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

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

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 24
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 による 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

ライブラリとしての分散 PDP の使用 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
テナントのデータの分離と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
アプリケーションに適した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
承認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 112
H 114
I 115
L 117
M 118
O 122
P 125
Q 128
R 128
S 131
T 135
U 136
V 137
W 137
Z 138
..... cxxxix

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 の拡散と、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には、マイクロサービス、バックエンドのフロントエンド (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制御をソリューションで強制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、このコードにアクセスできるデベロッパーが誤って、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、エラーが発生しやすく、安全ではありませんでした。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、テナンシー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、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一元化して記述できます。
- 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。
- 抽象化により、デベロッパーによる認可ロジックへの偶発的な変更を防ぐことができます。
- 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るため、監査は簡素化されます。
- 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、複雑な認可要件や厳格な 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 つの特定の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。このガイドで提供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について通知し、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 (""")、それらのポリシーが認可決定に達すると評価されるポリシー決定ポイント (PDP)、および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、"" 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認可ロジックを分離する – 認可ロジックは、アプリケーションコードに埋め込まれたり、アドホックな強制メカニズムによって実装されたりすると、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別に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すように試みます。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。このガイドで説明するアーキテクチャにより、モデルのいずれかまたは両方を正常に採用できます。
- 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とって特に重要です。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 2 つの概念の違いの詳細については、「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」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

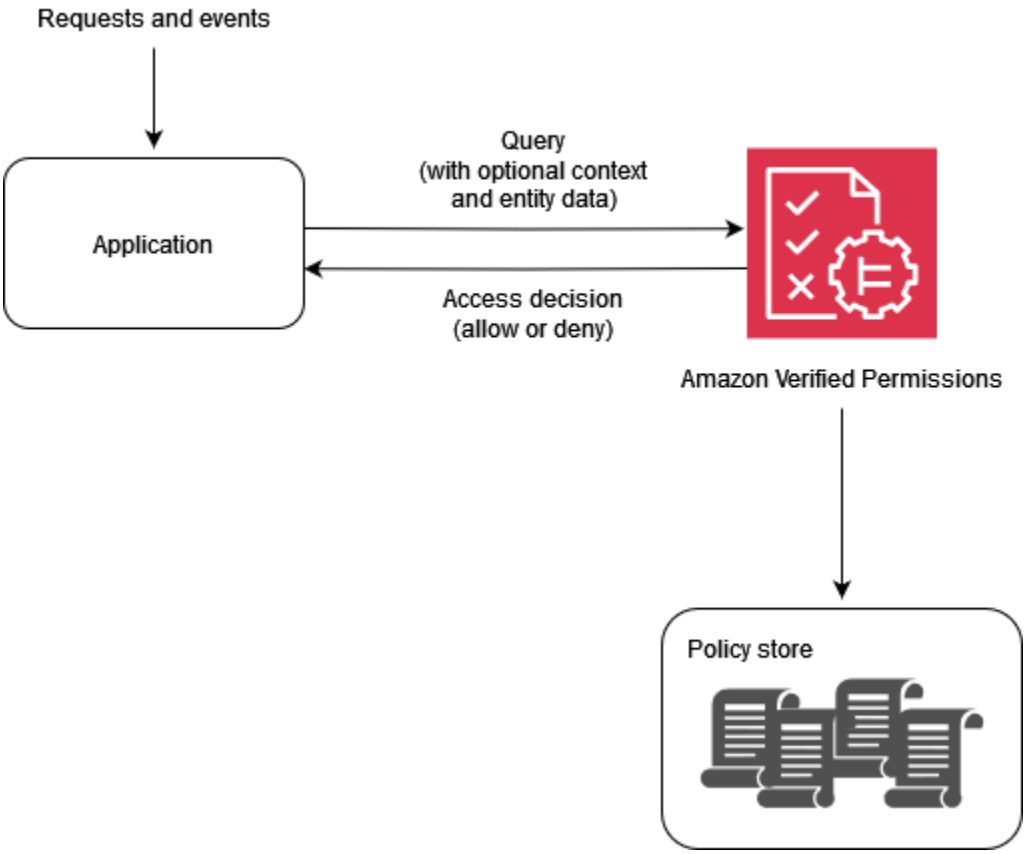
Amazon Verified Permissions を使用した PDP の実装

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、過度に特権があるか無効であるアクセス許可を強調表示することができます。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、デベロッパーがより安全なアプリケーションをより迅速に構築するのに役立ちます。Verified Permissions は、認可ロジックとアプリケーションロジックを分離することで、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に最小特権と継続的な検証を実装することで、デベロッパー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 つのテナントが、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詳細度を定義できます。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 ユースケースの候補として最適です。Verified Permissions は、すべてのテナントに適用されるポリシーを作成する機能もサポートしているため、SaaS プロバイダーとしての不正アクセスを防ぐためにガードレールポリシーを適用するのは簡単です。



