンシパルはテナント A に関連付けられているすべてのリソースに対して viewDataおよび updateDataアクションを実行できます。

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
    ],
    resource
);
```

次のポリシーは、DATAMICROSERVICE_POLICYSTORE_Bポリシーストアにあります。最初のポリシーは、プリンシパルがタイプ のupdateDataRoleグループの一部であることを確認しますRole。その場合、テナント B に関連付けられているリソースに対してupdateDataアクションを実行するアクセス許可をプリンシパルに付与します。

```
permit (
```

```
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
);
```

この 2 番目のポリシーでは、 タイプのviewDataRoleグループの一部であるプリンシパルに、テナント B に関連付けられているリソースに対して viewDataアクションを実行することを許可Roleする必要があります。

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
);
```

テナント A から行われた認可リクエストは、DATAMICROSERVICE_POLICYSTORE_Aポリシーストアに送信され、そのストアに属するポリシーによって検証される必要があります。この場合、この例の一部として前述した最初のポリシーによって検証されます。この認可リクエストでは、 の値Userを持つ タイプのプリンシパルAliceが viewDataアクションの実行をリクエストしています。プリンシパルはタイプ allAccessRoleのグループに属しますRole。Alice はSampleDataリソースに対して viewDataアクションを実行しようとしています。Alice には allAccessRole口ールがあるため、この評価の結果がALLOW決定されます。

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
```

```
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

代わりに、 によってテナント B から行われたリクエストを表示するとUser Bob、次のような承認リクエストが表示されます。リクエストはテナント B から送信されるため、DATAMICROSERVICE_POLICYSTORE_Bポリシーストアに送信されます。このリクエストでは、プリンシパルはリソース updateDataに対してアクションを実行Bobしたいと考えていますSampleData。ただし、 Bobはそのリソースの updateData アクションにアクセスできるグループの一部ではありません。したがって、リクエストはDENY決定になります。

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_B",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Bob"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
```

```
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Bob"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "viewDataRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

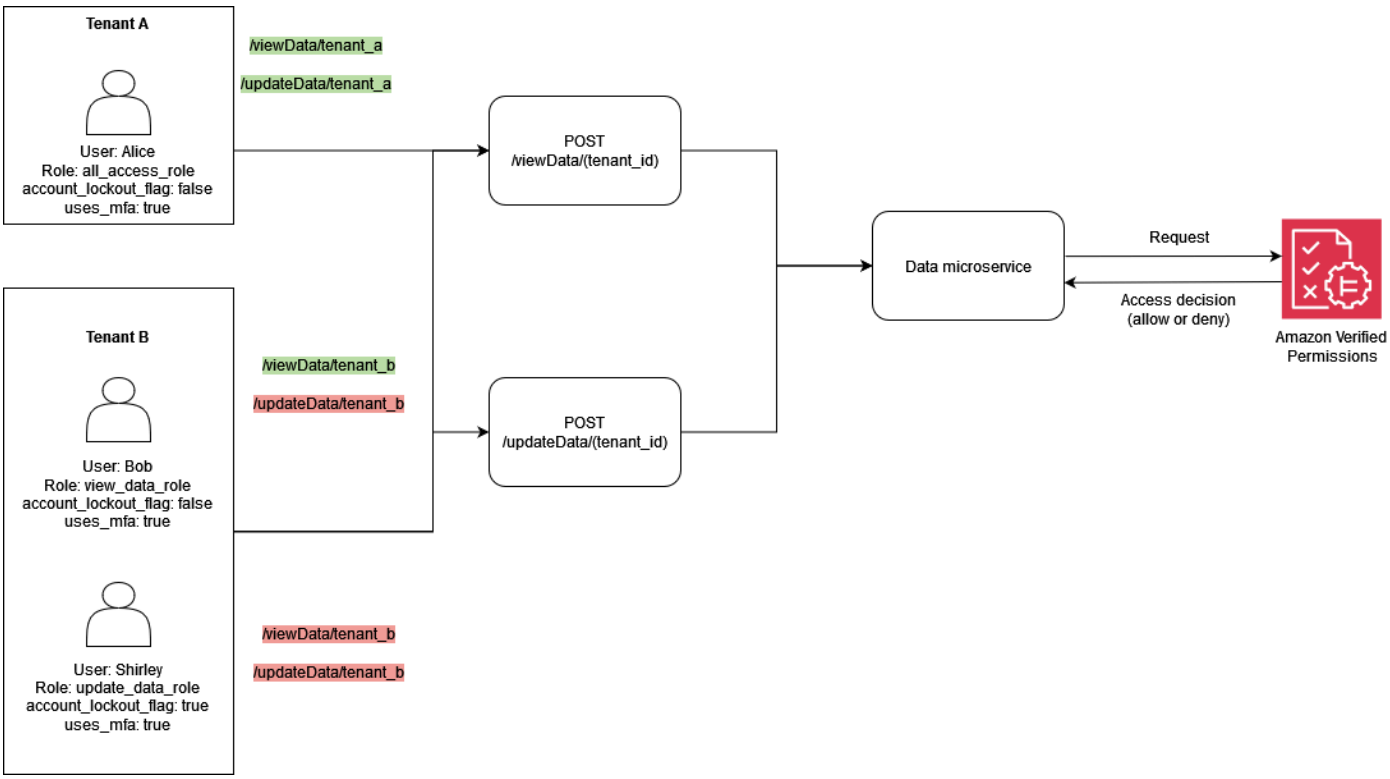
この 3 番目の例では、 はリソース に対して viewDataアクションを実行User Alice しようとしますSampleData。このリクエストは、プリンシパルがテナント A にAlice属しているため、DATAMICROSERVICE_POLICYSTORE_Aポリシーストアに送信されます。 Aliceは allAccessRoleタイプの グループの一部でありRole、 リソースに対してviewDataアクションを実行することを許可します。そのため、リクエストはALLOW決定になります。

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
```

```
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

例 4: RBAC と ABAC によるマルチテナントアクセスコントロール

前のセクションの RBAC の例を強化するために、ユーザーに属性を追加して、マルチテナントアクセスコントロールの RBAC-ABAC ハイブリッドアプローチを作成できます。この例では、前の例と同じロールが含まれていますが、ユーザー属性 `account_lockout_flag`とコンテキストパラメータ を追加します`uses_mfa`。この例では、RBAC と ABAC の両方を使用してマルチテナントアクセスコントロールを実装するアプローチも異なり、テナントごとに異なるポリシーストアではなく 1 つの共有ポリシーストアを使用します。



この例は、前の例と同様に、テナント A とテナント B の承認決定を行う必要があるマルチテナント SaaS ソリューションを表します。

ユーザーロック機能を実装するために、この例では、認可リクエストのUserエンティティプリンシパルaccount_lockout_flagに 属性を追加します。このフラグは、システムへのユーザーアクセスをロックし、ロックアウトされたユーザーに対するDENYすべての権限を付与します。account_lockout_flag 属性はUserエンティティに関連付けられ、フラグが複数のセッションにわたってアクティブに取り消Userされるまで に対して有効です。この例では、 when条件を使用して を評価しますaccount_lockout_flag。

この例では、リクエストとセッションに関する詳細も追加します。コンテキスト情報は、セッションが多要素認証を使用して認証されたことを指定します。この検証を実装するために、この例ではwhen条件を使用して、コンテキストフィールドの一部として uses_mfaフラグを評価します。コンテキストを追加するためのベストプラクティスの詳細については、Cedar ドキュメントを参照してください。

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
```

```
    ],
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

このポリシーは、リソースがリクエスト元のプリンシパルの Tenant 属性と同じグループにある場合を除き、リソースへのアクセスを禁止します。テナント分離を維持するためのこのアプローチは、1 つの共有マルチテナントポリシーストアアプローチと呼ばれます。マルチテナント SaaS アプリケーションの Verified Permissions 設計上の考慮事項の詳細については、「Verified Permissions マルチテナント設計上の考慮事項」セクションを参照してください。

このポリシーは、プリンシパルが のメンバーであることも保証allAccessRoleし、アクションを viewDataおよび に制限しますupdateData。さらに、このポリシーは、account_lockout_flagが falseであり、 のコンテキスト値が にuses_mfa評価されることを確認しますtrue。

同様に、次のポリシーでは、プリンシパルとリソースの両方が同じテナントに関連付けられていることを確認し、クロステナントアクセスを防止します。このポリシーは、プリンシパルが のメンバーであることも保証viewDataRoleし、アクションを に制限しますviewData。さらに、account_lockout_flagが falseであり、 のコンテキスト値が にuses_mfa評価されることを確認しますtrue。

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

3 番目のポリシーは、前のポリシーと似ています。このポリシーでは、リソースが で表されるエンティティに対応するグループのメンバーであることが必要ですprincipal.Tenant。これにより、プリンシパルとリソースの両方がテナント B に関連付けられ、クロステナントアクセスが防止され

ます。このポリシーは、プリンシパルが のメンバーであることを保証updateDataRoleし、アクションを に制限しますupdateData。さらに、このポリシーは、 account_lockout_flagが であり、 のコンテキスト値が にuses_mfa評価falseされることを確認しますtrue。

```
permit (
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

次の認可リクエストは、このセクションで前述した 3 つのポリシーによって評価されます。この認可リクエストでは、 型のプリンシパルUserと の値が のプリンシパルは、ロール を使用してupdateDataリクエストAliceを行いますallAccessRole。 Alice には、値がTenantである 属性がありますTenant::"TenantA"。実行AliceしようとしているアクションはupdateData,で、適用するリソースのタイプは SampleData ですData。 SampleData は親エンティティTenantAとして を持ちます。

ポリシーストアの最初の<DATAMICROSERVICE_POLICYSTOREID>ポリシーに従って、 はポリシーの when句の条件が満たされていると仮定して、リソースに対して updateData アクションを実行Aliceできます。最初の条件では、 principal.Tenant 属性を に評価する必要がありますTenantA。2 番目の条件では、プリンシパルの 属性を account_lockout_flagにする必要がありますfalse。最後の条件では、コンテキストを にuses_mfaする必要がありますtrue。3 つの条件がすべて満たされるため、リクエストはALLOW決定を返します。

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
```

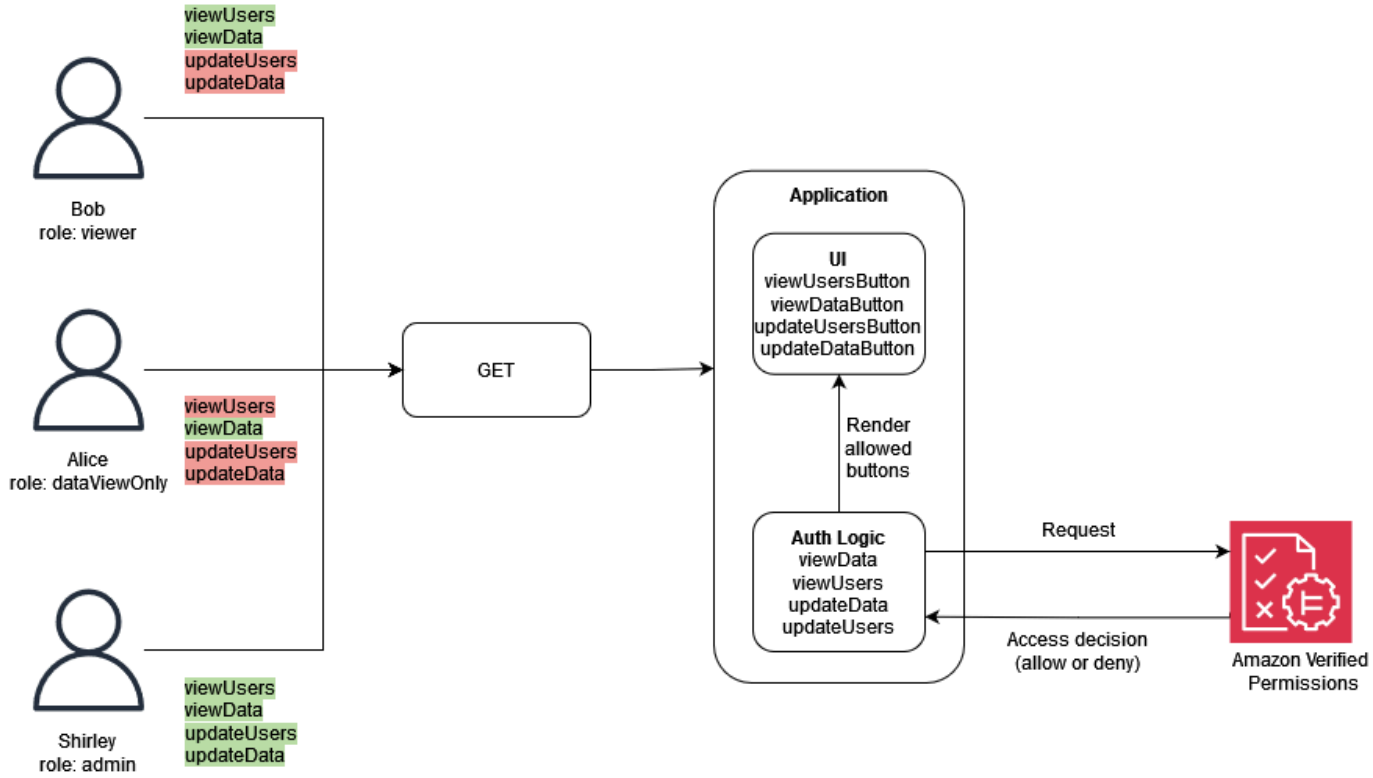
```
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "uses_mfa": {
            "boolean": true
        }
    }
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {
            {
                "account_lockout_flag": {
                    "boolean": false
                },
                "Tenant": {
                   "entityIdentifier": {
                        "entityType":"MultitenantApp::Tenant",
                        "entityId":"TenantA"
                   }
                }
            }
        },
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
        },
     {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": [
            {
```

```
                "entityType": "MultitenantApp::Tenant",
                "entityId": "TenantA"
            }
        ]
      }
    ]
  }
}
```

例 5: Verified Permissions と Cedar を使用した UI フィルタリング

Verified Permissions を使用して、承認されたアクションに基づいて UI 要素の RBAC フィルタリングを実装することもできます。これは、マルチテナント SaaS アプリケーションの場合に、特定のユーザーまたはテナントに関連付けられる可能性のあるコンテキスト依存の UI 要素を持つアプリケーションにとって非常に役立ちます。

次の例では、 Usersの Roleviewerは更新を実行できません。これらのユーザーの場合、UI は更新ボタンをレンダリングしないでください。



この例では、単一ページのウェブアプリケーションには 4 つのボタンがあります。表示されるボタンは、現在アプリケーションにログインしているRoleユーザーの によって異なります。単一ページ

のウェブアプリケーションが UI をレンダリングすると、Verified Permissions にクエリを実行して、ユーザーが実行を許可されているアクションを決定し、承認の決定に基づいてボタンを生成します。

次のポリシーでは、値が の タイプRoleがユーザーとデータの両方を表示viewerできることを指定します。このポリシーの承認ALLOW決定には、 viewDataまたは viewUsersアクションが必要であり、リソースをタイプ Dataまたは に関連付ける必要もありますUsers。ALLOW 決定により、UI は viewDataButtonと の 2 つのボタンをレンダリングできますviewUsersButton。

```
permit (
    principal in GuiAPP::Role::"viewer",
    action in [GuiAPP::Action::"viewData", GuiAPP::Action::"viewUsers"],
    resource
)
when {
   resource in [GuiAPP::Type::"Data", GuiAPP::Type::"Users"]
};
```

次のポリシーでは、値が の 型Roleがデータのみを表示viewerDataOnlyできることを指定します。このポリシーの承認ALLOW決定には viewDataアクションが必要であり、 タイプに関連付けるリソースも必要ですData。ALLOW 決定により、UI はボタン をレンダリングできますviewDataButton。

```
permit (
    principal in GuiApp::Role::"viewerDataOnly",
    action in [GuiApp::Action::"viewData"],
    resource in [GuiApp::Type::"Data"]
);
```

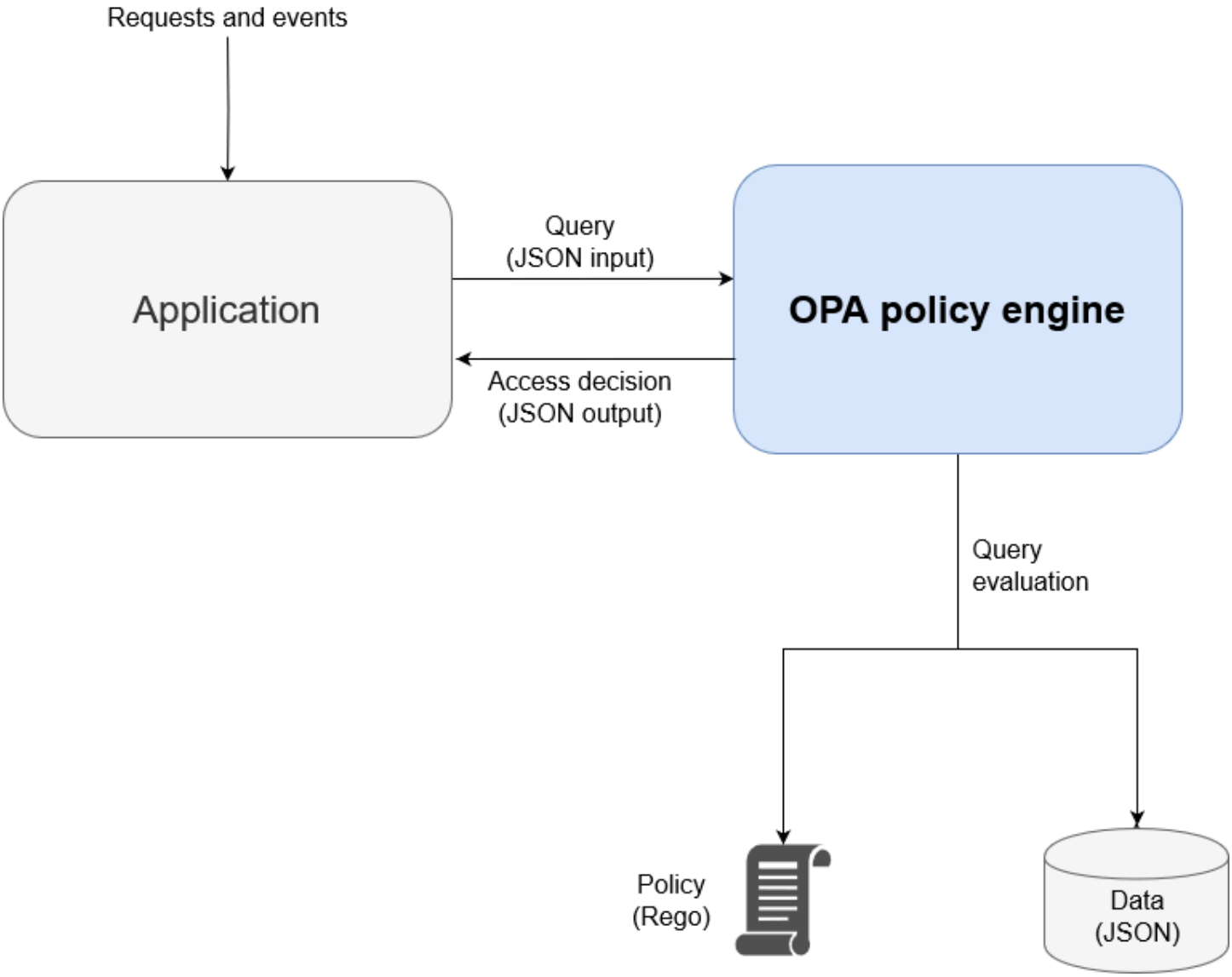
次のポリシーでは、値が の タイプRoleがデータとユーザーを編集して表示adminできるように指定します。このポリシーの承認ALLOW決定には、updateData、、updateUsersviewData,または のアクションが必要です。またviewUsers、リソースをタイプ Dataまたは に関連付ける必要がありますUsers。ALLOW 決定により、UI は 、updateDataButton、updateUsersButton、viewDataButtonの 4 つのボタンをすべてレンダリングできますviewUsersButton。

```
permit (
    principal in GuiApp::Role::"admin",
    action in [
        GuiApp::Action::"updateData",
        GuiApp::Action::"updateUsers",
        GuiApp::Action::"viewData",
```

```
        GuiApp::Action::"viewUsers"
       ],
    resource
)
when {
   resource in [GuiApp::Type::"Data", GuiApp::Type::"Users"]
};
```

OPA を使用した PDP の実装

Open Policy Agent (OPA) は、オープンソースの汎用ポリシーエンジンです。OPA には多くのユースケースがありますが、PDP 実装に関連するユースケースは、認可ロジックをアプリケーションから切り離す機能です。これはポリシーのデカップリングと呼ばれます。OPA は、いくつかの理由で PDP の実装に役立ちます。ポリシーとルールのドラフトには、Rego という高レベルの宣言言語を使用します。これらのポリシーとルールはアプリケーションとは別に存在し、アプリケーション固有のロジックなしで認可の決定をレンダリングできます。また、OPA は RESTful API を公開して、認可の決定を簡単かつ簡単に取得できるようにします。認可の決定を行うために、アプリケーションは JSON 入力を使用して OPA をクエリし、OPA は指定されたポリシーに対して入力を評価して JSON でアクセス決定を返します。OPA は、認可の決定に関連する可能性のある外部データをインポートすることもできます。



OPA には、カスタムポリシーエンジンよりもいくつかの利点があります。

- Rego による OPA とそのポリシー評価は、承認の決定に必要なポリシーとデータの挿入のみを必要とする、柔軟で構築済みのポリシーエンジンを提供します。このポリシー評価ロジックは、カスタムポリシーエンジンソリューションで再作成する必要があります。
- OPA は、宣言言語でポリシーを記述することで、認可ロジックを簡素化します。これらのポリシーとルールは、アプリケーション開発スキルなしで、アプリケーションコードとは独立して変更および管理できます。
- OPA は RESTful API を公開し、ポリシー適用ポイント (PEPs。
- OPA は、JSON ウェブトークン (JWTs) の検証とデコードの組み込みサポートを提供します。

- OPA は承認された認可標準です。つまり、特定の問題を解決するために支援や調査が必要な場合は、ドキュメントや例が豊富です。
- OPA などの認可標準を採用することで、チームのアプリケーションで使用されるプログラミング言語に関係なく、Rego で記述されたポリシーをチーム間で共有できます。

OPA が自動的に提供しないものは 2 つあります。

- OPA には、ポリシーを更新および管理するための堅牢なコントロールプレーンはありません。OPA は、管理 API を公開することでポリシーの更新、モニタリング、ログ集約を実装するための基本的なパターンを提供しますが、この API との統合は OPA ユーザーが処理する必要があります。ベストプラクティスとして、継続的インテグレーションと継続的デプロイ (CI/CD) パイプラインを使用して、ポリシーバージョンを管理、変更、追跡し、OPA でポリシーを管理する必要があります。
- デフォルトでは、OPA は外部ソースからデータを取得できません。認可決定の外部データソースは、ユーザー属性を保持するデータベースである可能性があります。外部データを OPA に提供する方法には柔軟性があります。事前にローカルにキャッシュすることも、認可決定がリクエストされたときに API から動的に取得することもできますが、この情報を取得することは、OPA がユーザーに代わって実行できることではありません。

Rego の概要

Rego は汎用ポリシー言語です。つまり、スタックの任意のレイヤーと任意のドメインで機能します。Rego の主な目的は、インフラストラクチャリソース、アイデンティティ、およびオペレーションに関するポリシー対応の決定を行うために評価される JSON/YAML 入力とデータを受け入れることです。Rego を使用すると、言語の変更や拡張を必要とせずに、スタックまたはドメインの任意のレイヤーに関するポリシーを記述できます。Rego が実行できる決定の例を以下に示します。

- この API リクエストは許可または拒否されますか?
- このアプリケーションのバックアップサーバーのホスト名は何ですか?
- この提案されたインフラストラクチャ変更のリスクスコアは何ですか?
- 高可用性を実現するために、このコンテナをどのクラスターにデプロイする必要がありますか?
- このマイクロサービスにはどのようなルーティング情報を使用する必要がありますか?