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、ユーザーが情報を共有し、コラボレーションするのに

役立つアプリケーションを構築できます。Verified Permissions は、ID 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、ポリシーを一元的に作成して保存し、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型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、ポリシーを作成し、プロビジョニングの一部として自動化できます。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。

認可の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て、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、および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、デベロッパー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。管理者とデベロッパーは、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`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承認ルールをサポートし、ロールベースのアクセス制御 (RBAC) と属性ベースのアクセス制御 (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。

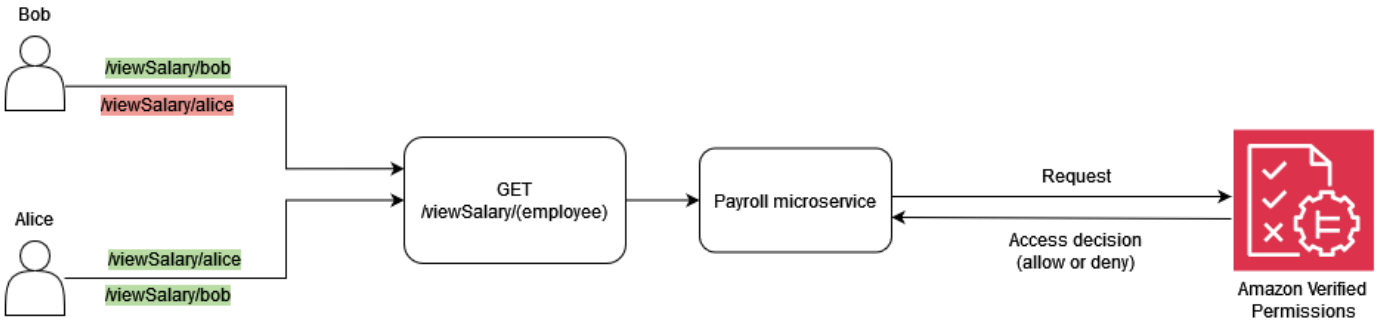
- 表現性 – Cedar は認可のユースケースをサポートすることを目的として構築されており、人間の読みやすさを念頭に置いて開発されました。
- パフォーマンス – Cedar は、迅速な取得のためのインデックス作成ポリシーをサポートし、制限されたレイテンシーで高速でスケーラブルなリアルタイム評価を提供します。
- 分析 – Cedar は、ポリシーを最適化し、セキュリティモデルを検証できる分析ツールをサポートしています。

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へのアクセスが許可され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 と 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`メソッドに関連する 2 つの一般的なビジネスルールを適用します。従業員には自分の給与を表示でき、従業員には自分の給与をレポートするすべての人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。

従業員が各自の給与を表示できるビジネスルールを満たすには、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。例えば、ボブ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、以下を出力として返します。 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 true である必要があります。

従業員は、自分にレポートするすべての人の給与を表示できます。

このビジネスルールを満たすために、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、リクエスト内のリソースにプリンシパルowner.managerと等しい属性があるかどうかを に評価します。例えば、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、 タイプを の値Employeeで 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。その後、 タイプの owner 属性を の値Salaryで関連付ける必要がありますSalary-Bob。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、まず owner 属性をチェックします。 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、従業員は報告者全員の給与を表示でき、Verified Permissions は各ビジネスルールの条件を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、自分自身の給与と、自分にレポートする全員の給与を表示できます。

前の承認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principal、または とowner等しい属性があるかどうかを に評価しますprincipal。

```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
```