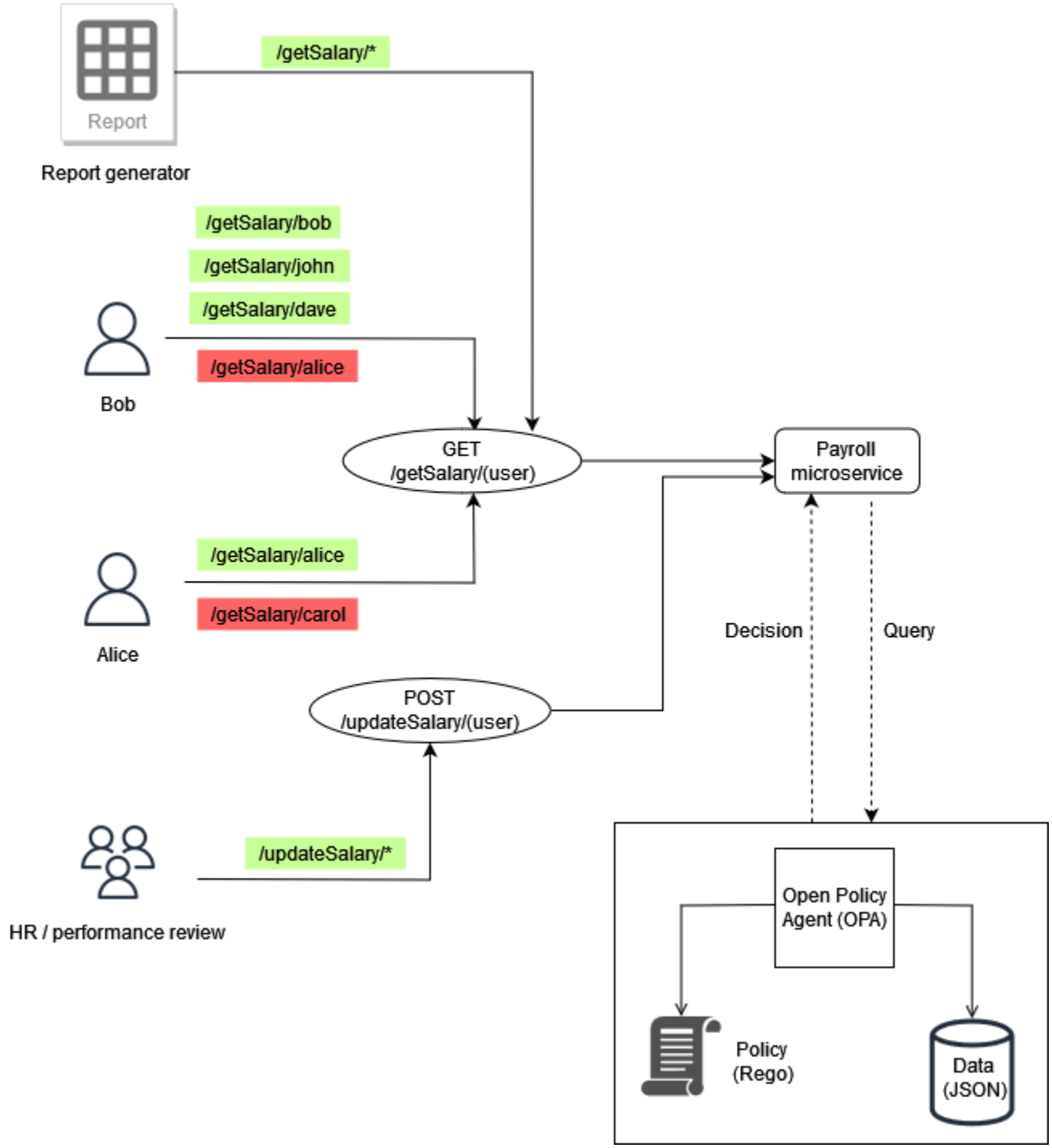
これらの質問に答えるために、Rego はこれらの決定方法に関する基本的な哲学を採用しています。Rego でポリシーを作成するときの 2 つの重要な原則は次のとおりです。

- すべてのリソース、アイデンティティ、またはオペレーションは、JSON または YAML データとして表現できます。
- ポリシーは、データに適用される口ジックです。

Rego は、JSON/YAML データの入力の評価方法に関する口ジックを定義することで、ソフトウェアシステムが認可を決定するのに役立ちます。この問題の一般的な解決策は C、Java、Go、Python などのプログラミング言語ですが、Rego はシステムを表すデータと入力、およびこの情報を使用してポリシーを決定するための口ジックに焦点を当てるように設計されています。

例 1: OPA と Rego を使用した基本的な ABAC

このセクションでは、架空のペイロールマイクロサービス内の情報へのアクセスをどのユーザーに許可するかについて、OPA を使用してアクセスを決定するシナリオについて説明します。Rego コードスニペットは、Rego を使用してアクセスコントロールの決定をレンダリングする方法を示すために提供されています。これらの例は、Rego および OPA 機能の網羅的な調査でも、完全な調査でもありません。Rego の詳細については、OPA ウェブサイトの Rego ドキュメントを参照することをお勧めします。



基本的な OPA ルールの例

前の図では、ペイロールマイクロサービスに対して OPA によって適用されるアクセスコントロールルールの 1 つを以下に示します。

従業員は自分の給与を読み取ることができます。

Bob が給与マイクロサービスにアクセスして自分の給与を確認しようとすると、給与マイクロサービスは API コールを OPA RESTful API にリダイレクトしてアクセスを決定できます。ペイロールサービスは、次の JSON 入力を使用して OPA に決定をクエリします。

```
{
    "user": "bob",
    "method": "GET",
    "path": ["getSalary", "bob"]
}
```

OPA は、クエリに基づいてポリシーを選択します。この場合、次のポリシーは Rego で記述され、JSON 入力を評価します。

```
default allow = false
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}
```

このポリシーは、デフォルトでアクセスを拒否します。次に、クエリ内の入力をグローバル変数 にバインドして評価しますinput。ドット演算子は、変数の値にアクセスするためにこの変数とともに使用されます。ルールの式も true の場合、Rego ルールは true allowを返します。Rego ルールは、入力method の が GET と等しいことを確認します。次に、リスト内の 2 番目の要素を変数 に割り当てるgetSalary前に、リスト内の最初の要素path が であることを確認しますuser。最後に、アクセスするパスが、リクエストuserを行う が user変数input.userと一致することを確認する/getSalary/bob ことによって であることを確認します。このルールallowは if-then ロジックを適用して、出力に示すようにブール値を返します。

```
{
    "allow": true
}
```

外部データを使用した部分的なルール

追加の OPA 機能を実証するために、適用するアクセスルールに要件を追加できます。前の図のコンテキストでこのアクセスコントロール要件を適用するとします。

従業員は、自分の部下の給与を読み取ることができます。

この例では、OPA は、アクセス決定を行うためにインポートできる外部データにアクセスできます。

```
"managers": {
        "bob": ["dave", "john"],
        "carol": ["alice"]
}
```

OPA で部分的なルールを作成することで、任意の JSON レスポンスを生成できます。このルールは、固定レスポンスの代わりに値のセットを返します。これは部分的なルールの例です。

```
direct_report[user_ids] {
    user_ids = data.managers[input.user][_]
}
```

このルールは、 の値に報告するすべてのユーザーのセットを返します。この場合input.user、 はですbob。ルールの [_]コンストラクトは、セットの値を繰り返し処理するために使用されます。これはルールの出力です。

```
{
    "direct_report": [
      "dave",
      "john"
    ]
}
```

この情報を取得すると、ユーザーがマネージャーの直属部下であるかどうかを判断できます。一部のアプリケーションでは、シンプルなブールレスポンスを返すよりも動的 JSON を返すことをお勧めします。

まとめ

最後のアクセス要件は、両方の要件で指定された条件を組み合わせるため、最初の 2 つよりも複雑です。

従業員は、自分の給与と自分の部下の給与を読み取ることができます。

この要件を満たすには、次の Rego ポリシーを使用できます。

```
default allow = false
```

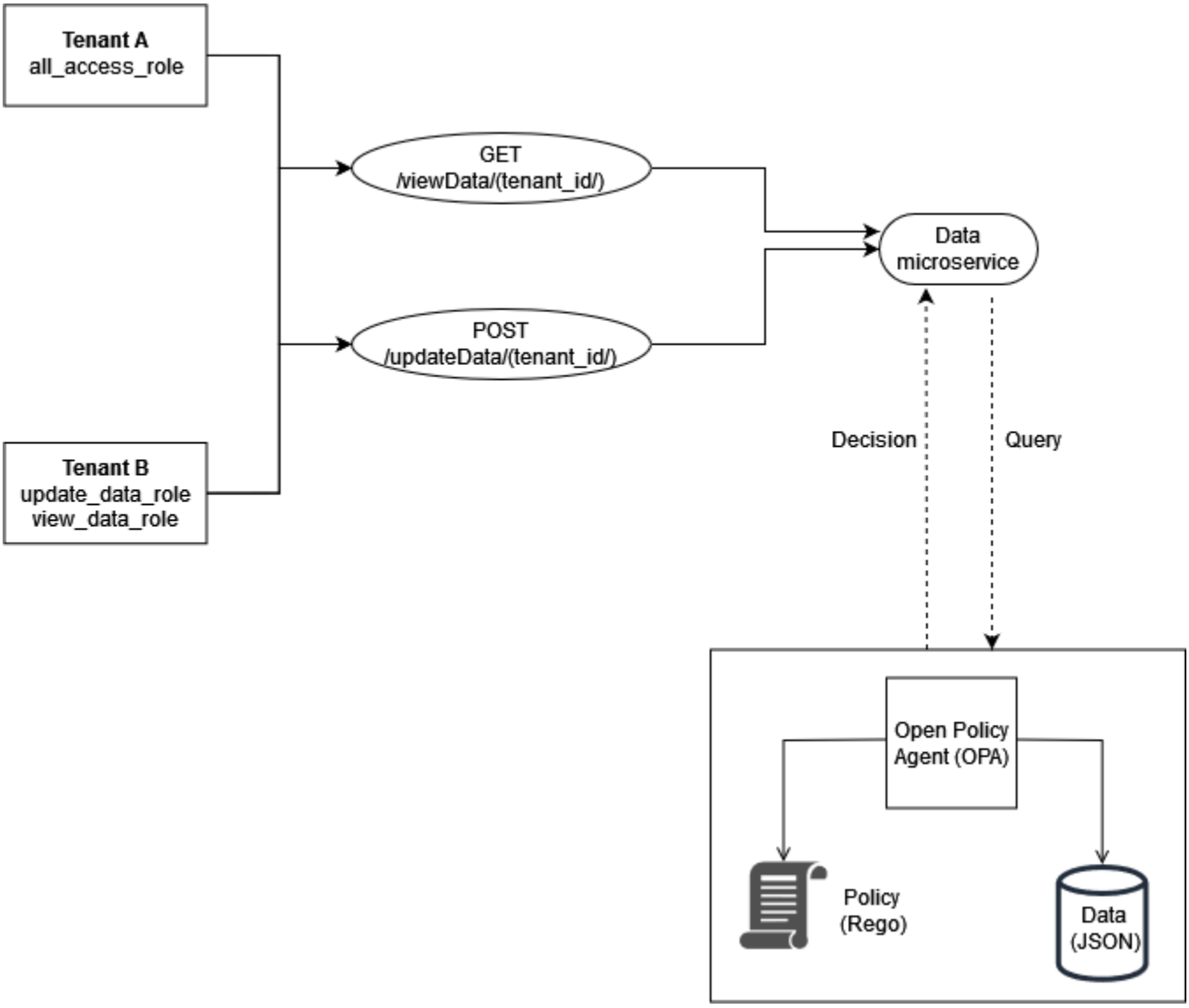
```
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    managers := data.managers[input.user][_]
    contains(managers, user)
}
```

ポリシーの最初のルールでは、前述のように、自分の給与情報を表示しようとするすべてのユーザーが にアクセスできます。同じ名前の 2 つのルールを持つ はallow、Rego の論理または演算子として機能します。2 番目のルールは、 input.user (前の図のデータから) に関連付けられているすべてのダイレクトレポートのリストを取得し、このリストを managers変数に割り当てます。最後に、ルールは、自分の名前が managers変数に含まれていることを確認input.userして、給与を確認しようとしているユーザーが の直接レポートであるかどうかをチェックします。

このセクションの例は非常に基本的なものであり、Rego と OPA の機能を完全にまたは徹底的に調べることはできません。詳細については、OPA ドキュメントを参照し、OPA GitHub README ファイルを参照し、Rego プレイグラウンドで実験してください。

例 2: OPA と Rego を使用したマルチテナントアクセスコントロールとユーザー定義 RBAC

この例では、OPA と Rego を使用して、テナントユーザーが定義したカスタムロールを持つマルチテナントアプリケーションの API でアクセスコントロールを実装する方法を示します。また、テナントに基づいてアクセスを制限する方法も示します。このモデルは、OPA が高レベルのロールで提供される情報に基づいてきめ細かなアクセス許可の決定を行う方法を示しています。



テナントのロールは、OPA のアクセス決定を行うために使用される外部データ (RBAC データ) に保存されます。

```
{
    "roles": {
        "tenant_a": {
            "all_access_role": ["viewData", "updateData"]
        },
        "tenant_b": {
            "update_data_role": ["updateData"],
            "view_data_role": ["viewData"]
        }
    }
```

```
}
```

これらのロールは、テナントユーザーが定義する場合、テナント定義のロールをアクセス許可とテナント自体にマッピングする際に信頼できるソースとして機能する外部データソースまたは ID プロバイダー (IdP) に保存する必要があります。

この例では、OPA で 2 つのポリシーを使用して認可の決定を行い、これらのポリシーがテナント分離をどのように適用するかを調べます。これらのポリシーは、前に定義した RBAC データを使用します。

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
}
```

このルールがどのように機能するかを表示するには、次の入力を持つ OPA クエリを検討してください。

```
{
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET"
}
```

この API コールの承認決定は、RBAC データ、OPA ポリシー、および OPA クエリ入力を組み合わせることで、次のように行われます。

1. のユーザーは、 への API コールTenant Aを行います/viewData/tenant_a。
2. データマイクロサービスは呼び出しを受け取り、allowViewDataルールをクエリして、OPA クエリ入力の例に示されている入力を渡します。
3. OPA は、OPA ポリシーでクエリされたルールを使用して、提供された入力を評価します。また、OPA は RBAC データからのデータを使用して入力を評価します。OPA は以下を実行します。

   a. API コールの実行に使用されるメソッドが であることを確認しますGET。

b. リクエストされたパスが であることを確認しますviewData。

c. パスtenant_idの がユーザーinput.tenant_idに関連付けられている と等しいことを確認します。これにより、テナントの分離が維持されます。同じロールを持つ別のテナントは、この API コールの実行を承認できません。

d. ロールの外部データからロールのアクセス許可のリストを取得し、変数 に割り当てますrole_permissions。このリストは、 のユーザーに関連付けられているテナント定義のロールを使用して取得されます。 input.role.

e. アクセス許可が含まれているrole_permissionsかどうかを確認します viewData.

4. OPA は、データマイクロサービスに次の決定を返します。

```
{
    "allowViewData": true
}
```

このプロセスは、RBAC とテナントの認識が OPA による認可の決定にどのように貢献できるかを示しています。この点をさらに説明するために、次のクエリ入力/viewData/tenant_bを使用してへの API コールを検討してください。

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["viewData", "tenant_b"],
    "method": "GET"
}
```

このルールは OPA クエリ入力と同じ出力を返しますが、ロールが異なる別のテナント用です。これは、この呼び出しが 用/tenant_bであり、RBAC データview_data_role内の にはまだそれに関連付けられたviewDataアクセス許可があるためです。に同じタイプのアクセスコントロールを適用するには/updateData、同様の OPA ルールを使用できます。

```
default allowUpdateData = false
allowUpdateData = true {
    input.method == "POST"
    input.path = ["updateData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "updateData")
```

```
}
```

このルールは機能的にはallowViewDataルールと同じですが、別のパスと入力メソッドを検証します。このルールは引き続きテナントの分離を保証し、テナント定義のロールが API 発信者に許可を付与することを確認します。これがどのように適用されるかを確認するには、 への API コールの次のクエリ入力を調べます/updateData/tenant_b。

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["updateData", "tenant_b"],
    "method": "POST"
}
```

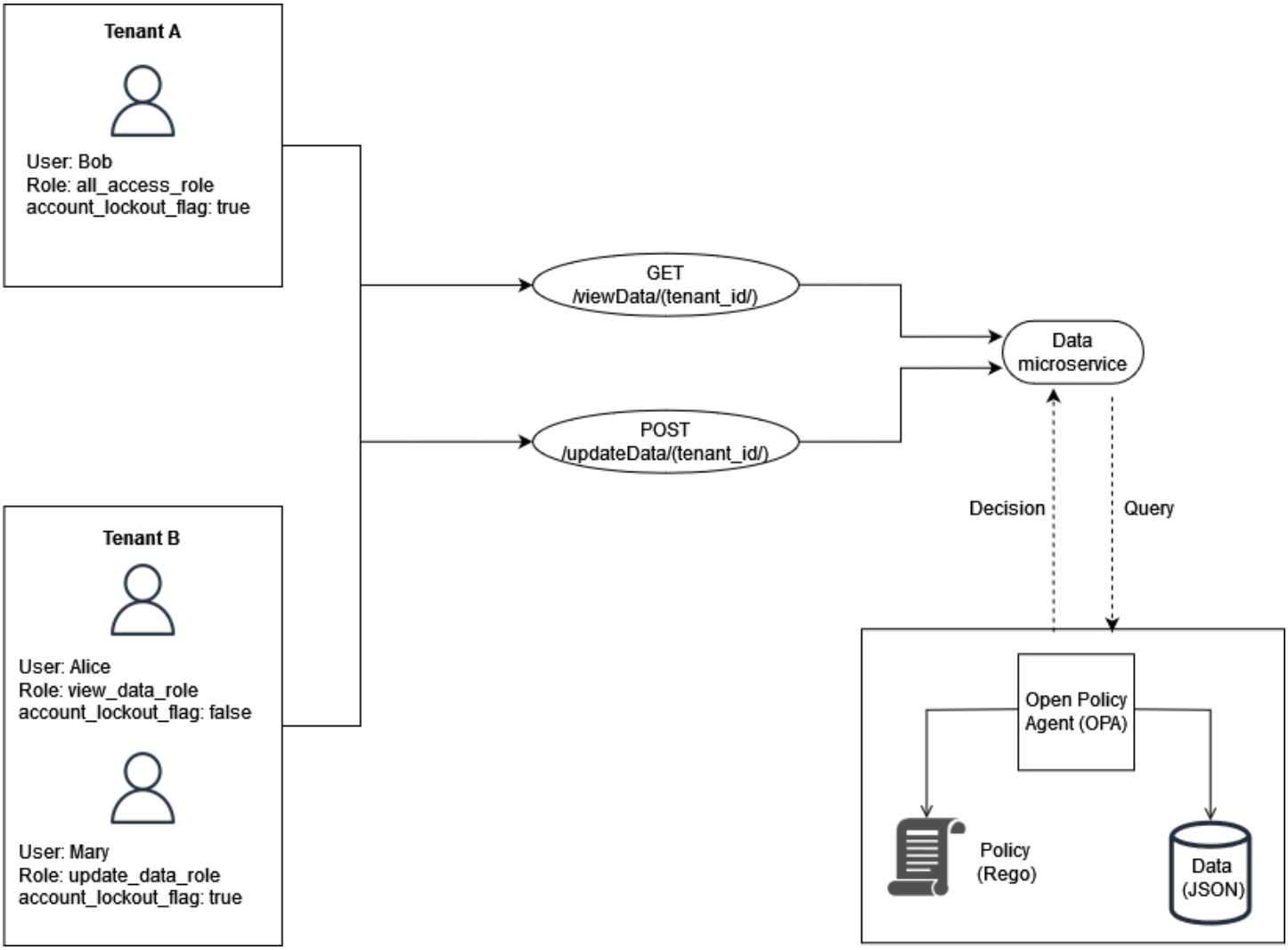
このクエリ入力は、 allowUpdateData ルールで評価されると、次の認可決定を返します。

```
{
    "allowUpdateData": false
}
```

この呼び出しは承認されません。API 発信者は正しい に関連付けられtenant_idており、承認された方法を使用して API を呼び出していますが、 input.roleはテナント定義の ですview_data_role。には アクセスupdateData許可view_data_roleがないため、 への呼び出し/updateDataは許可されていません。この呼び出しは、 を持つtenant_bユーザーに対して成功しましたupdate_data_role。

例 3: OPA と Rego を使用した RBAC と ABAC のマルチテナントアクセスコントロール

前のセクションの RBAC の例を強化するために、ユーザーに属性を追加できます。



この例では、前の例と同じロールが含まれていますが、ユーザー属性 を追加しますaccount_lockout_flag。これは、特定のロールに関連付けられていないユーザー固有の属性です。この例で以前に使用したものと同じ RBAC 外部データを使用できます。

```
 {
    "roles": {
        "tenant_a": {
            "all_access_role": ["viewData", "updateData"]
        },
        "tenant_b": {
            "update_data_role": ["updateData"],
            "view_data_role": ["viewData"]
        }
    }
}
```

account_lockout_flag ユーザー属性は、ユーザー Bob /viewData/tenant_aの の OPA クエリへの入力の一部としてデータサービスに渡すことができます。

```
 {
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET",
    "account_lockout_flag": "true"
}
```

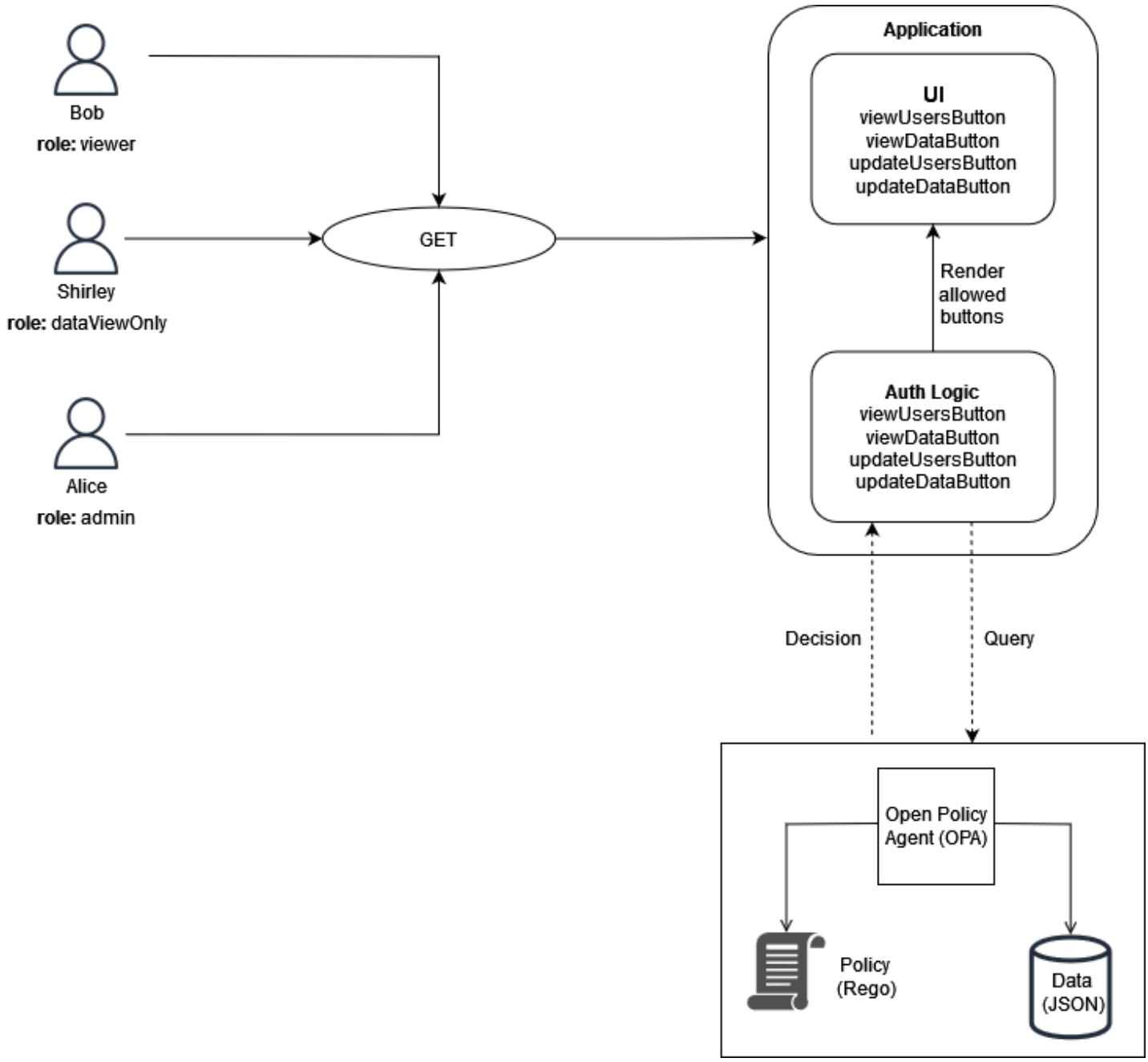
アクセス決定のためにクエリされるルールは、前の例と似ていますが、 account_lockout_flag 属性をチェックするための追加の行が含まれています。

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
    input.account_lockout_flag == "false"
}
```

このクエリは、 の承認決定を返しますfalse。これは、 account_lockout_flag attributeが Bob true用であり、Bob allowViewData に正しいロールとテナントがあるにもかかわらず、Rego ルールがアクセスを拒否するためです。

例 4: OPA と Rego を使用した UI フィルタリング

OPA と Rego の柔軟性は、UI 要素をフィルタリングする機能をサポートします。次の例は、OPA 部分ルールが RBAC を使用して UI に表示する要素について承認を決定する方法を示しています。このメソッドは、OPA で UI 要素をフィルタリングできるさまざまな方法の 1 つです。



この例では、単一ページのウェブアプリケーションには 4 つのボタンがあります。Bob の UI、Shirley の UI、Alice の UI をフィルタリングして、ロールに対応するボタンのみを表示するとします。UI はユーザーからリクエストを受信すると、OPA 部分ルールをクエリして、UI に表示するボタンを決定します。Bob (ロール `viewer`) が UI にリクエストを行うと、クエリは OPA に入力として以下を渡します。

```
{
    "role": "viewer"
```

```
}
```

OPA は、RBAC 用に構造化された外部データを使用してアクセスを決定します。

```
{
    "roles": {
        "viewer": ["viewUsers", "viewData"],
        "dataViewOnly": ["viewData"],
        "admin": ["viewUsers", "viewData", "updateUsers", "updateData"]
    }
}
```

OPA 部分ルールは、外部データと入力の両方を使用して、許可されたアクションのリストを生成します。

```
user_permissions[permissions] {
    permissions := data.roles[input.role][_]
}
```

部分ルールでは、OPA はクエリの一部としてinput.role指定された を使用して、表示するボタンを決定します。Bob にはロール がありviewer、外部データはビューワーに viewUsersと の 2 つのアクセス許可があることを指定しますviewData。したがって、Bob (およびビューワーロールを持つ他のユーザー) のこのルールの出力は次のとおりです。

```
{
    "user_permissions": [
        "viewData",
        "viewUsers"
    ]
}
```

dataViewOnly ロールを持つ Shirley の出力には、アクセス許可ボタン が含まれますviewData。admin ロールを持つ Alice の出力には、これらのアクセス許可がすべて含まれます。これらのレスポンスは、OPA が に対してクエリされると UI に返されますuser_permissions。その後、アプリケーションはこのレスポンスを使用して、viewUsersButton、、viewDataButtonupdateUsersButton、および を非表示または表示できますupdateDataButton。

カスタムポリシーエンジンの使用

PDP を実装するもう 1 つの方法は、カスタムポリシーエンジンを作成することです。このポリシーエンジンの目的は、認可ロジックをアプリケーションから切り離すことです。カスタムポリシーエンジンは、Verified Permissions や OPA と同様に、ポリシーのデカップリングを達成するための認可の決定を行います。このソリューションと Verified Permissions または OPA の主な違いは、ポリシーを記述および評価するためのロジックがカスタムポリシーエンジン用にカスタム構築されていることです。エンジンとのやり取りは、認可の決定がアプリケーションに到達できるようにするために、API またはその他の方法を通じて公開する必要があります。カスタムポリシーエンジンは、任意のプログラミング言語で記述することも、共通表現言語 (CEL) などのポリシー評価の他のメカニズムを使用することもできます。

PEP の実装

ポリシー適用ポイント (PEP) は、評価のためにポリシー決定ポイント (PDP) に送信される認可リクエストを受信する責任があります。PEP は、データとリソースを保護する必要があるアプリケーション、または認可ロジックが適用されるアプリケーション内の任意の場所に配置できます。PEPs は PDPs。PEP は、認可決定のリクエストと評価にのみ責任を負い、認可ロジックは必要ありません。PEPsは PDPs とは異なり、SaaS アプリケーションに一元化することはできません。これは、アプリケーションとそのアクセスポイント全体に認可とアクセスコントロールを実装する必要があるためです。PEPsは、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはアクセスコントロールが望ましいか必要なアプリケーション内の任意のポイントに適用できます。PEPs をアプリケーションに広めることで、認可が複数のポイントで頻繁に独立して検証されます。

PEP を実装するための最初のステップは、アプリケーションでアクセスコントロールの適用を行う場所を決定することです。PEPsをアプリケーションに統合する場所を決定するときは、この原則を考慮してください。

アプリケーションが API を公開する場合は、その API に対する認可とアクセスコントロールが必要です。

これは、マイクロサービス指向またはサービス指向のアーキテクチャでは、APIsさまざまなアプリケーション関数の区切り文字として機能するためです。アプリケーション関数間の論理チェックポイントとしてアクセスコントロールを含めることは理にかなっています。SaaS アプリケーション内の各 API にアクセスするための前提条件として PEPs を含めることを強くお勧めします。アプリケーションの他のポイントで認可を統合することもできます。モノリシックアプリケーションでは、PEPsをアプリケーション自体のロジックに統合する必要がある場合があります。PEPs を含める場所は 1 つではありませんが、API の原則を出発点として使用することを検討してください。

認可決定のリクエスト

PEP は PDP に認可決定をリクエストする必要があります。リクエストにはいくつかの形式があります。認可決定をリクエストする最も簡単でアクセスしやすい方法は、PDP (OPA または Verified Permissions) によって公開されている RESTful API に認可リクエストまたはクエリを送信することです。Verified Permissions を使用している場合は、 AWS SDK を使用して IsAuthorized メソッドを呼び出し、認可決定を取得することもできます。このパターンの PEP の唯一の機能は、認可リクエストまたはクエリが必要とする情報を転送することです。これは、API が受信したリクエストを PDP への入力として転送するのと同じくらい簡単です。PEPs を作成する方法は他にもあります。

例えば、API を使用する代わりに、OPA PDP を Go プログラミング言語で記述されたアプリケーションとライブラリとしてローカルに統合できます。

認可決定の評価

PEPs には、認可決定の結果を評価するロジックを含める必要があります。PDPs が APIs として公開されると、認可の決定は JSON 形式で API コールによって返される可能性があります。PEP はこの JSON コードを評価して、実行されているアクションが承認されているかどうかを判断する必要があります。例えば、PDP がブール型の許可または拒否認可の決定を提供するように設計されている場合、PEP はこの値をチェックし、許可の場合は HTTP ステータスコード 200、拒否の場合は HTTP ステータスコード 403 を返すだけです。API にアクセスするための前提条件として PEP を組み込むこのパターンは、SaaS アプリケーション全体にアクセス制御を実装するための、簡単に実装され、非常に効果的なパターンです。より複雑なシナリオでは、PEP が PDP によって返される任意の JSON コードを評価する責任を負う場合があります。PEP は、PDP が返す認可決定を解釈するために必要なロジックを含めるように記述する必要があります。PEP はアプリケーションのさまざまな場所に実装される可能性が高いため、選択したプログラミング言語で PEP コードを再利用可能なライブラリまたはアーティファクトとしてパッケージ化することをお勧めします。これにより、PEP はアプリケーションのどの時点でも、最小限の再作業で簡単に統合できます。

マルチテナント SaaS アーキテクチャの設計モデル

API アクセスコントロールと認可を実装する方法は多数あります。このガイドでは、マルチテナント SaaS アーキテクチャに効果的な 3 つの設計モデルに焦点を当てています。これらの設計は、ポリシー決定ポイント (PDPs) とポリシー適用ポイント (PEPs) を実装するための大まかなリファレンスとして機能し、アプリケーションのまとまりのあるユビキタスな認可モデルを形成します。

設計モデル:

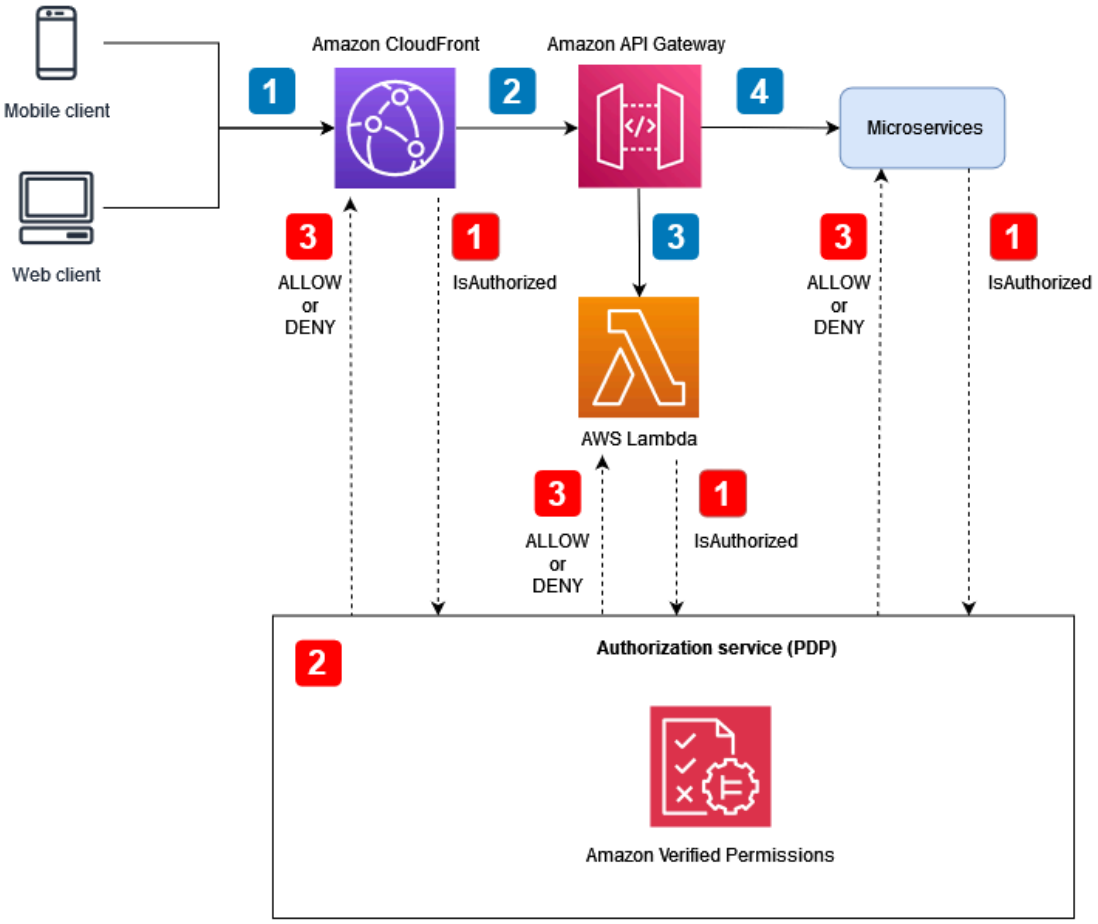
- Amazon Verified Permissions の設計モデル
- OPA の設計モデル

Amazon Verified Permissions の設計モデル

API での PEPs での一元化された PDP の使用 APIs

APIsモデルのポリシー適用ポイント (PEPs) を使用した一元化されたポリシー決定ポイント (PDP) は、業界のベストプラクティスに従って、API アクセスコントロールと認可のための効果的で簡単に維持できるシステムを作成します。このアプローチは、いくつかの主要な原則をサポートしています。

- 認可と API アクセスコントロールは、アプリケーションの複数のポイントに適用されます。
- 認可ロジックはアプリケーションから独立しています。
- アクセスコントロールの決定は一元管理されます。



アプリケーションフロー (図では青色の番号付きコールアウトで示されています) 。

1. JSON ウェブトークン (JWT) を持つ認証されたユーザーは、Amazon CloudFront への HTTP リクエストを生成します。
2. CloudFront は、CloudFront オリジンとして設定された Amazon API Gateway にリクエストを転送します。
3. API Gateway カスタムオーソライザーが呼び出され、JWT が検証されます。
4. マイクロサービスはリクエストに応答します。

認可と API アクセスコントロールフロー (図では赤の番号のコールアウトで示されています) 。

1. PEP は認可サービスを呼び出し、JWTs。
2. この場合、認可サービス (PDP) は Verified Permissions で、リクエストデータをクエリ入力として使用し、クエリで指定された関連ポリシーに基づいて評価します。
3. 認可決定は PEP に返され、評価されます。

このモデルは、一元化された PDP を使用して認可の決定を行います。PEPsは、PDP への認可リクエストを行うために、さまざまなポイントに実装されます。次の図は、架空のマルチテナント SaaS アプリケーションでこのモデルを実装する方法を示しています。

このアーキテクチャでは、PEPs Amazon CloudFront と Amazon API Gateway のサービスエンドポイント、およびマイクロサービスごとに認可決定をリクエストします。認可の決定は、認可サービスである Amazon Verified Permissions (PDP) によって行われます。Verified Permissions はフルマネージドサービスであるため、基盤となるインフラストラクチャを管理する必要はありません。Verified Permissions は、RESTful API または AWS SDK を使用して操作できます。

このアーキテクチャは、カスタムポリシーエンジンでも使用できます。ただし、Verified Permissions から得られる利点は、カスタムポリシーエンジンによって提供されるロジックに置き換える必要があります。

API 上の PEPs を使用した一元化された PDP は、APIs。 APIs これにより、認可プロセスが簡素化され、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認を決定するためのeasy-to-use繰り返し可能なインターフェイスも提供されます。

Cedar SDK の使用

Amazon Verified Permissions は Cedar 言語を使用して、カスタムアプリケーションのきめ細かなアクセス許可を管理します。Verified Permissions を使用すると、Cedar ポリシーを一元的に保存し、ミリ秒単位の処理で低レイテンシーを活用し、さまざまなアプリケーションにわたるアクセス許可を監査できます。オプションで Cedar SDK をアプリケーションに直接統合して、Verified Permissions を使用せずに認可の決定を行うこともできます。このオプションでは、ユースケースのポリシーを管理および保存するために、追加のカスタムアプリケーション開発が必要です。ただし、特に Verified Permissions へのアクセスが断続的であるか、インターネット接続に一貫性がないために不可能な場合は、実行可能な代替手段となる可能性があります。

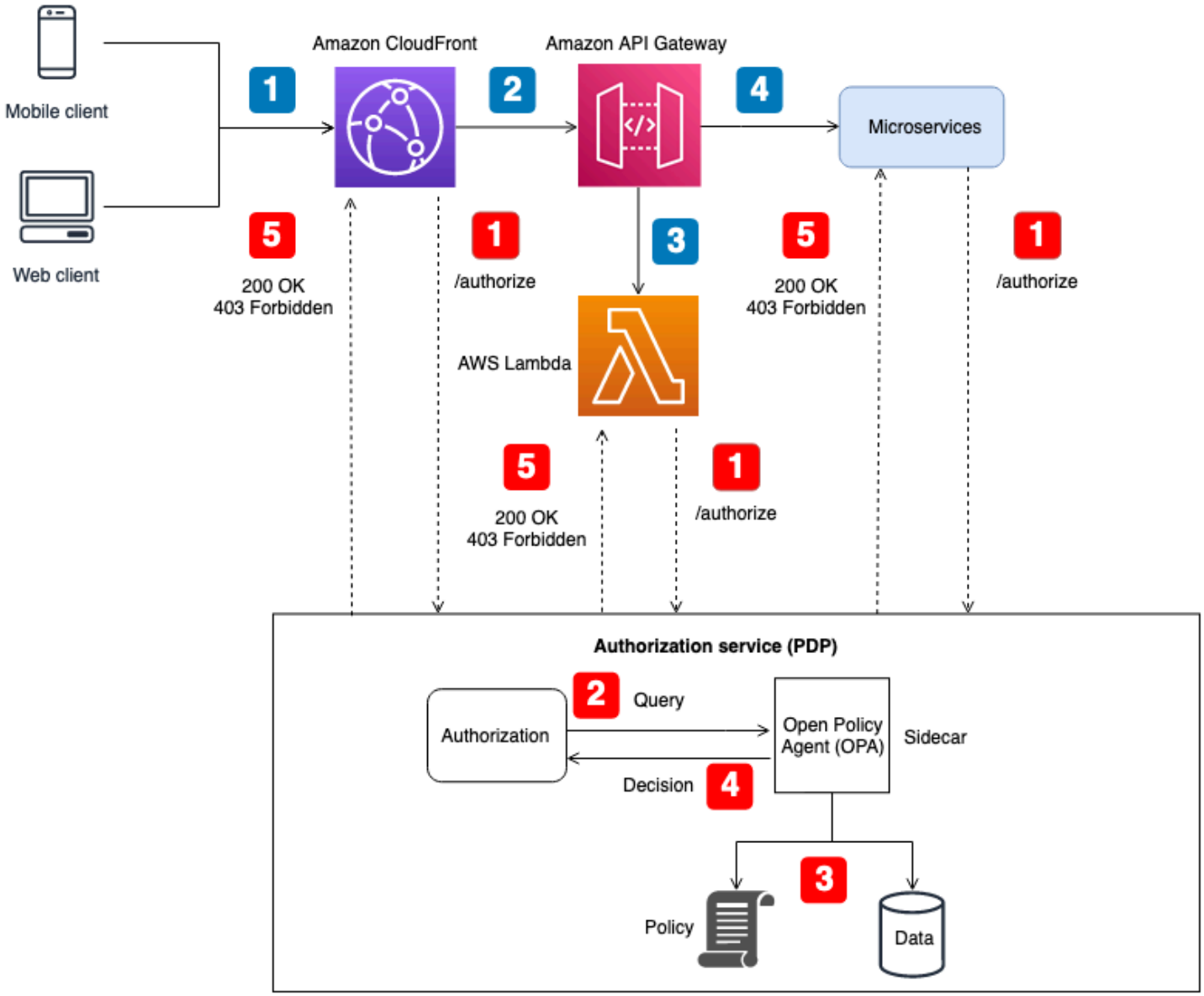
OPA の設計モデル

API での PEPs での一元化された PDP の使用 APIs

APIsモデルのポリシー適用ポイント (PEPs) を使用した一元化されたポリシー決定ポイント (PDP) は、業界のベストプラクティスに従って、API アクセスコントロールと認可のための効果的で簡単に維持できるシステムを作成します。このアプローチは、いくつかの主要な原則をサポートしています。

- 認可と API アクセスコントロールは、アプリケーションの複数のポイントに適用されます。
- 認可ロジックはアプリケーションから独立しています。
- アクセスコントロールの決定は一元管理されます。

このモデルは、一元化された PDP を使用して認可の決定を行います。PEPsは、PDP への認可リクエストを行うためにすべての APIs に実装されます。次の図は、架空のマルチテナント SaaS アプリケーションでこのモデルを実装する方法を示しています。