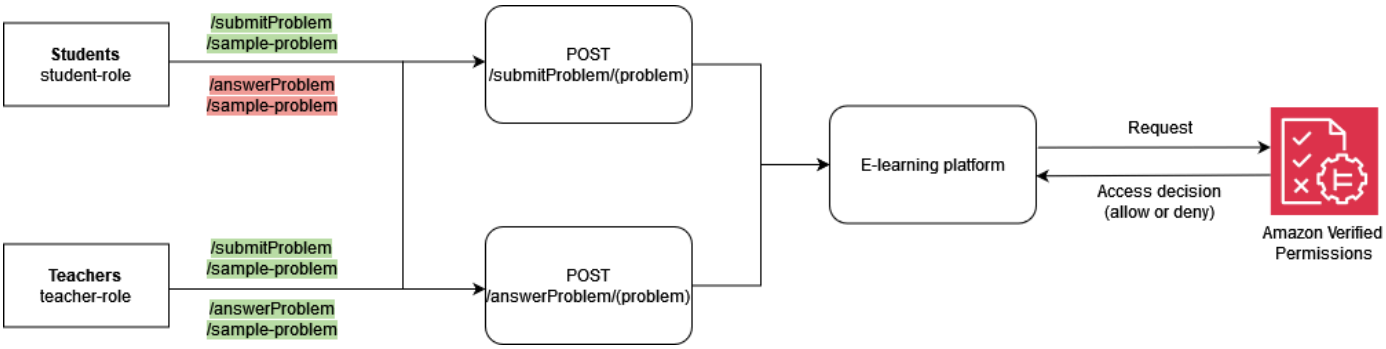
```
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

例えば、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 の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 にリンクすることによって表現されます。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 の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 アクションの決定に対して評価されますRole。

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

次のポリシーは、タイプ 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 の決定に評価Teachersされますrole。

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

これらのポリシーで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、認可呼び出しに関係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、 という親を持つ User::"Bob" と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BobRole::"Students"。ユーザーグループのエンティティRole::"Students"はアクション の実行のみが許可されるためsubmitProblem、この認可リクエストは に評価されますDENY。

一方、 の値Userを持ち、 グループの一部である タイプが answerProblemアクションを実行Role::"Teachers"しようとするAliceと、ポリシーによって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 アクションを実行できるように指示されるため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"
  },
```

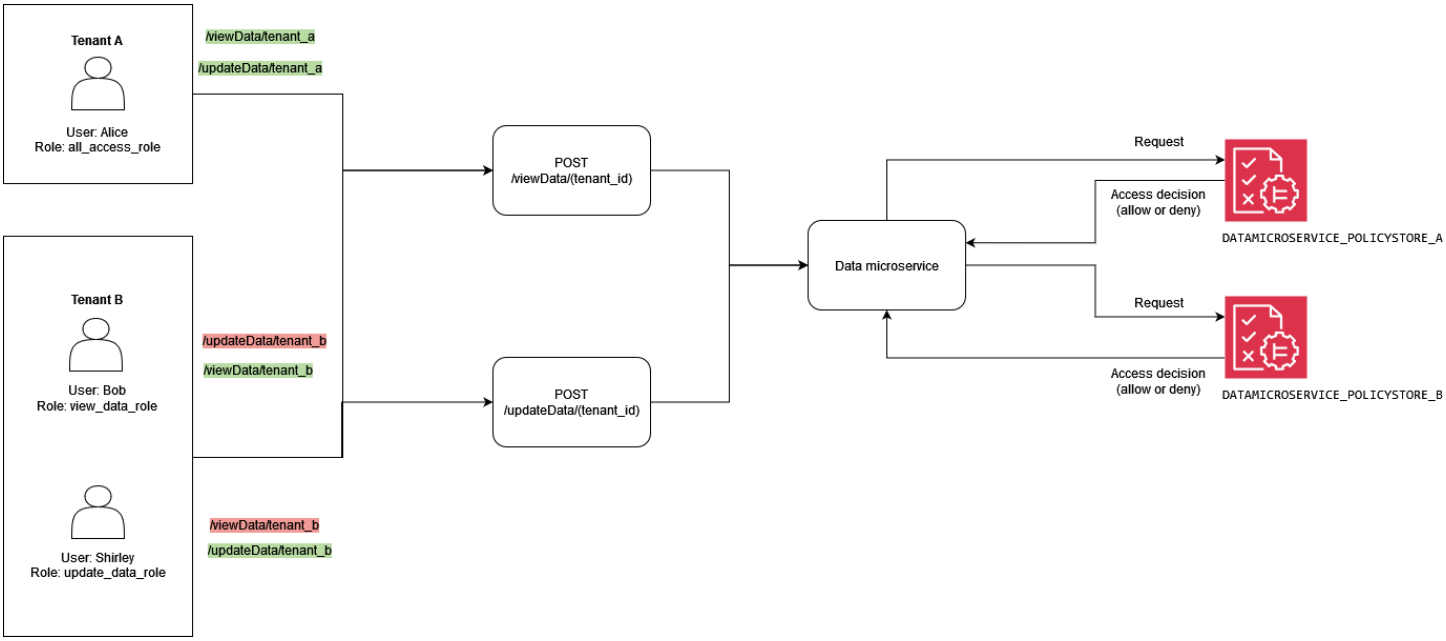
```
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "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、データが論理的または物理的にシステム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設計パターンを使用することは、Verified Permissions によるアクセスコントロール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が、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ロールがあるため、この評価では 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 から発信されるため、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 アクションを実行したいBobと考えていますSampleData。ただし、 Bobは、そのリソースupdateDataに対する 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": []
      }
    ]
  }
}
```