アプリケーションフロー (図では青色の番号付きコールアウトで示されています) 。

1. JWT を持つ認証されたユーザーは、Amazon CloudFront への HTTP リクエストを生成します。

2. CloudFront は、CloudFront オリジンとして設定された Amazon API Gateway にリクエストを転送します。
3. API Gateway カスタムオーソライザーが呼び出され、JWT が検証されます。
4. マイクロサービスはリクエストに応答します。

認可と API アクセスコントロールフロー (図では赤の番号のコールアウトで示されています) 。

1. PEP は認可サービスを呼び出し、JWTs。
2. 認可サービス (PDP) はリクエストデータを受け取り、サイドカーとして実行されている OPA エージェント REST API をクエリします。リクエストデータは、クエリへの入力として機能します。
3. OPA は、クエリで指定された関連ポリシーに基づいて入力を評価します。必要に応じて承認を決定するためにデータがインポートされます。
4. OPA は認可サービスに決定を返します。
5. 認可決定は PEP に返され、評価されます。

このアーキテクチャでは、PEPs Amazon CloudFront と Amazon API Gateway のサービスエンドポイント、およびマイクロサービスごとに認可決定をリクエストします。認可の決定は、OPA サイドカーを使用する認可サービス (PDP) によって行われます。この認可サービスは、コンテナまたは従来のサーバーインスタンスとして運用できます。OPA サイドカーは RESTful API をローカルに公開するため、API は認可サービスにのみアクセスできます。認可サービスは、PEPs。認可サービスが PEPs と OPA の間の仲介として機能することで、PEPs からの認可リクエストが OPA によって予期されるクエリ入力に準拠していない場合など、PEP と OPA の間に必要な変換ロジックを挿入できます。

このアーキテクチャは、カスタムポリシーエンジンでも使用できます。ただし、OPA から得られる利点は、カスタムポリシーエンジンによって提供されるロジックに置き換える必要があります。

API 上の PEPs を使用した一元化された PDP は、APIs。 APIs 実装が簡単で、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認決定を行うためのeasy-to-use繰り返し可能なインターフェイスも提供します。ただし、認可の決定では別の API を呼び出す必要があるため、このアプローチではアプリケーションでレイテンシーが長すぎる可能性があります。ネットワークレイテンシーに問題がある場合は、分散 PDP を検討してください。

API での PEPsでの分散 PDP の使用 APIs

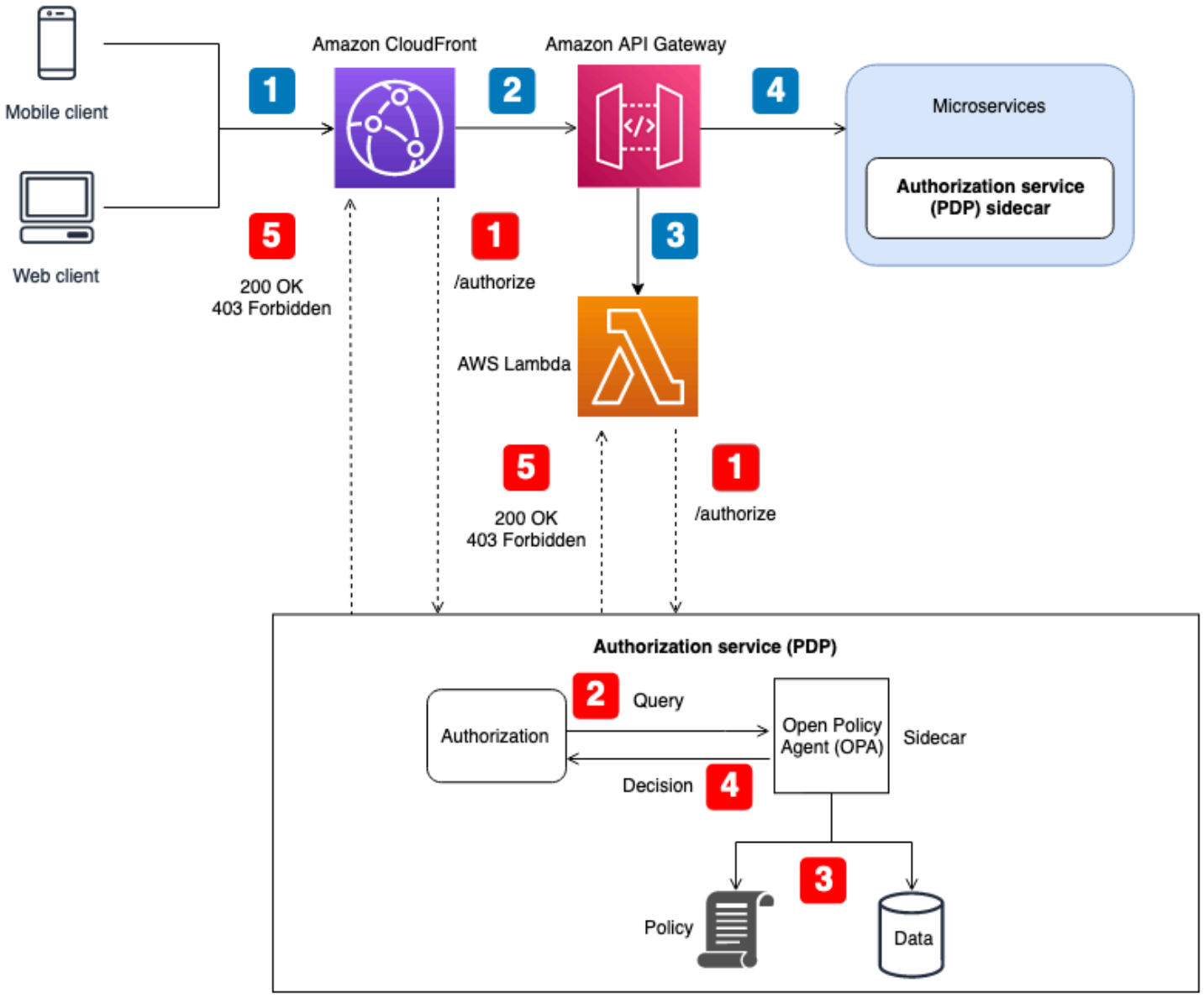
APIsモデルのポリシー適用ポイント (PEPs) を使用した分散ポリシー決定ポイント (PDP) は、業界のベストプラクティスに従って、API アクセスコントロールと認可のための効果的なシステムを作成します。API での PEPsを使用した一元化された PDP モデルと同様に、このアプローチは次の主要な原則をサポートしています。 APIs

- 認可と API アクセスコントロールは、アプリケーションの複数のポイントに適用されます。
- 認可ロジックはアプリケーションから独立しています。
- アクセスコントロールの決定は一元管理されます。

PDP が配布されるときに、アクセスコントロールの決定が一元化される理由が疑問に思うかもしれません。PDP はアプリケーションの複数の場所に存在する可能性がありますが、アクセスコントロールの決定を行うには同じ認可ロジックを使用する必要があります。すべての PDPs、入力が同じであれば、同じアクセスコントロールの決定を行います。PEPsは、PDP への認可リクエストを行うためにすべての APIs に実装されます。次の図は、この分散モデルを架空のマルチテナント SaaS アプリケーションで実装する方法を示しています。

このアプローチPDPs はアプリケーションの複数の場所に実装されます。サイドカーや Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを備えたコンテナ化されたサービスなど、OPA を実行し PDP をサポートできるオンボードコンピューティング機能を持つアプリケーションコンポーネントの場合、一元化された PDP サービスに対して RESTful API コールを行うことなく、PDP の決定をアプリケーションコンポーネントに直接統合できます。これは、すべてのアプリケーションコンポーネントが認可決定を取得するために追加の API コールを行う必要がないため、一元化された PDP モデルで発生する可能性のあるレイテンシーを減らすという利点があります。ただし、このモデルでは、Amazon CloudFront や Amazon API Gateway サービスなど、PDP の直接統合を可能にするオンボードコンピューティング機能を持たないアプリケーションコンポーネントに対して、一元化された PDP が依然として必要です。

次の図は、一元化された PDP と分散された PDP のこの組み合わせを、架空のマルチテナント SaaS アプリケーションで実装する方法を示しています。



アプリケーションフロー (図では青色の番号付きコールアウトで示されています) 。

1. JWT を持つ認証されたユーザーは、Amazon CloudFront への HTTP リクエストを生成します。
2. CloudFront は、CloudFront オリジンとして設定された Amazon API Gateway にリクエストを転送します。
3. API Gateway カスタムオーソライザーが呼び出され、JWT が検証されます。
4. マイクロサービスはリクエストに応答します。

認可と API アクセスコントロールフロー (図では赤の番号のコールアウトで示されています) 。

1. PEP は認可サービスを呼び出し、JWTs。
2. 認可サービス (PDP) はリクエストデータを受け取り、サイドカーとして実行されている OPA エージェント REST API をクエリします。リクエストデータは、クエリへの入力として機能します。
3. OPA は、クエリで指定された関連ポリシーに基づいて入力を評価します。必要に応じて承認を決定するためにデータがインポートされます。
4. OPA は認可サービスに決定を返します。
5. 認可決定は PEP に返され、評価されます。

このアーキテクチャでは、PEPs CloudFront と API Gateway のサービスエンドポイント、およびマイクロサービスごとに認可決定をリクエストします。マイクロサービスの認可決定は、アプリケーションコンポーネントでサイドカーとして動作する認可サービス (PDP) によって行われます。このモデルは、コンテナまたは Amazon Elastic Compute Cloud (Amazon EC2) インスタンスで実行されるマイクロサービス (または サービス) で使用できます。API Gateway や CloudFront などのサービスの承認決定は、外部認可サービスに連絡する必要があります。いずれにしても、認可サービスは PEPs。認可サービスが PEPs と OPA の間の仲介として機能することで、PEPs からの認可リクエストが OPA によって予期されるクエリ入力に準拠していない場合など、PEP と OPA の間に必要な変換ロジックを挿入できます。

このアーキテクチャは、カスタムポリシーエンジンでも使用できます。ただし、OPA から得られる利点は、カスタムポリシーエンジンによって提供されるロジックに置き換える必要があります。

API で PEPsを使用する分散 PDP は、APIs の堅牢な認可システムを作成するオプションを提供します。 APIs 実装が簡単で、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認決定を行うためのeasy-to-use繰り返し可能なインターフェイスを提供します。このアプローチには、一元化された PDP モデルで発生する可能性のあるレイテンシーを減らすという利点もあります。

分散 PDP をライブラリとして使用する

アプリケーション内で使用するためにライブラリまたはパッケージとして利用できる PDP に認可の決定をリクエストすることもできます。OPA は Go サードパーティーライブラリとして使用できます。他のプログラミング言語では、通常、このモデルを採用すると、カスタムポリシーエンジンを作成する必要があります。

Amazon Verified Permissions マルチテナント設計に関する考慮事項

マルチテナント SaaS ソリューションで Amazon Verified Permissions を使用して認可を実装するときに考慮すべき設計オプションがいくつかあります。これらのオプションを検討する前に、マルチテナント SaaS コンテキストでの分離と認可の違いを明確にしましょう。テナントを分離すると、インバウンドデータとアウトバウンドデータが間違ったテナントに公開されるのを防ぐことができます。認可により、ユーザーにテナントへのアクセス許可が付与されます。

Verified Permissions では、ポリシーはポリシーストアに保存されます。Verified Permissions ドキュメントで説明されているように、テナントごとに個別のポリシーストアを使用してテナントのポリシーを分離するか、すべてのテナントに単一のポリシーストアを使用してテナントにポリシーの共有を許可できます。このセクションでは、これら 2 つの分離戦略の利点と欠点について説明し、階層型デプロイモデルを使用してデプロイする方法を説明します。その他のコンテキストについては、Verified Permissions ドキュメントを参照してください。

このセクションで説明する基準は Verified Permissions に焦点を当てていますが、一般的な概念は分離の考え方とそれが提供するガイダンスに基づいています。SaaS アプリケーションは常にテナント分離を設計の一部として考慮する必要があります。この一般的な分離の原則は、SaaS アプリケーションに Verified Permissions を含めることにも及びます。このセクションでは、サイロ化された SaaS モデルやプールされた SaaS モデルなどのコア SaaS 分離モデルも参照します。 SaaS 詳細については、 AWS 「 Well-Architected フレームワーク、SaaS レンズ」の「コア分離の概念」を参照してください。

マルチテナント SaaS ソリューションを設計する際の主な考慮事項は、テナント分離とテナントオンボーディングです。テナントの分離は、セキュリティ、プライバシー、耐障害性、パフォーマンスに影響します。テナントオンボーディングは、運用オーバーヘッドとオブザーバビリティに関連する運用プロセスに影響します。SaaS ジャーニーを実行したり、マルチテナントソリューションを実装したりする組織は、SaaS アプリケーションによるテナンシーの処理方法を常に優先する必要があります。SaaS ソリューションは特定の分離モデルに傾いている可能性がありますが、SaaS ソリューション全体で必ずしも一貫性が必要なわけではありません。たとえば、アプリケーションのフロントエンドコンポーネント用に選択した 分離モデルは、マイクロサービスまたは認可サービス用に選択した分離モデルとは異なる場合があります。

設計上の考慮事項:

- テナントオンボーディングとユーザーテナント登録

- テナントごとのポリシーストア
- 1 つの共有マルチテナントポリシーストア
- 階層型デプロイモデル

テナントオンボーディングとユーザーテナント登録

SaaS アプリケーションは SaaS アイデンティティの概念に従い、ユーザーアイデンティティをテナントアイデンティティにバインドするという一般的なベストプラクティスに従います。バインドでは、テナント識別子を ID プロバイダーのユーザーのクレームまたは属性として保存します。これにより、ID をテナントにマッピングする責任が各アプリケーションからユーザー登録プロセスに移行されます。その後、認証された各ユーザーは、JSON ウェブトークン (JWT) の一部として正しいテナント ID を持ちます。

同様に、認可リクエストの正しいポリシーストアの選択は、アプリケーションロジックによって決定しないでください。特定の認可リクエストで使用するポリシーストアを決定するには、ユーザーのポリシーストアへのマッピング、またはテナントのポリシーストアへのマッピングを維持します。これらのマッピングは通常、アプリケーションが参照する Amazon DynamoDB や Amazon Relational Database Service (Amazon RDS) などのデータストアに保持されます。これらのマッピングを ID プロバイダー (IdP) のデータで提供または補足することもできます。テナント、ユーザー、ポリシーストア間の関係は通常、認可リクエストに必要なすべての関係を含む JWT を通じてユーザーに提供されます。

この例では、テナントに属`TenantA`し`Alice`、承認`ps-43214321`のためにポリシーストア ID を持つポリシーストアを使用するユーザー に対して JWT がどのように表示されるかを示します。

```
{
   "sub":"1234567890",
   "name":"Alice",
   "tenant":"TenantA",
   "policyStoreId":"ps-43214321"
}
```

テナントごとのポリシーストア

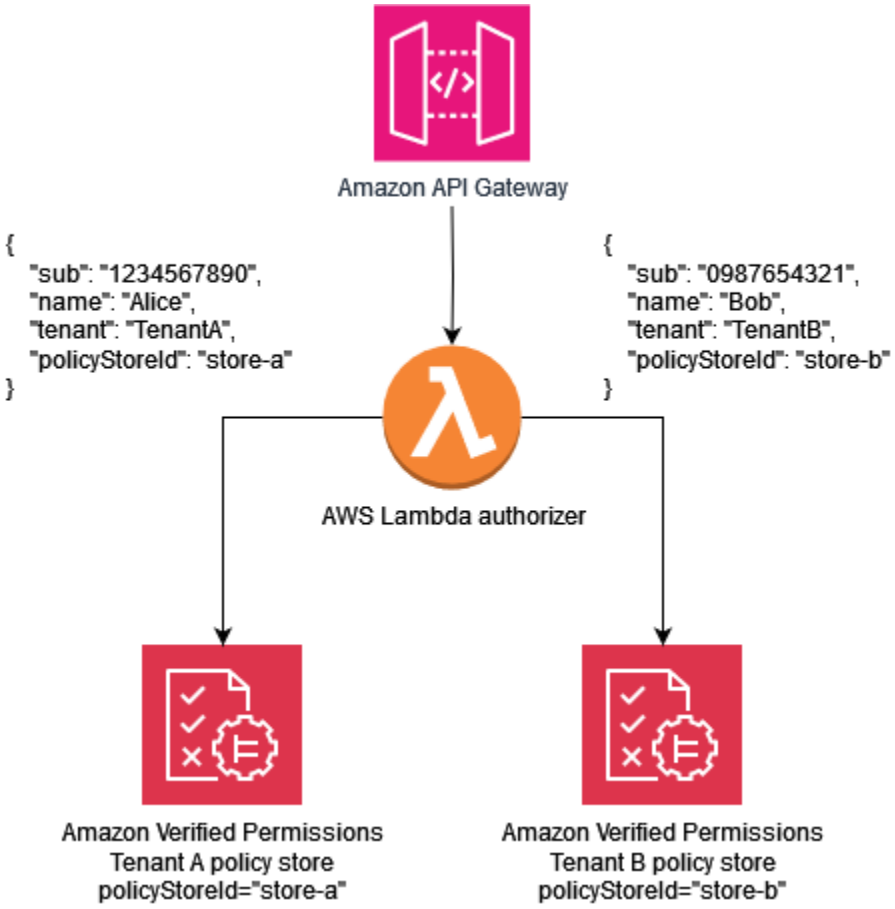
Amazon Verified Permissions のテナントごとのポリシーストア設計モデルは、SaaS アプリケーションの各テナントを独自のポリシーストアに関連付けます。このモデルは、SaaS サイロ分離モデルと似ています。どちらのモデルもテナント固有のインフラストラクチャの作成を義務付けており、

同様の利点と欠点があります。このアプローチの主な利点は、インフラストラクチャが強制するテナントの分離、テナントごとに固有の認可モデルのサポート、ノイズの多い近隣の懸念の排除、ポリシーの更新やデプロイにおける障害の影響範囲の縮小です。このアプローチの欠点には、より複雑なテナントオンボーディングプロセス、デプロイ、運用などがあります。テナントごとのポリシーストアは、ソリューションにテナントごとに一意のポリシーがある場合に推奨されるアプローチです。

SaaS アプリケーションが必要とする場合、テナントごとのポリシーストアモデルは、テナント分離に対して高度にサイロ化されたアプローチを提供できます。プール分離でこのモデルを使用することもできますが、Verified Permissions の実装では、管理やオペレーションの簡素化など、より広範なプール分離モデルの標準的な利点は共有されません。

テナントごとのポリシーストアでは、前述のように、テナントのポリシーストア識別子をユーザー登録プロセス中にユーザーの SaaS ID にマッピングすることで、テナントの分離が達成されます。このアプローチは、テナントのポリシーストアをユーザープリンシパルに強く結び付け、SaaS ソリューション 全体でマッピングを共有する一貫した方法を提供します。マッピングを IdP の一部として、または DynamoDB などの外部データソースに維持することで、SaaS アプリケーションへのマッピングを提供できます。これにより、プリンシパルがテナントの一部であること、およびテナントのポリシーストアが使用されることも保証されます。

この例では、`tenant`ユーザーの `policyStoreId`と を含む JWT が API エンドポイントから AWS Lambda オーソライザーのポリシー評価ポイントに渡され、リクエストが正しいポリシーストアにルーティングされる方法を示します。



次のサンプルポリシーは、テナントごとのポリシーストア設計パラダイムを示しています。ユーザーは にAlice属しています。policyStoreId TenantA.store-aは のテナント ID にもマッピングAlice,され、正しいポリシーストアの使用を強制します。これにより、 のポリシーTenantAが使用されます。

> **Note**
>
> テナントごとのポリシーストアモデルは、テナントのポリシーを分離します。認可は、ユーザーがデータに対して実行できるアクションを適用します。このモデルを使用する架空のアプリケーションに関連するリソースは、AWS Well-Architected Framework、SaaS レンズドキュメントで定義されているように、他の分離メカニズムを使用して分離する必要があります。

このポリシーでは、 Aliceにはすべてのリソースのデータを表示するアクセス許可があります。

```
permit (
    principal == MultiTenantApp::User::"Alice",
```

```
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

認可リクエストを行い、Verified Permissions ポリシーを使用して評価を開始するには、テナントにマッピングされた一意の ID に対応するポリシーストア ID を指定する必要がありますstore-a。

```
{
   "policyStoreId":"store-a",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Alice"
               },
               "attributes":{},
               "parents":[]
            },
            {
               "identifier":{
                  "entityType":"MultiTenantApp::Data",
                  "entityId":"my_example_data"
               },
               "attributes":{},
               "parents":[]
            }
         ]
      ]
   }
```

```
}
```

ユーザーはテナント B にBob属し、policyStoreId `store-b`は のテナント ID にもマッピングされるためBob、正しいポリシーストアが使用されます。これにより、テナント B のポリシーが使用されます。

このポリシーでは、 Bobにはすべてのリソースのデータをカスタマイズするアクセス許可があります。この例では、 はテナント B にのみ固有の`customizeData`アクションである可能性があるため、ポリシーはテナント B に対して一意になります。テナントごとのポリシーストアモデルは、テナントごとに本質的にカスタムポリシーをサポートします。

```
permit (
    principal == MultiTenantApp::User::"Bob",
    action == MultiTenantApp::Action::"customizeData",
    resource
);
```

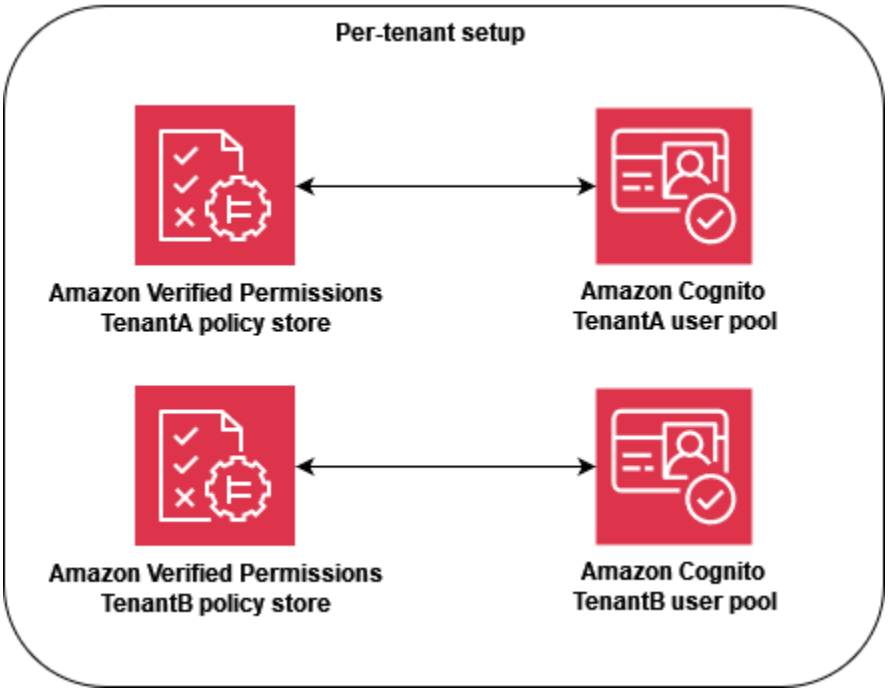
認可リクエストを行い、Verified Permissions ポリシーを使用して評価を開始するには、テナントにマッピングされた一意の ID に対応するポリシーストア ID を指定する必要があります`store-b`。

```
{
   "policyStoreId":"store-b",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Bob"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"customizeData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Bob"
```

```
                },
                "attributes":{},
                "parents":[]
            },
            {
                "identifier":{
                    "entityType":"MultiTenantApp::Data",
                    "entityId":"my_example_data"
                },
                "attributes":{},
                "parents":[]
            }
        ]
    ]
  }
}
```

Verified Permissions では、IdP をポリシーストアと統合することは可能ですが、必須ではありません。この統合により、ポリシーは ID ストアのプリンシパルをポリシーのプリンシパルとして明示的に参照できます。Verified Permissions の IdP として Amazon Cognito と統合する方法の詳細については、Verified Permissions ドキュメントと Amazon Cognito ドキュメントを参照してください。

ポリシーストアを IdP と統合する場合、ポリシーストアごとに 1 つの ID ソースのみを使用できます。例えば、Verified Permissions を Amazon Cognito と統合する場合は、Verified Permissions ポリシーストアと Amazon Cognito ユーザープールのテナント分離に使用される戦略をミラーリングする必要があります。ポリシーストアとユーザープールも同じ に存在する必要があります AWS アカウント。



運用レベルでは、テナントごとのポリシーストアには監査上の利点があります。これは、記録されたアクティビティをテナントごとに でAWS CloudTrail 個別に簡単にクエリできるためです。ただし、テナントごとのディメンションに追加のカスタムメトリクスを Amazon CloudWatch に記録することをお勧めします。

テナントごとのポリシーストアアプローチでは、2 つの Verified Permissions クォータに細心の注意を払って、SaaS ソリューションのオペレーションに干渉しないようにする必要があります。これらのクォータは、アカウントごとのリージョンごとのポリシーストアと、アカウントごとのリージョンごとの 1 秒あたりの IsAuthorized リクエストです。両方のクォータの引き上げをリクエストできます。

テナントごとのポリシーストアモデルを実装する方法の詳細な例については、 AWS ブログ記事「Amazon Verified Permissions with atenant policy store SaaS」を参照してください。

1 つの共有マルチテナントポリシーストア

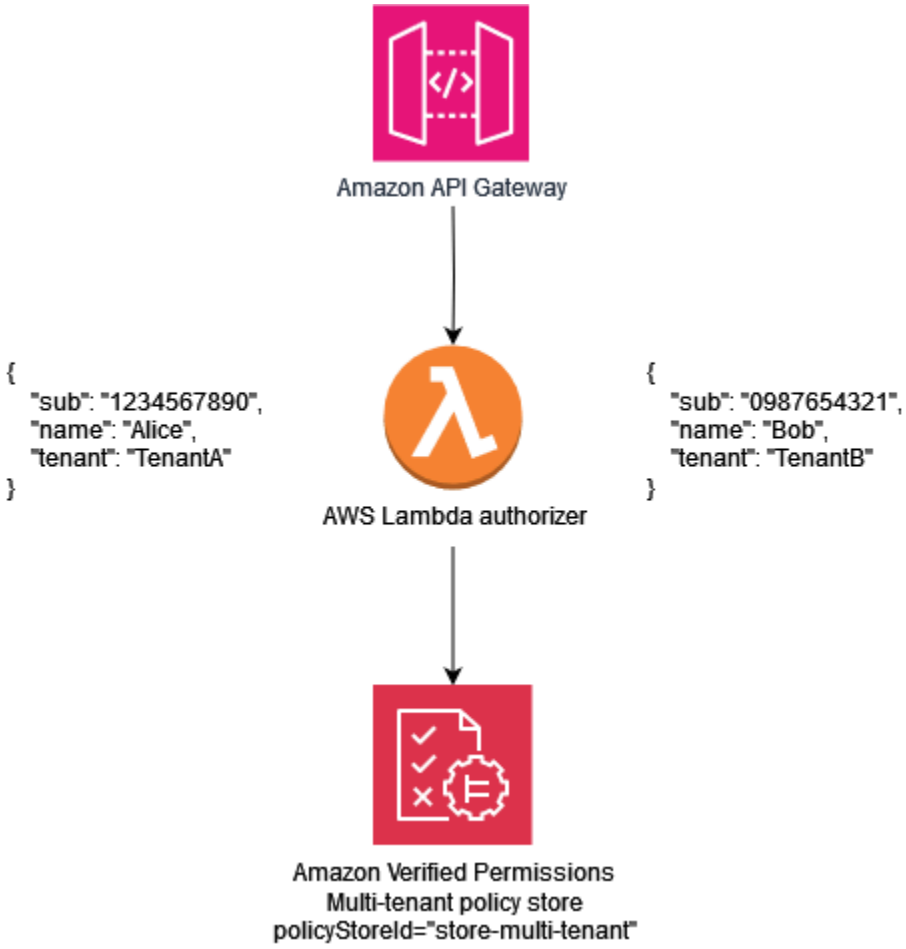
1 つの共有マルチテナントポリシーストア設計モデルは、SaaS ソリューション内のすべてのテナントに対して Amazon Verified Permissions の 1 つのマルチテナントポリシーストアを使用します。このアプローチの主な利点は、管理と運用の簡素化です。特に、テナントのオンボーディング中に追加のポリシーストアを作成する必要がないためです。このアプローチの欠点は、ポリシーの更新やデプロイの失敗やミスによる影響範囲の拡大、ノイズの多い近隣への影響の増大です。さらに、ソリューションがテナントごとに一意のポリシーを必要とする場合、このアプローチはお勧めしません。この

場合、正しいテナントのポリシーが使用されるように、代わりにテナントごとのポリシーストアモデルを使用します。

1 つの共有マルチテナントポリシーストアアプローチは、SaaS プール分離モデルに似ています。SaaS アプリケーションが必要とする場合、テナント分離にプールアプローチを提供できます。SaaS ソリューションがサイロ化された分離をマイクロサービスに適用する場合は、このモデルを使用することもできます。モデルを選択するときは、テナントデータ分離の要件と、SaaS アプリケーションに必要な Verified Permissions ポリシーの構造を個別に評価する必要があります。

SaaS ソリューション全体でテナント識別子を共有する一貫した方法を適用するには、前述のように、ユーザー登録中に識別子をユーザーの SaaS ID にマッピングすることをお勧めします。このマッピングは、IdP の一部として、または DynamoDB などの外部データソースに維持することで、SaaS アプリケーションに提供できます。また、共有ポリシーストア ID をユーザーにマッピングすることをお勧めします。ID はテナント分離の一部としては使用されませんが、将来の変更が容易になるため、これは良い方法です。

次の例は、API エンドポイントが、異なるテナントに属しているがBob、承認`store-multi-tenant`のためにポリシーストア ID とポリシーストアを共有するユーザー`Alice`と に JWT を送信する方法を示しています。すべてのテナントは単一のポリシーストアを共有するため、ポリシーストア ID をトークンまたはデータベースに保持する必要はありません。すべてのテナントは単一のポリシーストア ID を共有するため、アプリケーションがポリシーストアを呼び出すために使用できる環境変数として ID を指定できます。



次のサンプルポリシーは、1 つの共有マルチテナントポリシー設計パラダイムを示しています。このポリシーでは、親MultiTenantApp::Userを持つプリンシパルには、すべてのリソースのデータを表示するアクセス許可MultiTenantApp::RoleAdminがあります。

```
permit (
    principal in MultiTenantApp::Role::"Admin",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

単一のポリシーストアが使用されているため、Verified Permissions ポリシーストアは、プリンシパルに関連付けられているテナンシー属性がリソースに関連付けられているテナンシー属性と一致することを確認する必要があります。これは、次のポリシーをポリシーストアに含めて、リソースとプリンシパルに一致するテナンシー属性を持たないすべての認可リクエストが拒否されるようにすることで実現できます。

```
forbid(
```

```
    principal,
    action,
    resource
)
unless {
    resource.Tenant == principal.Tenant
};
```

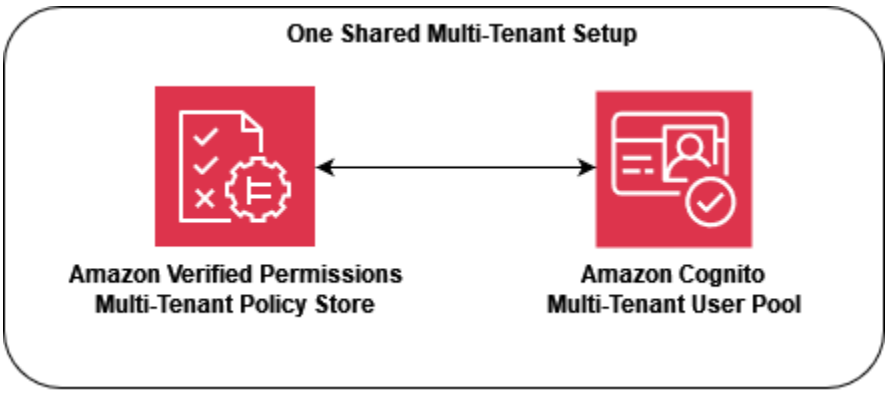
1 つの共有マルチテナントポリシーストアモデルを使用する認可リクエストの場合、ポリシーストア ID は共有ポリシーストアの識別子です。次のリクエストでは、 UserAliceは Roleの を持ちAdmin、リソースとプリンシパルに関連付けられたTenant属性は両方とも であるため、 へのアクセスが許可されますTenantA。

```
{
   "policyStoreId":"store-multi-tenant",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         {
            "identifier":{
               "entityType":"MultiTenantApp::User",
               "entityId":"Alice"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
                }
```

```
            },
            "parents":[
               {
                  "entityType":"MultiTenantApp::Role",
                  "entityId":"Admin"
               }
            ]
         },
         {
            "identifier":{
               "entityType":"MultiTenantApp::Data",
               "entityId":"my_example_data"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
                }
            },
            "parents":[]
         }
      ]
   }
}
```

Verified Permissions では、IdP をポリシーストアと統合することは可能ですが、必須ではありません。この統合により、ポリシーは ID ストア内のプリンシパルをポリシーのプリンシパルとして明示的に参照できます。Verified Permissions の IdP として Amazon Cognito と統合する方法の詳細については、Verified Permissions ドキュメントと Amazon Cognito ドキュメントを参照してください。

ポリシーストアを IdP と統合する場合、ポリシーストアごとに 1 つの ID ソースのみを使用できます。例えば、Verified Permissions を Amazon Cognito と統合する場合は、Verified Permissions ポリシーストアと Amazon Cognito ユーザープールのテナント分離に使用される戦略をミラーリングする必要があります。ポリシーストアとユーザープールも同じ に存在する必要があります AWS アカウント。



運用上および監査上の観点から、1 つの共有マルチテナントポリシーストアモデルには欠点があります。これは、ログに記録された AWS CloudTrail各 CloudTrail 呼び出しが同じポリシーストアを使用するため、 のログに記録されたアクティビティがテナントの個々のアクティビティを除外するために、より関連性の高いクエリを必要とするという点です。このシナリオでは、テナントごとのディメンションに関する追加のカスタムメトリクスを Amazon CloudWatch に記録して、適切なレベルのオブザーバビリティと監査機能を確保すると便利です。

1 つの共有マルチテナントポリシーストアアプローチでは、SaaS ソリューションのオペレーションに干渉しないように、Verified Permissions クォータにも細心の注意が必要です。特に、アカウントクォータごとにリージョンごとに 1 秒あたりの IsAuthorized リクエストをモニタリングして、制限を超えないようにすることをお勧めします。このクォータの引き上げをリクエストできます。

階層型デプロイモデル

階層型デプロイモデルを作成することで、優先度の高い「エンタープライズ階層」テナントを、潜在的に大量の「スタンダード階層」顧客から分離できます。このモデルでは、ポリシーストアのポリシーにデプロイされた変更を階層ごとに個別にロールアウトできます。これにより、各階層の顧客を階層外の変更から分離できます。階層型デプロイモデルでは、通常、ポリシーストアはテナントのオンボーディング時にデプロイされるのではなく、各階層の初期インフラストラクチャプロビジョニングの一部として作成されます。

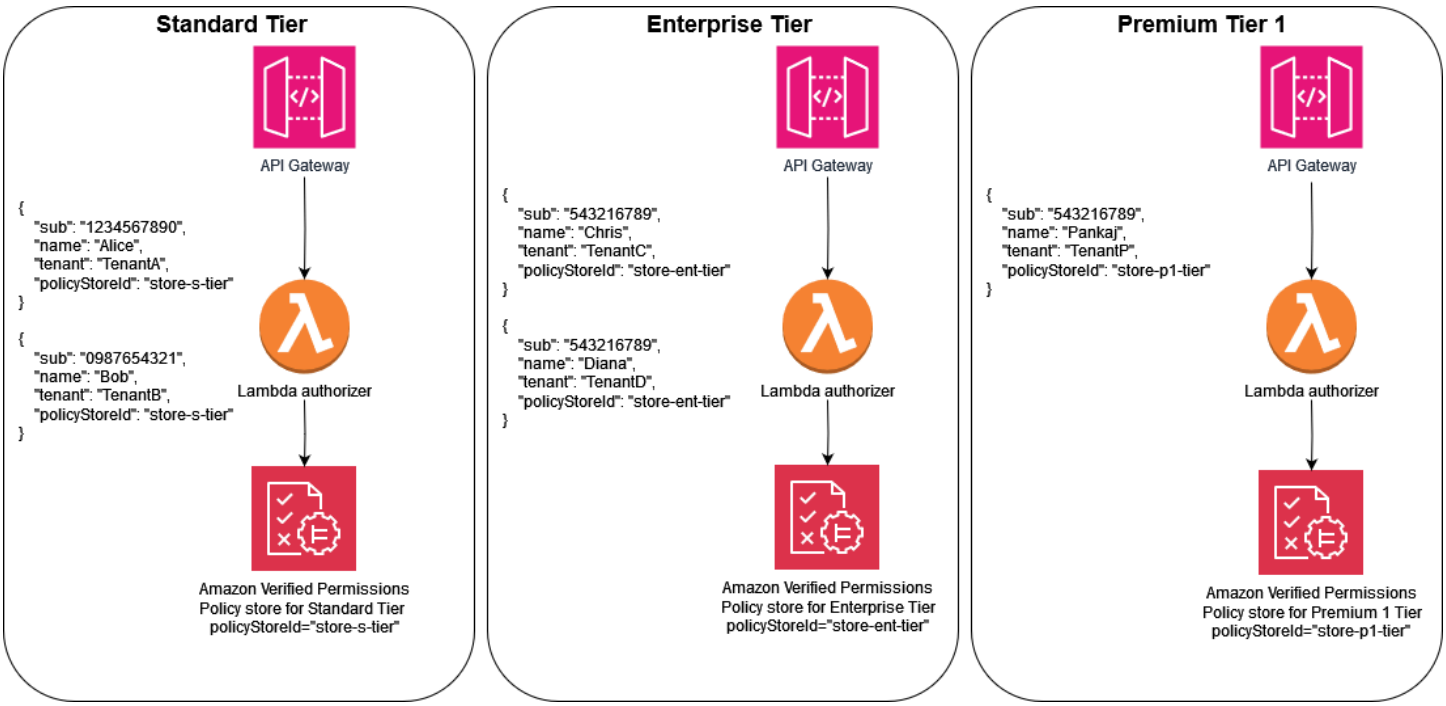
ソリューションがプールされた分離モデルを主に使用する場合は、追加の分離またはカスタマイズが必要になる場合があります。たとえば、各テナントが独自のテナント階層インフラストラクチャを取得する「プレミアム階層」を作成できます。これにより、テナントが 1 つしかないプールされたインスタンスをデプロイすることでサイロ化されたモデルが作成されます。これは、ポリシーストアを含む完全に分離された「プレミアム層テナント A」インフラストラクチャと「プレミアム層テナント B」インフラストラクチャの形式になる可能性があります。このアプローチにより、最高レベルの顧客に対してサイロ化された分離モデルが作成されます。

階層型デプロイモデルでは、各ポリシーストアは個別にデプロイされますが、同じ分離モデルに従う必要があります。複数のポリシーストアが使用されているため、SaaS ソリューション全体でテナントに関連付けられているポリシーストア識別子を一貫した方法で共有する必要があります。テナントごとのポリシーストアモデルと同様に、ユーザー登録時にテナント識別子をユーザーの SaaS ID にマッピングすることをお勧めします。

次の図は、Standard Tier、、 Enterprise Tierの 3 つの階層を示していますPremium Tier 1。各階層は独自のインフラストラクチャに個別にデプロイされ、階層内に 1 つの共有ポリシーストアを使用します。標準階層とエンタープライズ階層には複数のテナントが含まれています。TenantAと TenantBは にありStandard Tier、 TenantCと TenantDはエンタープライズ階層にあります。

Premium Tier 1 には のみが含まれているためTenantP、ソリューションが完全にサイロ化された分離モデルであるかのようにプレミアムテナントを提供し、カスタマイズされたポリシーなどの機能を提供できます。新しいプレミアム階層の顧客をオンボーディングすると、Premium Tier 2インフラストラクチャが作成されます。

> **Note**
>
> プレミアム階層のアプリケーション、デプロイ、テナントオンボーディングは、標準階層とエンタープライズ階層と同じです。唯一の違いは、プレミアム階層のオンボーディングワークフローが新しい階層インフラストラクチャのプロビジョニングから始まることです。


Standard Tier
API Gateway
{
"sub": "1234567890",
"name": "Alice",
"tenant": "TenantA",
"policyStoreId": "store-s-tier"
}
{
"sub": "0987654321",
"name": "Bob",
"tenant": "TenantB",
"policyStoreId": "store-s-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Standard Tier
policyStoreId="store-s-tier"
Enterprise Tier
API Gateway
{
"sub": "543216789",
"name": "Chris",
"tenant": "TenantC",
"policyStoreId": "store-ent-tier"
}
{
"sub": "543216789",
"name": "Diana",
"tenant": "TenantD",
"policyStoreId": "store-ent-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Enterprise Tier
policyStoreId="store-ent-tier"
Premium Tier 1
API Gateway
{
"sub": "543216789",
"name": "Pankaj",
"tenant": "TenantP",
"policyStoreId": "store-p1-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Premium 1 Tier
policyStoreId="store-p1-tier"

OPA マルチテナント設計に関する考慮事項

Open Policy Agent (OPA) は、ポリシーと認可の決定を行うためにアプリケーションが必要な多数のユースケースに適用できる柔軟なサービスです。マルチテナント SaaS アプリケーションで OPA を使用するには、テナント分離などの主要な SaaS ベストプラクティスが OPA の実装の一部であり続けるように、固有の基準を考慮する必要があります。これらの基準には、OPA デプロイパターン、テナント分離と OPA ドキュメントモデル、テナントオンボーディングが含まれます。これらはそれぞれ、マルチテナントアプリケーションに関連する OPA の最適な設計に影響します。

このセクションのディスカッションでは OPA に焦点を当てていますが、一般的な概念は分離の考え方とそれが提供するガイダンスに基づいています。SaaS アプリケーションは常にテナント分離を設計の一部として考慮する必要があります。この分離の一般的な原則は、SaaS アプリケーションに OPA を含めることにも及びます。OPA を適切に使用すると、SaaS アプリケーションで分離がどのように適用されるかの重要な部分になります。このセクションでは、サイロ化された SaaS モデルやプールされた SaaS モデルなどのコア SaaS 分離モデルも参照します。 SaaS 詳細については、AWS 「 Well-Architected フレームワーク、SaaS レンズ」の「コア分離の概念」を参照してください。

設計上の考慮事項:

- 集中型デプロイパターンと分散型デプロイパターンの比較
- OPA ドキュメントモデルによるテナント分離
- テナントオンボーディング

集中型デプロイパターンと分散型デプロイパターンの比較

OPA は集中型または分散型のデプロイパターンでデプロイでき、マルチテナントアプリケーションに最適な方法はユースケースによって異なります。これらのパターンの例については、このガイドの前半のPEPs APIs」およびAPIs での分散された PDP と PEPs」セクションを参照してください。OPA はオペレーティングシステムまたはコンテナにデーモンとしてデプロイできるため、マルチテナントアプリケーションをサポートする複数の方法で実装できます。

一元化されたデプロイパターンでは、OPA は RESTful API を使用してコンテナまたはデーモンとしてデプロイされ、アプリケーション内の他のサービスで使用できます。サービスが OPA からの決定を必要とする場合、中央 OPA RESTful API が呼び出され、この決定が生成されます。このアプローチは、OPA のデプロイが 1 つしかないため、デプロイと保守が簡単になります。このアプローチの

欠点は、テナントデータの分離を維持するメカニズムが提供されていないことです。OPA のデプロイは 1 つのみであるため、OPA によって参照される外部データを含め、OPA の決定で使用されるすべてのテナントデータが OPA で利用できる必要があります。このアプローチでテナントデータの分離を維持できますが、OPA のポリシーとドキュメント構造、または外部データへのアクセスによって強制される必要があります。一元化されたデプロイパターンでは、各認可決定が別のサービスに対して RESTful API コールを行う必要があるため、レイテンシーも長くする必要があります。

分散デプロイパターンでは、OPA はマルチテナントアプリケーションのサービスと共にコンテナまたはデーモンとしてデプロイされます。サイドカーコンテナとしてデプロイすることも、オペレーティングシステムでローカルに実行されるデーモンとしてデプロイすることもできます。OPA から決定を取得するために、サービスはローカル OPA デプロイに対して RESTful API コールを実行します。(OPA は Go パッケージとしてデプロイできるため、RESTful API コールを使用する代わりに Go をネイティブに使用して決定を取得できます)。一元化されたデプロイパターンとは異なり、分散パターンはアプリケーションの複数の領域に存在するため、デプロイ、保守、更新にははるかに堅牢な労力が必要です。分散デプロイパターンの利点は、特にサイロ化された SaaS モデルを使用するアプリケーションで、テナントデータの分離を維持する機能です。分散モデルの OPA はテナントと一緒にデプロイされるため、テナント固有のデータは、そのテナントに固有の OPA デプロイで分離できます。さらに、各認可決定はローカルで実行できるため、分散デプロイパターンのレイテンシーは一元化されたデプロイパターンよりもはるかに低くなります。