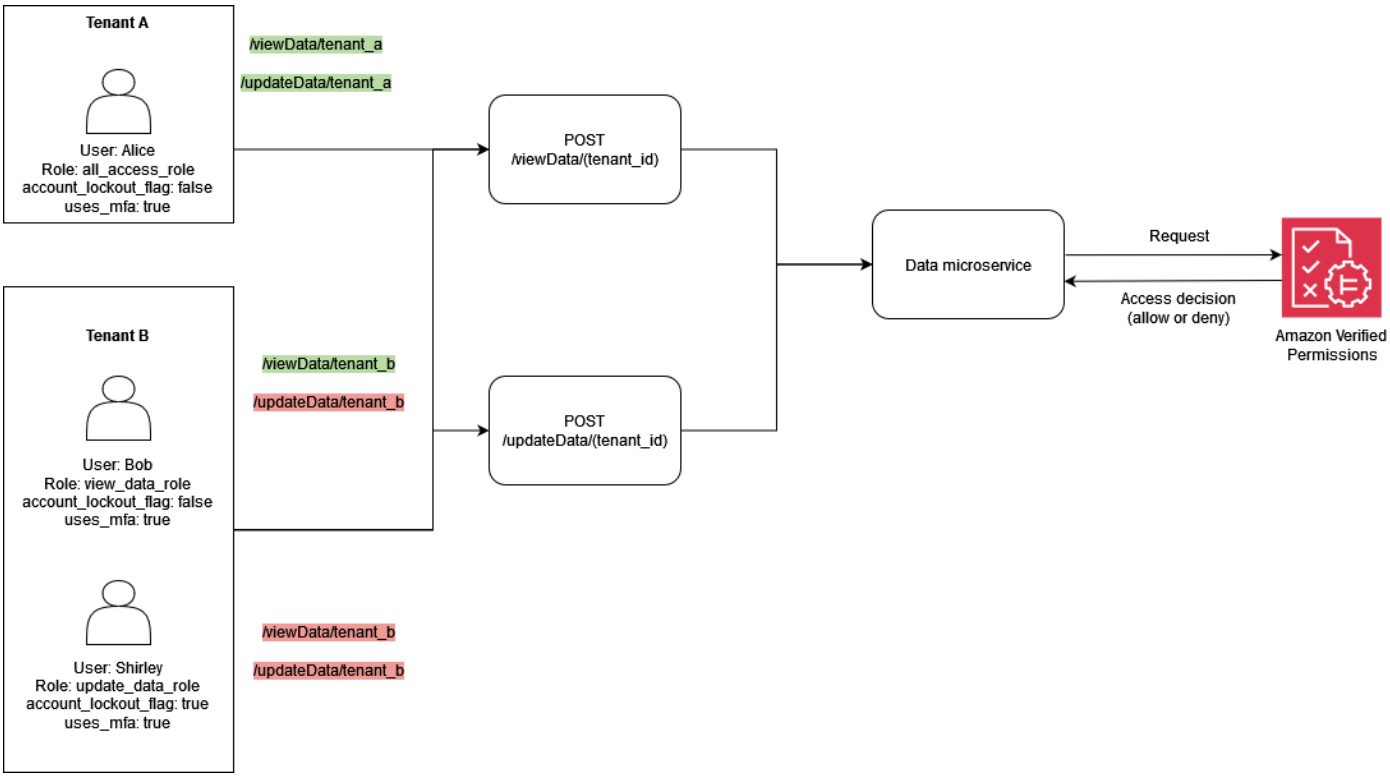
この 3 番目の例では、 はリソース に対して viewDataアクションを実行User Alice しようとしますSampleData。このリクエストは、プリンシパルがテナント A にAlice属しているため、DATAMICROSERVICE_POLICYSTORE_Aポリシーストアに送信されます。 Aliceはタイプ allAccessRoleのグループの一部でありRole、 リソースに対して viewDataアクションを実行することを許可します。そのため、リクエストは ALLOW決定になります。

```
{
```

```
  "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ために異なるアプローチを採用しており、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ています。



この例では、前の例と同様に、テナント A とテナント B に認可の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。

この例では、ユーザーロック機能を実装account_lockout_flagするために、認可リクエストのUserエンティティプリンシパルに 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DENYすべての権限を与えます。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Userに取り消されるまで、 に対して有効になります。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。この例では、この検証を実装するために

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 の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ない限り、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。

また、このポリシーにより、プリンシパルが のメンバーであることが保証されallAccessRole、アクションが viewDataおよび に制限されます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評価されることを検証しますtrue。

同様に、次のポリシーにより、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、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必要があります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確認しupdateDataRole、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、タイプ Userで値が のプリンシパルは、ロール で 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、適用されるリソースのタイプは SampleData ですData。親エンティティTenantAとして SampleData があります。

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 属性が に評価される必要がありますTenantA。2 番目の条件では、プリンシパルの 属性が account_lockout_flagである必要がありますfalse。最終条件では、コンテキストを uses_mfaにする必要がありますtrue。3 つの条件がすべて満たされているため、リクエストは ALLOW 決定を返します。

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
```

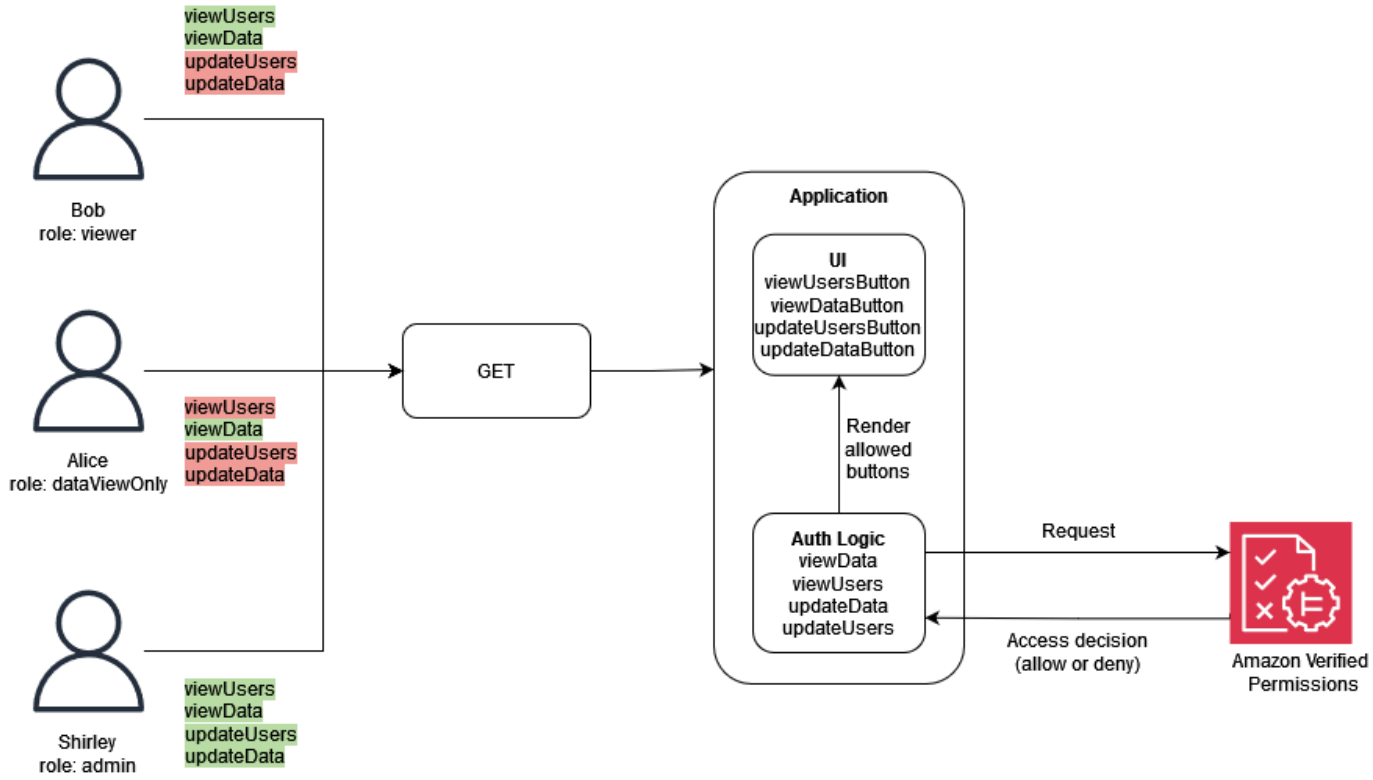
```
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "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、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す。また、リソースを Dataまたは タイプに関連付ける必要があります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できるように指定します。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"]
);
```