マルチテナントアプリケーションで OPA デプロイパターンを選択するときは、アプリケーションにとって最も重要な基準を必ず評価してください。マルチテナントアプリケーションがレイテンシーの影響を受けやすい場合、分散デプロイパターンは、より複雑なデプロイとメンテナンスを犠牲にして、パフォーマンスを向上させます。この複雑さの一部は DevOps と自動化を通じて管理できますが、一元化されたデプロイパターンと比較して、追加の労力が必要です。

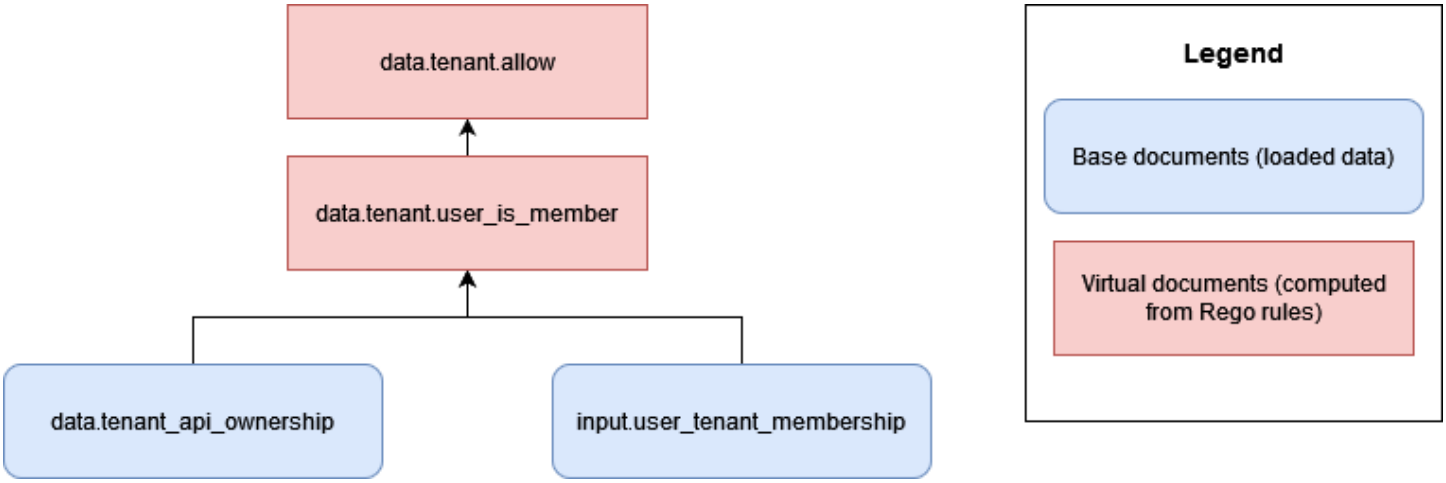
マルチテナントアプリケーションがサイロ化された SaaS モデルを使用している場合は、分散 OPA デプロイパターンを使用して、テナントデータ分離に対するサイロ化されたアプローチを模倣できます。これは、各テナント固有のアプリケーションサービスと一緒に OPA を実行する場合、各 OPA デプロイをカスタマイズして、そのテナントに関連付けられているデータのみを含めることができるためです。一元化された OPA デプロイパターンで OPA データをサイロ化することはできません。一元化されたデプロイパターンまたは分散パターンをプールされた SaaS モデルと組み合わせて使用する場合、テナントデータの分離は OPA ドキュメントモデルで維持する必要があります。

OPA ドキュメントモデルによるテナント分離

OPA はドキュメントを使用して意思決定を行います。これらのドキュメントにはテナント固有のデータを含めることができるため、テナントデータの分離を維持する方法を検討する必要がありま

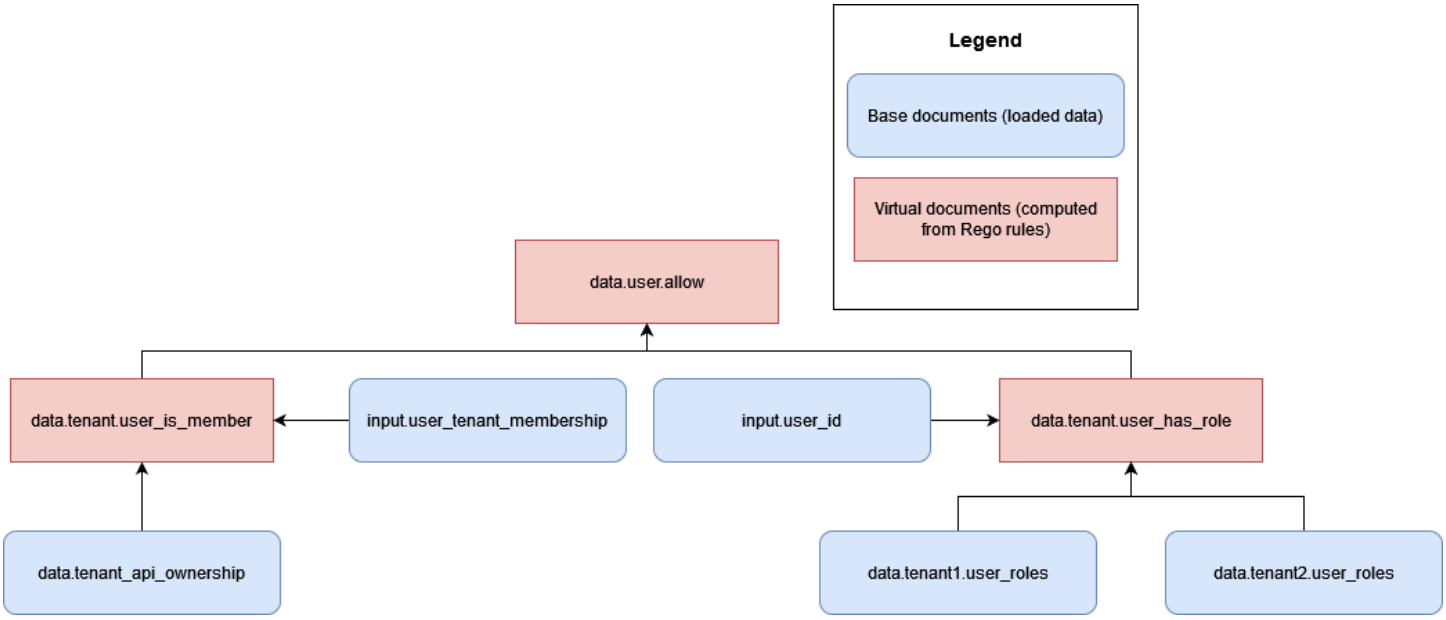
す。OPA ドキュメントは、ベースドキュメントと仮想ドキュメントで構成されます。ベースドキュメントには、外部からのデータが含まれています。これには、OPA に直接提供されるデータ、OPA リクエストに関するデータ、および入力として OPA に渡される可能性のあるデータが含まれます。仮想ドキュメントはポリシーによって計算され、OPA ポリシーとルールが含まれます。詳細については、OPA ドキュメントを参照してください。

マルチテナントアプリケーション用に OPA でドキュメントモデルを設計するには、まず OPA で決定する必要があるベースドキュメントのタイプを考慮する必要があります。これらのベースドキュメントにテナント固有のデータが含まれている場合は、このデータが誤ってクロステナントアクセスにさらされないように対策を講じる必要があります。幸い、多くの場合、OPA で決定を下すためにテナント固有のデータは必要ありません。次の例は、入力ドキュメントに示されているように、API を所有するテナントと、ユーザーがテナントのメンバーであるかどうかに基づいて API へのアクセスを許可する架空の OPA ドキュメントモデルを示しています。



このアプローチでは、OPA は、API を所有するテナントに関する情報を除き、テナント固有のデータにアクセスできません。この場合、OPA がアクセス決定を行うために使用する唯一の情報は、ユーザーのテナントとの関連付けとテナントの APIs との関連付けであるため、OPA によるクロステナントアクセスの促進に懸念はありません。各テナントは独立したリソースを所有するため、このタイプの OPA ドキュメントモデルをサイロ化された SaaS モデルに適用できます。

ただし、多くの RBAC 認可アプローチでは、クロステナントの情報が公開される可能性があります。次の例では、架空の OPA ドキュメントモデルにより、ユーザーがテナントのメンバーであるかどうか、およびユーザーが API にアクセスするための正しいロールを持っているかどうかに基づいて、API へのアクセスが許可されます。



このモデルでは、クロステナントアクセスのリスクが生じます。これは、 および で複数のテナントのロールdata.tenant1.user_rolesとアクセス許可を OPA がアクセスできるようにして、認可の決定を行うdata.tenant2.user_roles必要があるためです。テナントの分離とロールマッピングのプライバシーを維持するために、このデータは OPA 内に存在しないでください。RBAC データは、データベースなどの外部データソースに存在する必要があります。さらに、事前定義されたロールを特定のアクセス許可にマッピングするために OPA を使用しないでください。これにより、テナントが独自のロールとアクセス許可を定義することが難しくなります。また、認可ロジックがリジッドになり、継続的な更新が必要になります。RBAC データを OPA 意思決定プロセスに安全に組み込む方法のガイダンスについては、このガイドの後半にある「テナントの分離とデータプライバシーに関する推奨事項」セクションを参照してください。

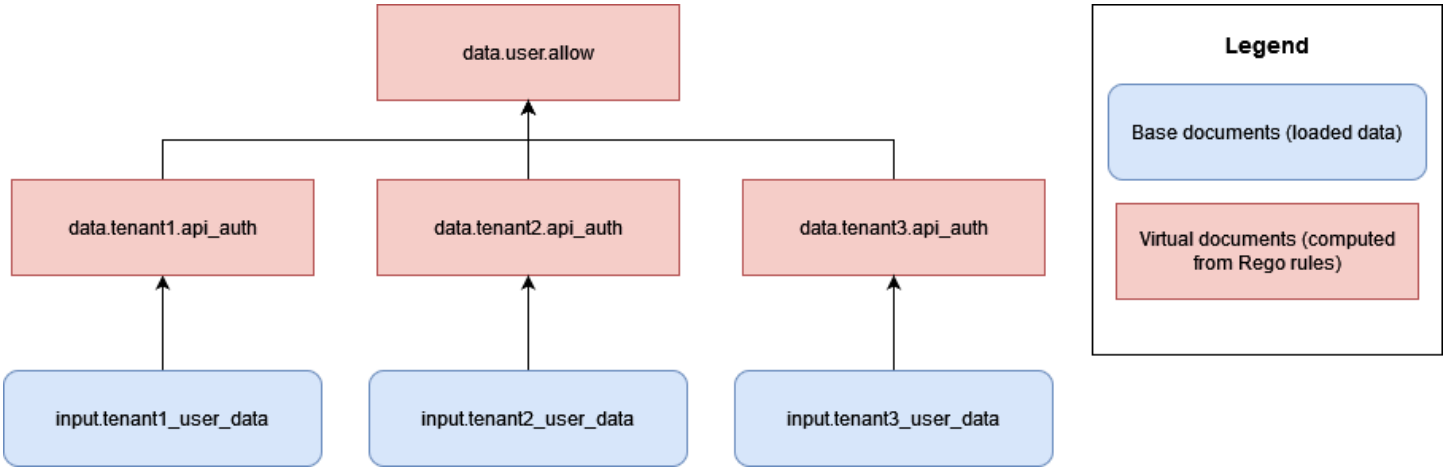
テナント固有のデータを非同期ベースドキュメントとして保存しないことで、OPA でテナント分離を簡単に維持できます。非同期ベースドキュメントは、メモリに保存され、OPA で定期的に更新できるデータです。OPA 入力などの他の基本ドキュメントは同期的に渡され、決定時にのみ使用できます。例えば、クエリへの OPA 入力の一部としてテナント固有のデータを提供することは、テナント分離の違反にはなりません。これは、そのデータが決定プロセス中に同期的にしか利用できないためです。

テナントオンボーディング

OPA ドキュメントの構造では、面倒な要件を導入することなく、テナントのオンボーディングを簡素化できる必要があります。OPA ドキュメントモデル階層で仮想ドキュメントをパッケージで整理でき、これらのパッケージには多くのルールを含めることができます。マルチテナントアプリケー

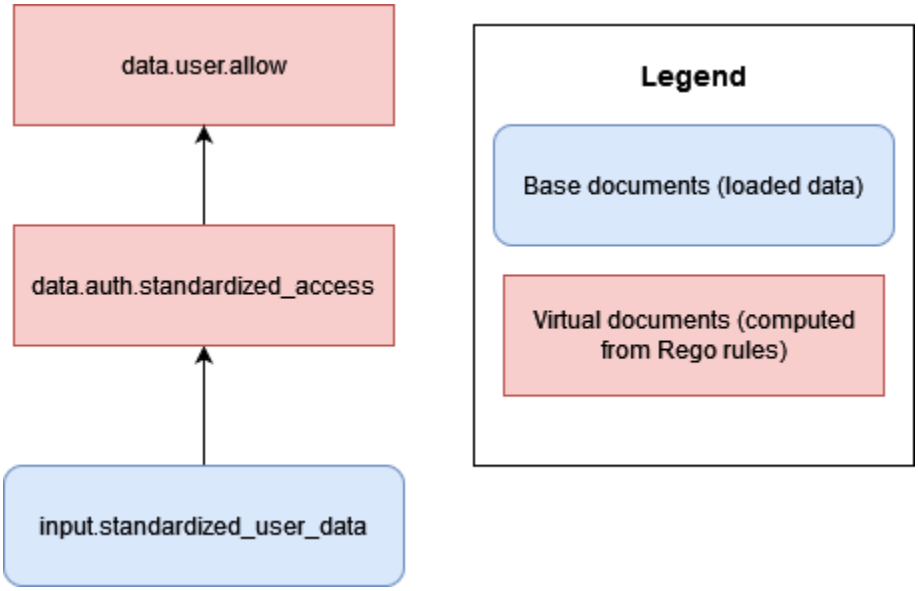
ションの OPA ドキュメントモデルを計画するときは、まず OPA が決定を下すために必要なデータを決定します。データを入力として提供したり、OPA に事前ロードしたり、外部データソースから決定時または定期的に提供したりできます。OPA で外部データを使用する方法の詳細については、このガイドの後半の「OPA での PDP の外部データの取得」セクションを参照してください。

OPA で決定するために必要なデータを決定したら、パッケージとして整理された OPA ルールを実装する方法を検討し、そのデータで決定を行います。たとえば、各テナントに認可の決定方法に関する固有の要件があるサイロ化された SaaS モデルでは、テナント固有の OPA パッケージのルールを実装できます。



このアプローチの欠点は、SaaS アプリケーションに追加するテナントごとに、テナントごとに固有の新しい OPA ルールセットを追加する必要があることです。これは面倒でスケーリングが難しいですが、テナントの要件によっては避けられない場合があります。

または、プールされた SaaS モデルでは、すべてのテナントが同じルールに基づいて認可を決定し、同じデータ構造を使用する場合は、一般的に適用可能なルールを持つ標準 OPA パッケージを使用して、テナントのオンボーディングと OPA 実装のスケーリングを簡単に行うことができます。



可能であれば、一般化された OPA ルールとパッケージ (または仮想ドキュメント) を使用して、各テナントが提供する標準化されたデータに基づいて決定を行うことをお勧めします。このアプローチにより、OPA がルールを通じてどのように意思決定を行うかではなく、テナントごとに OPA に提供されるデータのみを変更するため、OPA を簡単に拡張できます。個々のテナントが独自の決定を必要とする場合や、他のテナントとは異なるデータを OPA に提供する必要がある場合にのみ、テナントrules-per-tenantモデルを導入する必要があります。

PDP の DevOps、モニタリング、ログ記録、およびデータの取得

この提案された認可パラダイムでは、ポリシーは認可サービスに一元化されます。このガイドで説明する設計モデルの目標の 1 つは、ポリシーのデカップリング、またはアプリケーション内の他のコンポーネントから認可ロジックを削除することであるため、この一元化は意図的です。Amazon Verified Permissions と Open Policy Agent (OPA) はどちらも、認可ロジックの変更が必要な場合にポリシーを更新するためのメカニズムを提供します。

Verified Permissions の場合、プログラムでポリシーを更新するメカニズムは AWS SDK によって提供されます (Amazon Verified Permissions API リファレンスガイドを参照)。SDK を使用すると、新しいポリシーをオンデマンドでプッシュできます。さらに、Verified Permissions はマネージドサービスであるため、更新を実行するためにコントロールプレーンやエージェントを管理、設定、または維持する必要はありません。ただし、継続的インテグレーションと継続的デプロイ (CI/CD) パイプラインを使用して、 AWS SDK を使用した Verified Permissions ポリシーストアとポリシー更新のデプロイを管理することをお勧めします。

Verified Permissions では、オブザーバビリティ機能に簡単にアクセスできます。セキュリティインシデントや監査リクエストにすばやく対応できるように AWS CloudTrail、、Amazon CloudWatch ロググループ、Amazon Simple Storage Service (Amazon S3) バケット、または Amazon Data Firehose 配信ストリームに対するすべてのアクセス試行を記録するように設定できます。さらに、 を使用して Verified Permissions サービスの正常性をモニタリングできます AWS Health Dashboard。Verified Permissions はマネージドサービスであるため、そのヘルスは によって維持され AWS、他の AWS マネージドサービスを使用してオブザーバビリティ機能を設定できます。

OPA の場合、REST APIsプログラムでポリシーを更新する方法を提供します。APIsして、確立された場所からポリシーバンドルの新しいバージョンをプルしたり、オンデマンドでポリシーをプッシュしたりできます。さらに、OPA は、新しいエージェントを動的に設定し、検出バンドルを配布するコントロールプレーンによって一元管理できる基本的な検出サービスを提供します。(OPA のコントロールプレーンは、OPA オペレーターが設定および設定する必要があります)。ポリシーエンジンが Verified Permissions、OPA、または別のソリューションであるかどうかにかかわらず、ポリシーをバージョニング、検証、更新するための堅牢な CI/CD パイプラインを作成することをお勧めします。

OPA の場合、コントロールプレーンにはモニタリングと監査のオプションもあります。OPA の承認決定を含むログをリモート HTTP サーバーにエクスポートして、ログを集約できます。これらの決定ログは、監査の目的で非常に重要です。

アクセスコントロールの決定がアプリケーションから切り離される認可モデルの導入を検討している場合は、認可サービスに新しい PDPs のオンボーディングやポリシーの更新のための効果的なモニタリング、ログ記録、CI/CD 管理機能があることを確認してください。

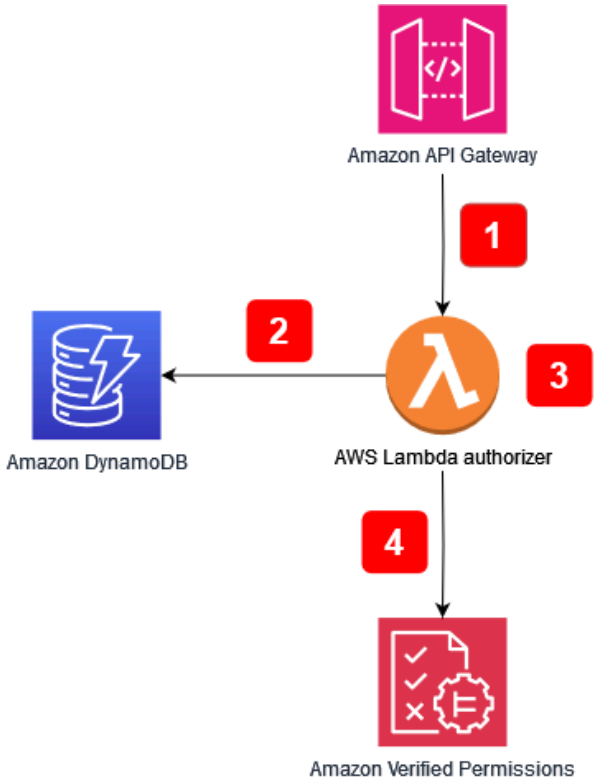
トピック

- Amazon Verified Permissions での PDP の外部データの取得
- OPA での PDP の外部データの取得
- テナントの分離とデータのプライバシーに関する推奨事項

Amazon Verified Permissions での PDP の外部データの取得

Amazon Verified Permissions は、PDP の外部データの取得をサポートしていませんが、ユーザーが提供したデータをスキーマの一部として保存できます。OPA と同様に、認可決定のすべてのデータを認可リクエストの一部として、またはリクエストの一部として渡される JSON Web Token (JWT) の一部として提供できる場合、追加の設定は必要ありません。ただし、Verified Permissions を呼び出すアプリケーションのオーソライザーサービスの一部として、認可リクエストを通じて外部ソースから Verified Permissions に追加データを提供できます。例えば、アプリケーションのオーソライザーサービスは、DynamoDB や Amazon RDS などの外部ソースにデータをクエリでき、これらのサービスは認可リクエストの一部として外部から提供されたデータを含めることができます。

次の図は、追加のデータを取得して Verified Permissions 認可リクエストに組み込む方法の例を示しています。この方法を使用して、RBAC ロールマッピングなどのデータを取得したり、リソースやプリンシパルに関連する追加の属性を取得したり、データがアプリケーションの異なる部分に存在し、ID プロバイダー (IdP) トークンを介して提供できない場合に必要になる場合があります。



アプリケーションフロー:

1. アプリケーションは Amazon API Gateway への API コールを受け取り、そのコールを AWS Lambda オーソライザーに転送します。
2. Lambda オーソライザーは Amazon DynamoDB を呼び出して、リクエストを行ったプリンシパルに関する追加データを取得します。
3. Lambda オーソライザーは、Verified Permissions に対して行われた認可リクエストに追加データを組み込みます。
4. Lambda オーソライザーは Verified Permissions に認可リクエストを行い、認可決定を受け取ります。

この図には、Lambda オーソライザーと呼ばれる Amazon API Gateway の機能が含まれています。この機能は、他の サービスまたはアプリケーションによって提供される APIs で使用できない場合がありますが、オーソライザーを使用して追加のデータを取得し、さまざまなユースケースで Verified Permissions 認可リクエストに組み込むという一般的なモデルをレプリケートできます。

OPA での PDP の外部データの取得

OPA の場合、認可決定に必要なすべてのデータを入力として、またはクエリのコンポーネントとして渡される JSON ウェブトークン (JWT) の一部として提供できる場合、追加の設定は必要ありません。(クエリ入力の一部として JWTs と SaaS コンテキストデータを OPA に渡すのは比較的簡単です)。OPA は、過負荷入力アプローチと呼ばれる で任意の JSON 入力を受け入れることができます。PDP が入力または JWT として含めることができるデータを超えるデータを必要とする場合、OPA はこのデータを取得するためのいくつかのオプションを提供します。これには、バンドル、データのプッシュ (レプリケーション)、動的データ取得が含まれます。