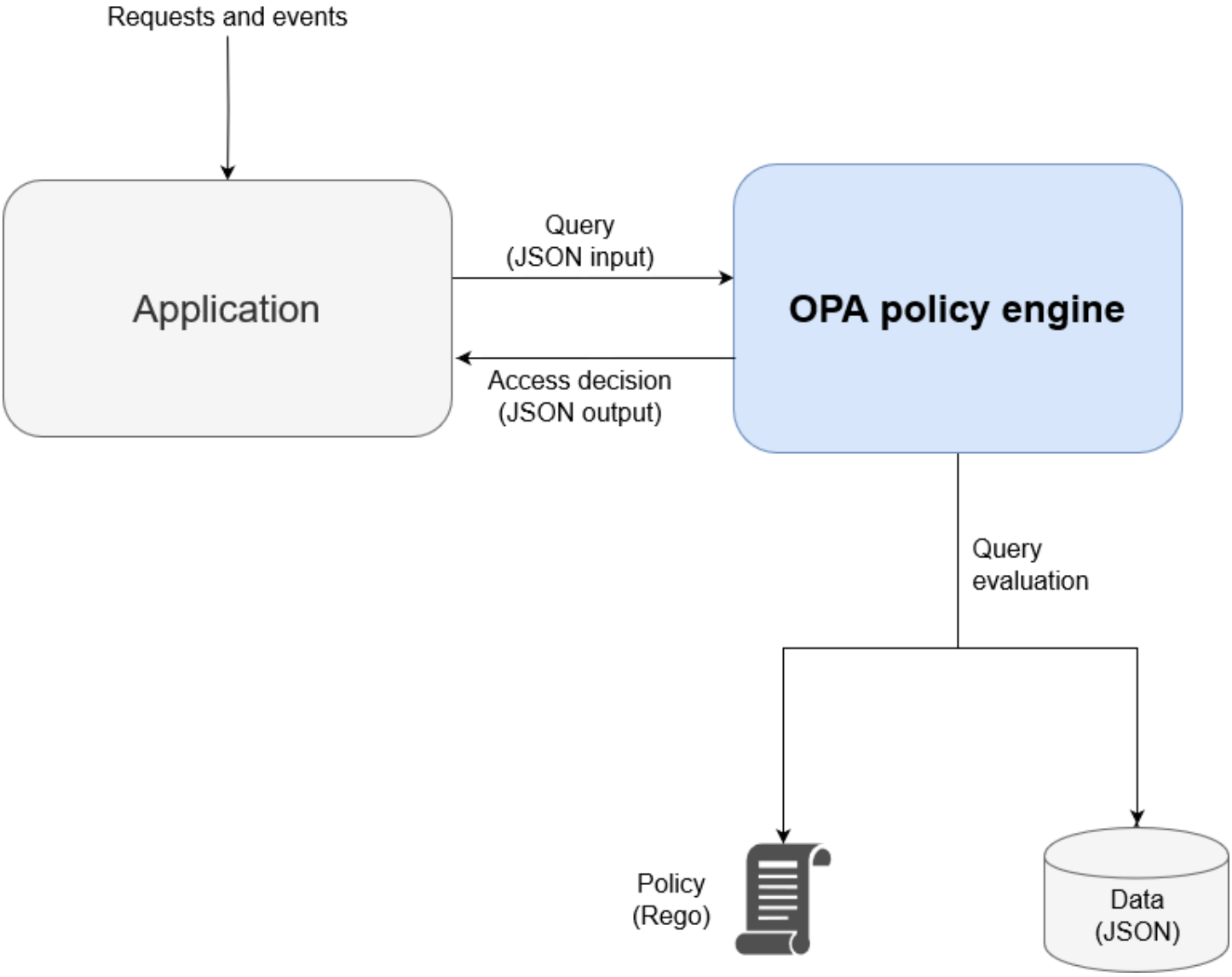
次のポリシーでは、 の値Roleを持つ タイプがデータとユーザーを編集して表示adminできるように指定します。このポリシーの承認ALLOW決定には、updateData、、updateUsersviewData,または のアクションが必要です。またviewUsers、リソースをタイプ Dataまたは に関連付ける必要がありますUsers。ALLOW 決定により、UI は updateDataButton、、updateUsersButton、viewDataButtonの 4 つのボタンをすべてレンダリングできますviewUsersButton。

```
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。Rego という高レベルの宣言言語を使用して、ポリシーとルールをドラフトします。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口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、認可の決定の取得を簡単かつ簡単にします。承認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 には、JSON ウェブトークン (JWTs。
- OPA は広く認められている認可標準です。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、Rego で記述されたポリシーを、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。

OPA が自動的に提供しないことは 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 には、管理 API を公開してポリシーの更新、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がいくつか用意されていますが、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、JSON/YAML 入力とデータを受け入れ、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作成できます。Rego が実行できる決定の例を次に示します。

- 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どのくらいですか？