OPA バンドル

OPA バンドル機能は、外部データ取得の次のプロセスをサポートしています。

1. ポリシー適用ポイント (PEP) は認可決定をリクエストします。
2. OPA は、外部データを含む新しいポリシーバンドルをダウンロードします。
3. バンドルサービスは、データソース (複数可) からデータをレプリケートします。

バンドル機能を使用すると、OPA は一元化されたバンドルサービスからポリシーとデータバンドルを定期的にダウンロードします。(OPA はバンドルサービスの実装とセットアップを提供しません)。バンドルサービスからプルされたすべてのポリシーと外部データはメモリに保存されます。このオプションは、外部データサイズが大きすぎてメモリに保存できない場合、またはデータが頻繁に変更された場合は機能しません。

バンドル機能の詳細については、OPA ドキュメントを参照してください。

OPA レプリケーション (データのプッシュ)

OPA レプリケーションアプローチは、外部データ取得の以下のプロセスをサポートします。

1. PEP は認可決定をリクエストします。
2. データレプリケーターはデータを OPA にプッシュします。
3. データレプリケーターは、データソース (複数可) からデータをレプリケートします。

このバンドルアプローチの代わりに、データは OPA によって定期的にプルされるのではなく、 にプッシュされます。(OPA はレプリケーターの実装とセットアップを提供しません)。OPA はすべ

てのデータをメモリに保存するため、プッシュアプローチのデータサイズにはバンドルアプローチと同じ制限があります。プッシュオプションの主な利点は、毎回すべての外部データを置き換えるのではなく、OPA のデータをデルタで更新できることです。これにより、頻繁に変更されるデータセットに対してプッシュオプションがより適切になります。

レプリケーションオプションの詳細については、OPA ドキュメントを参照してください。

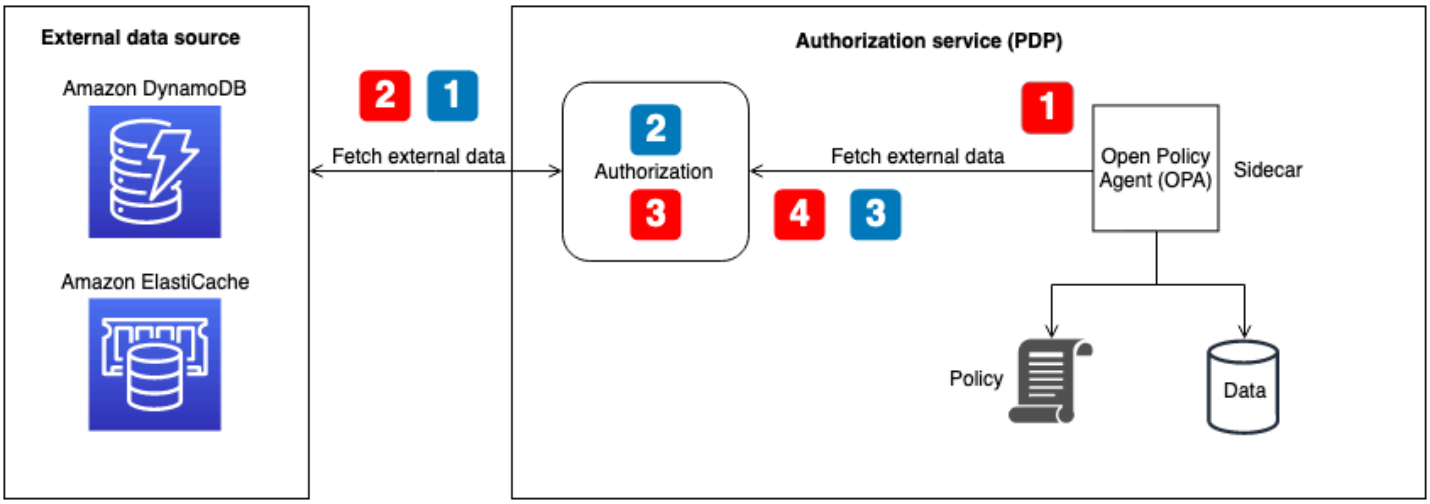
OPA 動的データ取得

取得する外部データが大きすぎて OPA のメモリにキャッシュできない場合、認可決定の評価中に外部ソースからデータを動的にプルできます。このアプローチを使用すると、データは常に最新の状態になります。このアプローチには、ネットワークのレイテンシーとアクセシビリティという 2 つの欠点があります。現在、OPA は HTTP リクエストを介してのみ実行時にデータを取得できます。外部データソースへの呼び出しがデータを HTTP レスポンスとして返すことができない場合は、このデータを OPA に提供するカスタム API またはその他のメカニズムが必要です。OPA は HTTP リクエストを介してのみデータを取得できるため、データの取得速度は極めて重要です。可能であれば、Amazon DynamoDB AWS のサービス などの を使用して外部データを保持することをお勧めします。

プルアプローチの詳細については、OPA ドキュメントを参照してください。

OPA での実装に認可サービスを使用する

バンドル、レプリケーション、または動的プルアプローチを使用して外部データを取得する場合は、認可サービスがこのやり取りを容易にすることをお勧めします。これは、認可サービスが外部データを取得し、それを OPA の JSON に変換して認可を決定できるためです。次の図は、認可サービスがこれら 3 つの外部データ取り出しアプローチでどのように機能するかを示しています。



OPA フローの外部データの取得 – 決定時にバンドルまたは動的データの取得 (図では赤い番号のコールアウトで示されています)。

1. OPA は認可サービスのローカル API エンドポイントを呼び出します。このエンドポイントは、認可の決定時に動的データを取得するバンドルエンドポイントまたはエンドポイントとして設定されます。
2. 認可サービスは、外部データソースをクエリまたは呼び出して外部データを取得します。(バンドルエンドポイントの場合、このデータには OPA ポリシーとルールも含まれます。 バンドルの更新は、データとポリシーの両方を OPA のキャッシュで置き換えます)。
3. 認可サービスは、返されたデータに必要な変換を実行して、予想される JSON 入力に変換します。
4. データは OPA に返されます。バンドル設定のためにメモリにキャッシュされ、動的認可の決定にすぐに使用されます。

OPA フローの外部データの取得 – レプリケーター (図では青色の番号付きコールアウトで示されています)。

1. レプリケーター (認可サービスの一部) は外部データソースを呼び出し、OPA で更新するデータを取得します。これには、ポリシー、ルール、外部データが含まれます。この呼び出しは、設定された頻度で行うことも、外部ソースのデータ更新に応答して行うこともできます。
2. 認可サービスは、返されたデータに必要な変換を実行して、それを予想される JSON 入力に変換します。
3. 認可サービスは OPA を呼び出し、データをメモリにキャッシュします。認可サービスは、データ、ポリシー、およびルールを選択的に更新できます。

テナントの分離とデータのプライバシーに関する推奨事項

前のセクションでは、OPA および Amazon Verified Permissions で外部データを使用して、認可の決定を支援するためのいくつかのアプローチについて説明しました。可能であれば、過負荷入力アプローチを使用して SaaS コンテキストデータを OPA に渡し、OPA のメモリにデータを保存するのではなく、認可の決定を行うことをお勧めします。このユースケースは AWS Cloud Map、 サービスへの外部データの保存をサポートしていないため、 には適用されません。

ロールベースのアクセスコントロール (RBAC) または RBAC および属性ベースのアクセスコントロール (ABAC) ハイブリッドモデルでは、認可リクエストまたはクエリによってのみ提供されるデータが不十分である可能性があります。これは、認可の決定を行うにはロールとアクセス許可を参照す

る必要があるためです。テナントの分離とロールマッピングのプライバシーを維持するために、このデータは OPA 内に存在しないでください。RBAC データは、データベースなどの外部データソースに存在するか、IdP から JWT のクレームの一部として渡す必要があります。Verified Permissions では、各テナントに独自の論理的に分離されたポリシーストアがあるため、RBAC データをテナントごとのポリシーストアモデルのポリシーとスキーマの一部として維持できます。ただし、1 つの共有マルチテナントポリシーストアモデルでは、テナントの分離を維持するために、ロールマッピングデータを Verified Permissions 内に配置しないでください。

さらに、事前定義されたロールを特定のアクセス許可にマッピングするために OPA および Verified Permissions を使用しないでください。これにより、テナントが独自のロールとアクセス許可を定義することが難しくなります。また、認可ロジックがリジッドになり、継続的な更新が必要になります。このガイドラインの例外は、Verified Permissions のテナントごとのポリシーストアモデルです。このモデルでは、テナントごとに個別に評価できる独自のポリシーを各テナントに付与できるためです。

Amazon Verified Permissions

Verified Permissions が潜在的にプライベートな RBAC データを保存できる唯一の場所は、 スキーマです。これはテナントごとのポリシーストアモデルで許容されます。各テナントには独自の論理的に分離されたポリシーストアがあるためです。ただし、1 つの共有マルチテナントポリシーストアモデルでテナントの分離が損なわれる可能性があります。認可の決定を行うためにこのデータが必要な場合は、DynamoDB や Amazon RDS などの外部ソースから取得し、Verified Permissions 認可リクエストに組み込む必要があります。

OPA

RBAC データのプライバシーとテナント分離を維持するための OPA による安全なアプローチには、動的データ取得またはレプリケーションを使用して外部データを取得することが含まれます。これは、前の図に示す認可サービスを使用して、認可決定を行うためにテナント固有またはユーザー固有の外部データのみを提供できるためです。たとえば、レプリケーターを使用して、ユーザーがログインしたときに RBAC データまたはアクセス許可マトリックスを OPA キャッシュに提供し、入力データ で指定されたユーザーに基づいてデータを参照させることができます 。動的にプルされたデータでも同様のアプローチを使用して、認可の決定に関連するデータのみを取得できます。さらに、動的データ取り出しアプローチでは、このデータを OPA にキャッシュする必要はありません。バンドルアプローチは、テナントの分離を維持する動的取得アプローチほど効果的ではありません。これは、OPA キャッシュ内のすべての内容を更新し、正確な更新を処理できないためです。バンドルモデルは、OPA ポリシーと非 RBAC データを更新するための優れたアプローチです。

ベストプラクティス

このセクションでは、このガイドの要点をいくつか示します。各ポイントの詳細な説明については、対応するセクションへのリンクに従ってください。

アプリケーションで動作するアクセスコントロールモデルを選択する

このガイドでは、いくつかのアクセスコントロールモデルについて説明します。アプリケーションとビジネスの要件に応じて、自分に合ったモデルを選択する必要があります。これらのモデルを使用してアクセスコントロールのニーズを満たす方法と、選択したアプローチを変更する必要があるアクセスコントロールのニーズがどのように進化するかを検討してください。

PDP の実装

ポリシー決定ポイント (PDP) は、ポリシーまたはルールエンジンとして特徴付けることができます。このコンポーネントは、ポリシーまたはルールを適用し、特定のアクセスを許可するかどうかの決定を返す責任があります。PDP を使用すると、アプリケーションコードの認可ロジックを別のシステムにオフロードできます。これにより、アプリケーションコードを簡素化できます。また、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認を決定するためのeasy-to-useべき等インターフェイスも提供します。PDP を使用して、アプリケーション全体でテナンシー要件を一貫して適用できます。

アプリケーション内のすべての API に PEPsを実装する

ポリシー適用ポイント (PEP) を実装するには、アプリケーションでアクセスコントロールの適用を行う場所を決定する必要があります。最初のステップとして、PEPs。PEPs。

アプリケーションが API を公開する場合は、その API に対する認可とアクセスコントロールが必要です。

PDP のポリシーエンジンとして Amazon Verified Permissions または OPA を使用することを検討してください。

Amazon Verified Permissions には、カスタムポリシーエンジンよりも利点があります。これは、構築するアプリケーション用のスケーラブルできめ細かなアクセス許可管理および認可サービスで

す。高レベルの宣言型オープンソース言語 Cedar でのポリシーの記述をサポートしています。その結果、Verified Permissions を使用してポリシーエンジンを実装する場合、独自のソリューションを実装するよりも開発作業が少なくて済みます。さらに、Verified Permissions はフルマネージド型であるため、基盤となるインフラストラクチャを管理する必要はありません。

Open Policy Agent (OPA) には、カスタムポリシーエンジンよりも利点があります。Rego による OPA とそのポリシー評価は、高レベルの宣言言語でのポリシーの記述をサポートする、柔軟で構築済みのポリシーエンジンを提供します。これにより、ポリシーエンジンの実装に必要な労力のレベルは、独自のソリューションを構築するよりも大幅に少なくなります。さらに、OPA はすぐに十分にサポートされている認可標準になりつつあります。

DevOps、モニタリング、ログ記録用の OPA のコントロールプレーンを実装する

OPA はソース制御を通じて認可ロジックの変更を更新および追跡する手段を提供しないため、これらの機能を実行するコントロールプレーンを実装することをお勧めします。これにより、特に OPA が分散システムで動作している場合に、OPA エージェントに更新をより簡単に配信できるため、OPA を使用する管理上の負担が軽減されます。さらに、コントロールプレーンを使用して集約用のログを収集し、OPA エージェントのステータスをモニタリングできます。

Verified Permissions でログ記録とオブザーバビリティ機能を設定する

Verified Permissions では、オブザーバビリティ機能に簡単にアクセスできます。セキュリティインシデントや監査リクエストにすばやく対応できるように AWS CloudTrail、、Amazon CloudWatch ロググループ、S3 バケット、または Amazon Data Firehose 配信ストリームに対するすべてのアクセス試行を記録するようにサービスを設定できます。さらに、 を使用してサービスの正常性をモニタリングできます AWS Health Dashboard。Verified Permissions はマネージドサービスであるため、そのヘルスは によって維持され AWS、他の AWS マネージドサービスを使用してオブザーバビリティ機能を設定できます。

CI/CD パイプラインを使用して、Verified Permissions でポリシーストアとポリシーをプロビジョニングおよび更新する

Verified Permissions はマネージドサービスであるため、更新を実行するためにコントロールプレーンやエージェントを管理、設定、または維持する必要はありません。ただし、継続的インテ

グレーションと継続的デプロイ (CI/CD) パイプラインを使用して、 AWS SDK を使用して Verified Permissions ポリシーストアのデプロイとポリシーの更新を管理することをお勧めします。この作業により、手動作業がなくなり、Verified Permissions リソースを変更するときにオペレーターエラーが発生する可能性が低くなります。

認可の決定に外部データが必要かどうかを判断し、それに対応するモデルを選択する

PDP が JSON Web Token (JWT) に含まれるデータのみに基づいて認可の決定を行うことができる場合、通常、これらの決定を支援するために外部データをインポートする必要はありません。Verified Permissions または OPA を PDP として使用している場合、このデータが JWT に含まれていなくても、リクエストの一部として渡される追加の入力を受け入れることもできます。Verified Permissions では、追加データにコンテキストパラメータを使用できます。OPA では、JSON データをオーバーロード入力として使用できます。JWT を使用する場合、コンテキストまたはオーバーロードの入力方法は、通常、別のソースで外部データを維持するよりもはるかに簡単です。認可の決定を行うためにより複雑な外部データが必要な場合、OPA は外部データを取得するためのいくつかのモデルを提供し、Verified Permissions は認可サービスで外部ソースを参照することで認可リクエストのデータを補完できます。

よくある質問

このセクションでは、マルチテナント SaaS アプリケーションでの API アクセスコントロールと認可の実装に関してよく寄せられる質問に対する回答を提供します。

Q. 認可と認証の違いは何ですか？

A.認証とは、ユーザーが誰であるかを検証するプロセスです。認可は、特定のリソースにアクセスするためのアクセス許可をユーザーに付与します。

Q. SaaS アプリケーションの認可とテナント分離の違いは何ですか？

A. テナント分離とは、共有インフラストラクチャで運用されている場合でも、各テナントのリソースを分離するために SaaS システムで使用される明示的なメカニズムを指します。マルチテナント認可とは、インバウンドアクションを承認し、間違ったテナントに実装されないようにすることです。架空のユーザーは認証および認可される可能性がありますが、別のテナントのリソースにアクセスできる可能性があります。認証と認可は必ずしもこのアクセスをブロックするものではありませんが、この目標を達成するにはテナントの分離が必要です。これらの 2 つの概念の詳細については、「SaaS アーキテクチャの基礎」ホワイトペーパーの「テナント分離に関する説明」を参照してください。 AWS SaaS

Q. SaaS アプリケーションのテナント分離を検討する必要があるのはなぜですか？

A.SaaS アプリケーションには複数のテナントがあります。テナントは、顧客組織でも、その SaaS アプリケーションを使用する外部エンティティでもかまいません。アプリケーションの設計方法によっては、テナントが共有 APIs、データベース、またはその他のリソースにアクセスしている可能性があります。テナント分離を維持することが重要です。つまり、リソースへのアクセスを厳密に制御し、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コンストラクトは、あるテナントが別のテナントの個人情報にアクセスできないようにすることが重要です。SaaS アプリケーションは、多くの場合、テナントの分離がアプリケーション全体で維持され、テナントが独自のリソースにのみアクセスできるように設計されています。

Q. アクセスコントロールモデルが必要なのはなぜですか？

A.アクセスコントロールモデルは、アプリケーション内のリソースへのアクセスを許可する方法を決定する一貫した方法を作成するために使用されます。これは、ビジネスロジックと密接に整合しているユーザーにロールを割り当てることで実行できます。または、時刻やユーザーが事前定義された条件を満たしているかどうかなど、他のコンテキスト属性に基づいて実行できます。アクセスコント

ロールモデルは、ユーザーのアクセス許可を決定する認可を決定する際にアプリケーションが使用する基本ロジックを形成します。

Q. API アクセスコントロールはアプリケーションに必要ですか？

A.はい。APIs、発信者が適切なアクセス権を持っていることを常に確認する必要があります。また、広範な API アクセスコントロールにより、適切な分離を維持できるように、テナントに基づいてのみアクセスが許可されます。

Q. ポリシーエンジンまたは PDPs が認可に推奨されるのはなぜですか？

A.ポリシー決定ポイント (PDP) では、アプリケーションコードの認可ロジックを別のシステムにオフロードできます。これにより、アプリケーションコードを簡素化できます。また、APIs、マイクロサービス、バックエンド for Frontend (BFF) レイヤー、またはその他のアプリケーションコンポーネントの承認を決定するためのeasy-to-useべき等インターフェイスも提供します。

Q. PEP とは

A.ポリシー適用ポイント (PEP) は、評価のために PDP に送信される認可リクエストを受信する責任があります。PEP は、データとリソースを保護する必要があるアプリケーション、または認可ロジックが適用されるアプリケーション内の任意の場所に配置できます。PEPsは PDPs と比較して比較的簡単です。PEP は認可決定のリクエストと評価にのみ責任を負い、認可ロジックを組み込む必要はありません。

Q. Amazon Verified Permissions と OPA のどちらを選択すればよいですか？

A. Verified Permissions と Open Policy Agent (OPA) のどちらかを選択するには、ユースケースと固有の要件を常に考慮してください。Verified Permissions は、きめ細かなアクセス許可を定義し、アプリケーション全体のアクセス許可を監査し、アプリケーションのポリシー管理システムを一元化するフルマネージド型の方法を提供しながら、ミリ秒単位の処理でアプリケーションのレイテンシー要件を満たします。OPA は、オープンソースの汎用ポリシーエンジンであり、アプリケーションスタック間でポリシーを統一するのに役立ちます。OPA を実行するには、通常はコンテナまたは AWS Lambda 関数を使用して、 AWS 環境でホストする必要があります。

Verified Permissions はオープンソースの Cedar ポリシー言語を使用し、OPA は Rego を使用します。したがって、これらの言語の 1 つに精通していると、そのソリューションを選択するのを妨げられる可能性があります。ただし、両方の言語について読み、解決しようとしている問題から戻り、ユースケースに最適な解決策を見つけることをお勧めします。

Q: Verified Permissions と OPA に代わるオープンソースの代替手段はありますか？

A. 共通表現言語 (CEL) など、Verified Permissions や Open Policy Agent (OPA) に似たオープンソースシステムがいくつかあります。このガイドでは、スケーラブルなアクセス許可管理ときめ細かな認可サービスとしての Verified Permissions と、さまざまなタイプのアプリケーションと認可要件に広く採用、文書化され、適応可能な OPA の両方に焦点を当てています。

Q. OPA を使用するには認可サービスを記述する必要がありますか、それとも OPA と直接やり取りできますか？

A. OPA を直接操作できます。このガイダンスのコンテキストにおける認可サービスは、認可決定リクエストを OPA クエリに変換するサービスを指します。その逆も同様です。アプリケーションが OPA レスポンスを直接クエリして受け入れることができる場合、この複雑さを追加する必要はありません。

Q. 稼働時間と監査の目的で OPA エージェントをモニタリングするにはどうすればよいですか？

A.OPA はログ記録と基本的な稼働時間モニタリングを提供しますが、そのデフォルト設定はエンタープライズデプロイでは不十分である可能性があります。詳細については、このガイドの前半の「DevOps、モニタリング、ログ記録」セクションを参照してください。

Q. 稼働時間と監査の目的で Verified Permissions をモニタリングするにはどうすればよいですか？

A.検証済みアクセス許可は AWS マネージドサービスであり、 を通じて可用性をモニタリングできます AWS Health Dashboard。さらに、Verified Permissions は、Amazon CloudWatch Logs AWS CloudTrail、Amazon S3、Amazon Data Firehose へのログ記録が可能です。

Q. OPA の実行に使用できるオペレーティングシステムと AWS サービスはどれですか？

A. macOS、Windows、Linux で OPA を実行できます。OPA エージェントは、Amazon Elastic Compute Cloud (Amazon EC2) エージェントだけでなく、Amazon Elastic Container Service (Amazon ECS) や Amazon Elastic Kubernetes Service (Amazon EKS) などのコンテナ化サービスでも設定できます。

Q. Verified Permissions の実行に使用できるオペレーティングシステムと AWSサービスはどれですか？

A.Verified Permissions は AWS マネージドサービスであり、 によって運用されています AWS。Verified Permissions を使用するには、サービスに認可リクエストを行う機能を除いて、追加の設定、インストール、ホスティングは必要ありません。

Q. OPA は で実行できますか AWS Lambda？

A. Go ライブラリとして Lambda で OPA を実行できます。API Gateway Lambda オーソライザーに対してこれを行う方法については、 AWS ブログ記事「オープンポリシーエージェントを使用したカスタム Lambda オーソライザーの作成」を参照してください。

Q. 分散 PDP アプローチと集中型 PDP アプローチのどちらを決定すればよいですか？

A.これはアプリケーションによって異なります。ほとんどの場合、分散型 PDP モデルと集中型 PDP モデルのレイテンシーの違いに基づいて決定されます。概念実証を構築し、アプリケーションのパフォーマンスをテストしてソリューションを検証することをお勧めします。