- 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？

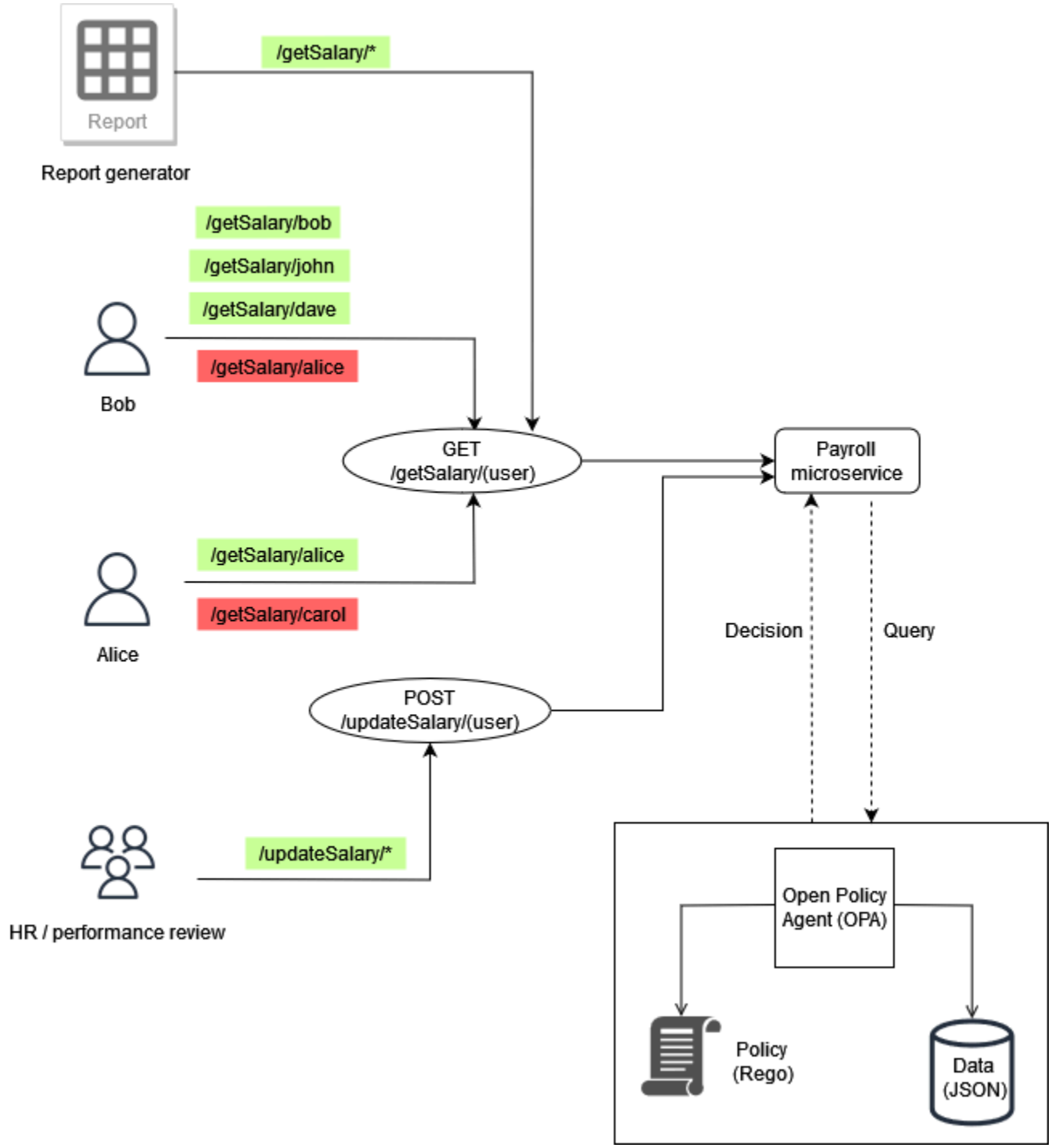
これらの質問に答えるために、Rego はこれらの決定を行う方法に関する基本的な哲学を採用しています。Rego でポリシーを作成する際の 2 つの重要な原則は次のとおりです。

- 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 データまたは YAML データとして表すことができます。
- ポリシーは、データに適用されるロジックです。

Rego は、JSON/YAML データの入力の評価方法に関するロ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。この問題の通常の解決策は 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ロ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、OPA 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へのアクセスを許可されるユーザーに関するアクセス決定を行う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 と OPA の機能のすべてを網羅しているわけでも、完全に調査しているわけでもありません。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 つは次のとおりです。

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、Rego で記述された次のポリシーが 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。Rego ルール内の式も 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pathgetSalary前に、リスト内の最初の要素が であることを確認しますuser。最後に、リクエストuserを行う が user変数と一致することを確認する/getSalary/bobことでinput.user、アクセスされるパスが 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、自分にレポートするすべての人の給与を読み取ることができます。

この例では、OPA は外部データにアクセスして、アクセスの決定に役立てることができます。