Q: APIs 以外のユースケースに OPA を使用できますか？

A.はい。OPA ドキュメントには、Kubernetes、Envoy、Docker、Kafka、SSH と sudo、Terraform の例が記載されています。さらに、OPA は Rego 部分ルールを使用してクエリに任意の JSON レスポンスを返すことができます。クエリに応じて、OPA を使用して JSON レスポンスで多くの質問に回答できます。

Q: API 以外のユースケースに Verified Permissions を使用できますか? APIs

A.はい。Verified Permissions は、受信した認可リクエストに対して `ALLOW`または `DENY`レスポンスを提供できます。Verified Permissions は、認可の決定を必要とするアプリケーションまたはサービスに対して認可レスポンスを提供できます。

Q. IAM ポリシー言語を使用して Verified Permissions でポリシーを作成できますか？

A. いいえ。ポリシーを作成するには Cedar ポリシー言語を使用する必要があります。Cedar はお客様のアプリケーションリソースのアクセス許可管理をサポートするように設計されていますが、AWS Identity and Access Management (IAM) ポリシー言語は AWS リソースのアクセスコントロールをサポートするように進化しました。

次のステップ

マルチテナント型 SaaS アプリケーションの承認と API アクセス制御の複雑さは、標準化された、言語に依存しないアプローチによって承認の決定を行うことで克服できます。これらのアプローチには、柔軟で広範な方法でアクセスを強制するポリシー決定ポイント (PDPs) とポリシー適用ポイント (PDPs) が組み込まれています。ロールベースのアクセス制御 (RBAC)、属性ベースのアクセス制御 (ABAC)、またはこの 2 つの組み合わせなど、アクセス制御に対する複数のアプローチを、まとまりのあるアクセス制御戦略に組み込むことができます。認可ロジックをアプリケーションから削除することで、アクセス制御に対処するためのアドホックソリューションをアプリケーションコードに組み込む際のオーバーヘッドがなくなります。このガイドで説明する実装とベストプラクティスは、マルチテナント型 SaaS アプリケーションにおける承認と API アクセス制御の実装方法を知り、標準化することを目的としています。このガイダンスは、情報を収集し、アプリケーションの堅牢なアクセス制御と承認システムを設計するための第一歩として活用できます。次のステップ：

- 承認とテナント分離のニーズを確認し、アプリケーションに適したアクセス制御モデルを選択します。
- Amazon Verified Permissions または Open Policy Agent (OPA) を使用するか、独自のカスタムポリシーエンジンを記述して、テストの概念実証を構築します。
- PEP を実装すべきアプリケーション内の API と場所を特定します。

リソース

リファレンス

- Amazon Verified Permissions ドキュメント (AWS ドキュメント)
- 認可に Amazon Verified Permissions を使用する方法 (AWS ブログ記事)
- Amazon Verified Permissions を使用して ASP.NET Core Apps のカスタム認可ポリシープロバイダーを実装する (AWS ブログ記事)
- Amazon Verified Permissions を使用して PBAC でロールとエンタイトルメントを管理する (AWS ブログ記事)
- テナントごとのポリシーストアでの Amazon Verified Permissions を使用した SaaS アクセスコントロール (AWS ブログ記事)
- OPA 公式ドキュメント
- 企業が最近公開された CNCF プロジェクトを受け入れる必要がある理由 — オープンポリシーエージェント (Janakiram MSV によるForbes 記事、2021 年 2 月 8 日)
- Open Policy Agent を使用したカスタム Lambda オーソライザーの作成 (AWS ブログ記事)
- 「Open Policy Agent を通じて AWS Cloud Development Kit でポリシーをコードとして実現する」(AWS ブログ記事)
- 「コードとしてのポリシー AWS による でのクラウドガバナンスとコンプライアンス」 (AWS ブログ記事)
- Amazon EKS での Open Policy Agent の使用 (AWS ブログ記事)
- オープンポリシーエージェント、Amazon EventBridge、および を使用した Amazon ECS のコードとしてのコンプライアンス AWS Lambda (AWS ブログ記事)
- Kubernetes のポリシーベースの対策 – パート 1 (AWS ブログ記事)
- API Gateway Lambda オーソライザーの使用 (AWS ドキュメント)

ツール

- Cedar プレイグラウンド (ブラウザで Cedar をテストする場合)
- Cedar Github リポジトリ
- Cedar 言語リファレンス
- Rego プレイグラウンド (ブラウザで Rego をテストする場合)
- OPA GitHub リポジトリ

パートナー

- Identity and Access Management パートナー
- アプリケーションセキュリティパートナー
- クラウドガバナンスパートナー
- セキュリティおよびコンプライアンスパートナー
- セキュリティオペレーションおよびオートメーションパートナー
- セキュリティエンジニアリングパートナー

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
Amazon Verified Permissions の詳細と例を追加	Amazon Verified Permissions を使用して PDP を実装するための詳細な説明、例、コードを追加しました。新しいセクションは次のとおりです。 • Amazon Verified Permissions を使用した PDP の実装 • Amazon Verified Permissions の設計モデル • Amazon Verified Permissions マルチテナント設計に関する考慮事項 • Amazon Verified Permissions での PDP の外部データの取得	2024 年 5 月 28 日
明確化された情報	APIs 設計モデルで PEPsを使用した分散 PDP を明確にしました。	2024 年 1 月 10 日
新しい AWS サービスに関する簡単な情報を追加	OPA と同じ機能と利点を提供する Amazon Verified Permissions に関する情報を追加しました。	2023 年 5 月 22 日
—	初版発行	2021 年 8 月 17 日

AWS 規範ガイダンスの用語集

以下は、 AWS 規範ガイダンスによって提供される戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行する。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行する。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) – 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例： Microsoft Hyper-Vアプリケーションを に移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを移行するためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。

- 廃止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

A2A (Agent-to-Agent)

タスクの委任と状態転送をサポートするagent-to-agentコラボレーション用のステートフルプロトコル。

ABAC

「属性ベースのアクセス制御」をご覧ください。

抽象化されたサービス

「マネージドユーザー」をご覧ください。

ACID

「原子性、一貫性、分離性、耐久性 (ACID)」をご覧ください。

アクティブ/アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。アクティブ/パッシブ移行よりも柔軟な方法ですが、さらに多くの作業が必要となります。

アクティブ/パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

[エージェント]

目標を達成するためのツールを使用して、自律的に推論、計画、アクションを実行できる AI システム。

エージェントオペレーション

AI エージェントを本番環境で大規模に構築、テスト、デプロイ、実行するための運用プラクティス。

集計関数

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、SUM や MAX などがあります。

AI

「人工知能」をご覧ください。

AIOps

「AI オペレーション」をご覧ください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーション制御

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、ポートフォリオの検出と分析プロセス の重要な要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「人工知能 (AI) とは何ですか?」をご覧ください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。 AWS 移行戦略での AIOps の使用方法については、オペレーション統合ガイド を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、 AWS Identity and Access Management (IAM) ドキュメントの「 の ABAC AWS」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティーゾーン (AZ)

他のアベイラビリティーゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティーゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立て AWS るための、 のガイドラインとベストプラクティスのフレームワークです。 AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスをまとめています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイダンスを提供します。詳細については、AWS CAF ウェブサイトと AWS CAF のホワイトペーパー を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool (AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱や損害を与えることを目的としたボット。

BCP

「ビジネス継続性計画 (BCP)」をご覧ください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントの「動作グラフのデータ」を参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

それぞれが独立しているが、同一の環境を 2 つ作成するデプロイ戦略。現在のアプリケーションバージョンを 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクローラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

マルウェアに感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されているボットのネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといいます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発したり、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたら、機能ブランチをメインブランチに統合します。詳細については、「ブランチの概要」(GitHub ドキュメント) を参照してください。

ブレークグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たない にすばやくアクセスできるようにします。詳細については、 AWS Well-Architected ガイダンスの「ブレークグラス手順の実装」インジケータを参照してください。

ブラウンフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウンフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、AWSでのコンテナ化されたマイクロサービスの実行ホワイトペーパーの「ビジネス機能を中心に組織化」セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

「AWS クラウド導入フレームワーク」を参照してください

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、現在のバージョン全体を置き換えます。

CCoE

「Cloud Center of Excellence」を参照してください。

CDC

「変更データキャプチャ」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストすること。AWS Fault Injection Service (AWS FIS) を使用して、 AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

シチズンデベロッパー

専門的な技術スキルを持たないノーコード/ローコードプラットフォームを使用して AI アプリケーションを作成するビジネスユーザー。

クライアント側の暗号化

ターゲットがデータ AWS のサービス を受信する前のローカルでのデータの暗号化。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、 AWS クラウド エンタープライズ戦略ブログの CCoE 投稿を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的に、エッジコンピューティングに接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「クラウド運用モデルの構築」を参照してください。

導入のクラウドステージ

組織が、 AWS クラウドへの移行時に通常実行する 4 つの段階。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーンの作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、 AWS クラウド エンタープライズ戦略ブログのブログ記事「クラウドファーストへのジャーニー」と「導入のステージ」で Stephen Orban によって定義されました。移行戦略との関連性については、 AWS「移行準備ガイド」を参照してください。

CMDB

「構成管理データベース (CMDB)」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、GitHub や Bitbucket Cloud があります。コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオといった、ビジュアル形式の情報を分析および抽出する AI の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性がありますが、この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンの単一のエンティティとしてデプロイ

することも、組織全体にデプロイすることもできます。詳細については、 AWS Config ドキュメントの「コンフォーマンスパック」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「継続的デリバリーの利点」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「継続的デリバリーと継続的なデプロイ」を参照してください。

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、 AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、「データ分類」を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行えるアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる ID のみが、期待されるネットワークから信頼できるリソースにアクセスできるようにします。詳細については、「 でのデータ境界の構築 AWS」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、一般的に、大量の履歴データが含まれており、多くの場合、それらはクエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「データベース定義言語」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせます。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

深層学習

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

多層防御

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を採用するときは AWS、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加して、リソースの安全性を確保します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、 AWS Organizations ドキュメントの「AWS Organizationsで使用できるサービス」を参照してください。

トラブルシューティング

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「環境」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、「 AWSでのセキュリティコントロールの実装」の「検出的コントロール」を参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニュファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

スタースキーマにおいて、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、クエリの制約、フィルタリング、結果セットのラベル付けに使用されます。

ディザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

ディザスタによるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」を参照してください。

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ボストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用す

る方法の詳細については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、 AWS CloudFormation を使用してシステムリソースのドリフトを検出したり、 を使用して AWS Control Tower 、ガバナンス要件への準拠に影響するランディングゾーンの変更を検出したりできます。

DVSM

「開発バリューストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」を参照してください。

EDI

「電子データ交換」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。クラウドコンピューティングと比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、ビジネスドキュメントの自動交換。詳細については、「電子データ交換とは」を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティング処理。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

「サービスエンドポイント」を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink 、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの「エンドポイントサービスを作成する」を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (会計、MES、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、 AWS Key Management Service (AWS KMS) ドキュメントの「エンベロープ暗号化」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが使用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。

- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、 AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。 AWS 移行戦略のエピックの詳細については、プログラム実装ガイドを参照してください。

ERP

「エンタープライズリソース計画」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

スタースキーマの中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2 種類の列で構成されます。1 つは測定値が含まれる列、もう 1 つはディメンションテーブルへの外部キーが含まれる列です。

フェイルファスト

開発ライフサイクルを短縮するために、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

では AWS クラウド、アベイラビリティーゾーン、コントロールプレーン AWS リージョン、データプレーンなどの境界で、障害の影響を制限し、ワークロードの耐障害性を向上させるのに役立ちます。詳細については、「AWS 障害分離境界」を参照してください。

機能ブランチ

「ブランチ」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021 年」、「5 月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

LLM に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例 (ショット) からモデルが学習する「インコンテキスト学習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「ゼロショットプロンプト」も参照してください。

FGAC

「きめ細かなアクセス制御」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

変更データのキャプチャによる継続的なデータ複製を利用して、段階的なアプローチではなく、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「基盤モデル」を参照してください。

基盤モデル (FM)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FM により、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「基盤モデルとは何ですか?」を参照してください。

FM ゲートウェイ

基盤モデルへのアクセスを制御および正規化する一元化された仲介者。LLM ゲートウェイとも呼ばれます。

G

生成 AI

AI モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できます。詳細については、「生成 AI とは何ですか?」を参照してください。

ジオブロッキング

「地理的制限」を参照してください。

地理的制限 (ジオブロッキング)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。アクセスを許可する国と禁止する国は、許可リストまたは禁止リストを使って指定します。詳細については、CloudFront ドキュメントの「コンテンツの地理的ディストリビューションの制限」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローは古いと見なされている方法であり、トランクベースのワークフローは推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名ブラウンフィールド) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、、Amazon GuardDuty AWS Security Hub CSPM、、 AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

ガードレール (AI)

エージェントの入力と出力をフィルタリング、検証、制約する安全メカニズムは、責任ある安全な AI の動作を確保するのに役立ちます。

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。AWS は、スキーマの変換に役立つ AWS SCTを提供します。

高可用性 (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

機械学習モデルのトレーニング用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、モデルのパフォーマンスを評価できます。

ヒューhuman-in-the-loop (HitL)

エージェントの実行が重要な決定時点で人間によるレビューと承認のために一時停止するワークフローパターン。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1～4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

IaC

「Infrastructure as Code」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5～20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「インダストリアル IoT」を参照してください。

イミュータブルインフラストラクチャ

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、ミュータブルインフラストラクチャよりも一貫性、信頼性、予測性に優れています。詳細については、 AWS Well-Architected フレームワークにある「イミュータブルインフラストラクチャを使用してデプロイする」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に Klaus Schwab 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

インダストリアル IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「インダストリアル IoT (IIoT) デジタルトランスフォーメーション戦略の構築」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「IoT とは」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

IoT

「IoT」を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、オペレーション統合ガイドを参照してください。

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、「安全でスケーラブルなマルチアカウント AWS 環境のセットアップ」を参照してください。

大規模言語モデル (LLM)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「大規模言語モデル (LLM) とは何ですか?」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースアクセス制御」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの「最小特権アクセス許可を適用する」を参照してください。

リフトアンドシフト

「7 Rs」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

LLM

「大規模言語モデル」を参照してください。

下位環境

「環境」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「機械学習」を参照してください。

メインブランチ

「ブランチ」を参照してください。

マルウェア

コンピュータのセキュリティやプライバシーを侵害するように設計されたソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスを招く可能性があります。マルウェアの例には、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービス がインフラストラクチャレイヤー、オペレーティングシステム、プラットフォームを AWS 運用し、ユーザーがエンドポイントにアクセスしてデータを保存および取

得します。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

「Migration Acceleration Program」を参照してください。

MCP

「モデルコンテキストプロトコル」を参照してください。

モデルコンテキストプロトコル (MCP)

エージェントとツール間の通信のためのステートレスプロトコル。

MCP サーバー

Model Context Protocol を通じて 1 つ以上のツールを公開するサービス。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。詳細については、AWS 「 Well-Architected フレームワーク」の「メカニズムの構築」を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。 アカウントが 組織のメンバーになることができるのは、一度に 1 つのみです。

MES

「製造実行システム」を参照してください。

Message Queuing Telemetry Transport (MQTT)

発行/サブスクライブのパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある IoT デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれ

る場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、AWS 「サーバーレスサービスを使用したマイクロサービスの統合」を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「 でのマイクロサービスの実装 AWS」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドに移行するための強力な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、AWS 移行戦略 の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストおよび所有者、移行エンジニア、デベロッパー、およびスプリントで作業する DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20～50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの移行ファクトリーに関する解説とCloud Migration Factory ガイドを参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例としては、ターゲットサブネット、セキュリティグループ、 AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、 AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。MPA ツール (ログインが必要) は、すべての AWS コンサルタントと APN パートナーコンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、移行準備状況ガイドを参照してください。MRA は、AWS 移行戦略の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。詳細については、この用語集の 7 Rs エントリと、「組織を動員して大規模な移行を加速する」を参照してください。

ML

「機械学習」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「AWS クラウドでのアプリケーションのモダナイズ戦略」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定された

ギャップに対処するためのアクションプランが得られます。詳細については、「AWS クラウドでのアプリケーションのモダナイゼーションの準備状況を評価する」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1 つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、「モノリスをマイクロサービスに分解する」を参照してください。

MPA

「Migration Portfolio Assessment」を参照してください。

MQTT

「Message Queuing Telemetry Transport」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」 または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、イミュータブルインフラストラクチャの使用をベストプラクティスとして推奨しています。

O

OAC

「オリジンアクセス制御」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織変更管理」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「オペレーション統合」を参照してください。

Ola

「オペレーショナルレベルアグリーメント」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「Open Process Communications - Unified Architecture」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、 AWS Well-Architected フレームワークの「Operational Readiness Reviews (ORR)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携させるハードウェアおよびソフトウェアシステム。製造分野では、Industry 4.0 への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、オペレーション統合ガイドを参照してください。

組織の証跡

組織 AWS アカウント 内のすべての のすべてのイベント AWS CloudTrail をログに記録する によって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウント に作成され、各アカウントのアクティビティを追跡します。詳細については、CloudTrail ドキュメントの「組織の証跡の作成」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。 AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードにより、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、OCM ガイドを参照してください。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、 AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUT および DELETEリクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。OAC も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用テクノロジー」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続を処理する VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントのアクセス許可の境界を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

「個人を特定できる情報」を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「プログラマブルロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

次の操作を可能にするオブジェクト: アクセス許可を定義する (ID ベースのポリシーを参照)。アクセス条件を指定する (リソースベースのポリシーを参照)。 AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する (サービスコントロールポリシーを参照)。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「移行の準備状況の評価」を参照してください。

述語

`true` または `false` を返すためのクエリ条件。一般的に、`WHERE` 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、「AWSでのセキュリティコントロールの実装」の「予防的コントロール」を参照してください。

プリンシパル

アクションを実行し AWS 、リソースにアクセスできる のエンティティ。このエンティティは通常、、IAM AWS アカウントロール、または ユーザーのルートユーザーです。詳細については、IAM ドキュメントの「ロールに関する用語と概念」にあるプリンシパルを参照してください。

プライバシーバイデザイン

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイ防止を目的としたセキュリティコントロール。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、プロビジョニングされません。詳細については、 AWS Control Tower ドキュメントの「コントロールリファレンスガイド」および「 セキュリティコントロールの実装」の「プロアクティブコントロール」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「環境」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの LLM プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

発行/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの MES の場合、マイクロサービスは、他のマイクロサービスがサブスクライブ可能なチャネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RAG

「検索拡張生成」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RCAC

「行と列のアクセス制御」を参照してください。

リードレプリカ

読み取り専用に使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

リアーキテクト

「7 Rs」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

「7 Rs」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョン は、耐障害性、安定性、耐障害性を提供するために、他の とは独立しています。詳細については、「アカウントが使用できる AWS リージョン を指定する」を参照してください。

リグレッション

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「7 Rs」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 Rs」を参照してください。

リプラットフォーム

「7 Rs」を参照してください。

再購入

「7 Rs」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。 AWS クラウドでの回復力を計画する際には、一般的に、高可用性とディザスタリカバリが考慮されます。詳細については、「AWS クラウド の耐障害性」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R) 、説明責任 (A) 、協議 (C) 、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は RACI マトリックスと呼ばれます。

レスポンシブコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、「 AWSでのセキュリティコントロールの実装」の「レスポンシブコントロール」を参照してください。

保持

「7 Rs」を参照してください。

廃止

「7 Rs」を参照してください。

検索拡張生成 (RAG)

生成 AI の技術。これにより、LLM では、レスポンスの生成前に、トレーニングデータソースの外部にある信頼できるデータソースが参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「RAG (検索拡張生成) とは何ですか?」を参照してください。

ローテーション

定期的にシークレット情報を更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「目標復旧時点」を参照してください。

RTO

「目標復旧時間」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdP) が使用しているオープンスタンダード。この機能を使用すると、フェデレーティッドシングルサインオン (SSO) が有効になるため、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、 にログイン AWS マネジメントコンソール したり AWS 、API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントの「SAML 2.0 ベースのフェデレーションについて」を参照してください。

SCADA

「監視制御とデータ取得」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

暗号化された形式で保存する AWS Secrets Managerパスワードやユーザー認証情報などの機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの概要」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、主に 4 つの種類があります。4 つとは、予防、検出、レスポンシブ、プロアクティブです。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントへの自動レスポンスまたは自動修復を目的として、事前定義およびプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ検出的または応答的な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報の更新などがあります。

サーバー側の暗号化

送信先で、それ AWS のサービス を受け取る によるデータの暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、 AWS Organizations ドキュメントの「サービスコントロールポリシー」を参照してください。

サービスエンドポイント

のエントリポイントの URL AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、「AWS 全般のリファレンス」の「AWS のサービス エンドポイント」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

サービスレベルインジケータによって測定され、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、お客様はクラウドのセキュリティを担当します。詳細については、「責任共有モデル」を参照してください。

シャドウ AI

組織内の管理対象チャネルの外部で構築または使用される認可されていない AI アプリケーション。

SIEM

「Security Information and Event Management システム」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

スプリットアンドシードモデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お

客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「AWS クラウドでのアプリケーションをモダナイズするための段階的アプローチ」を参照してください。

SPOF

「単一障害点」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1 つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1 つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、データウェアハウスやビジネスインテリジェンスを用途とするように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として Martin Fowler により提唱されました。このパターンの適用方法の例については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。

監視制御とデータ取得 (SCADA)

製造分野において、ハードウェアとソフトウェアを使用して物理アセットと本番運用をモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。Amazon CloudWatch Synthetics を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、LLM に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、 リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「AWS リソースのタグ付け」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要のある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「環境」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

tool

エージェントが外部システムでオペレーションを実行するために呼び出すことができる関数または API。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、 AWS Transit Gateway ドキュメントの「トランジットゲートウェイとは」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要なときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、 ドキュメントの「 を他の AWS のサービス AWS Organizations で使用する AWS Organizations 」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2 枚のピザを分け合えることができるくらい小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性 は、データに固有のノイズとランダム性によって引き起こされます。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「環境」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「VPC ピア機能とは」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといったタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「Write-Once-Read-Many」を参照してください。

WQF

「AWS ワークロード資格フレームワーク」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャは、イミュータブルと見なされます。

Z

ゼロデイエクスプロイト

ゼロデイ脆弱性を悪用した攻撃 (一般的にマルウェアによる)。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

LLM にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM は、事前トレーニング済みの知識を使用してタスクを処理する必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「数ショットプロンプト」も参照してください。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。