```
"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、 の値にレポート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接報告者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、自分にレポートする全員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
```

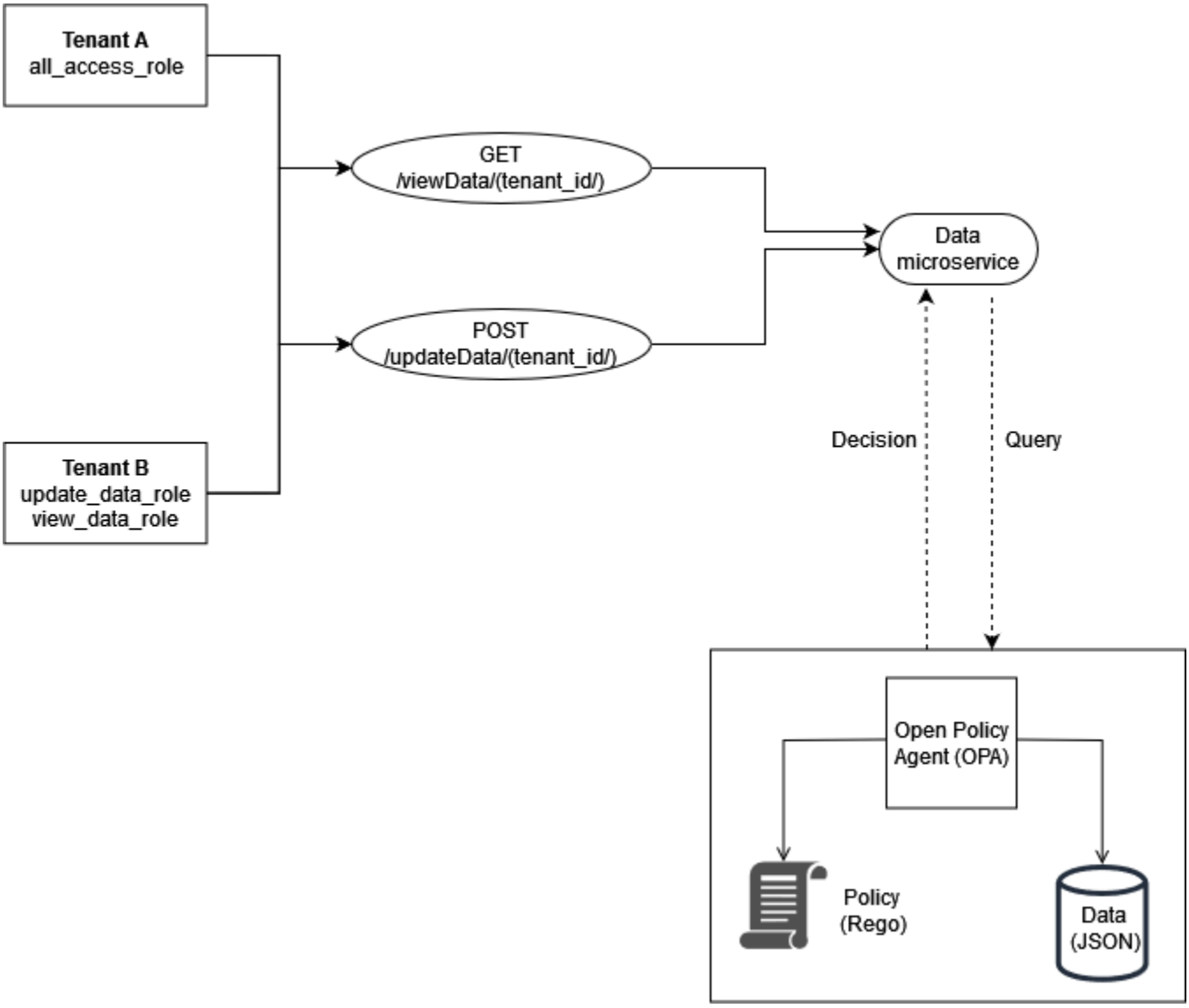
```
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

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 allowは、Rego で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、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では、非常に基本的なものであり、Rego と OPA の機能の完全または徹底的な調査は行いません。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、高レベルのロールで提供される情報に基づいて、OPA が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、テナントユーザーによって定義された場合、テナント定義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示すには、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 ポリシーのクエリされたルールを使用して、指定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に関連付けられた 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役立つかを示しています。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、ロールが異なる別のテナントの OPA クエリ入力と同じ出力を返します。これは、この呼び出しが 用/tenant_bであり、RBAC データの view_data_role には引き続き、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"
}
```

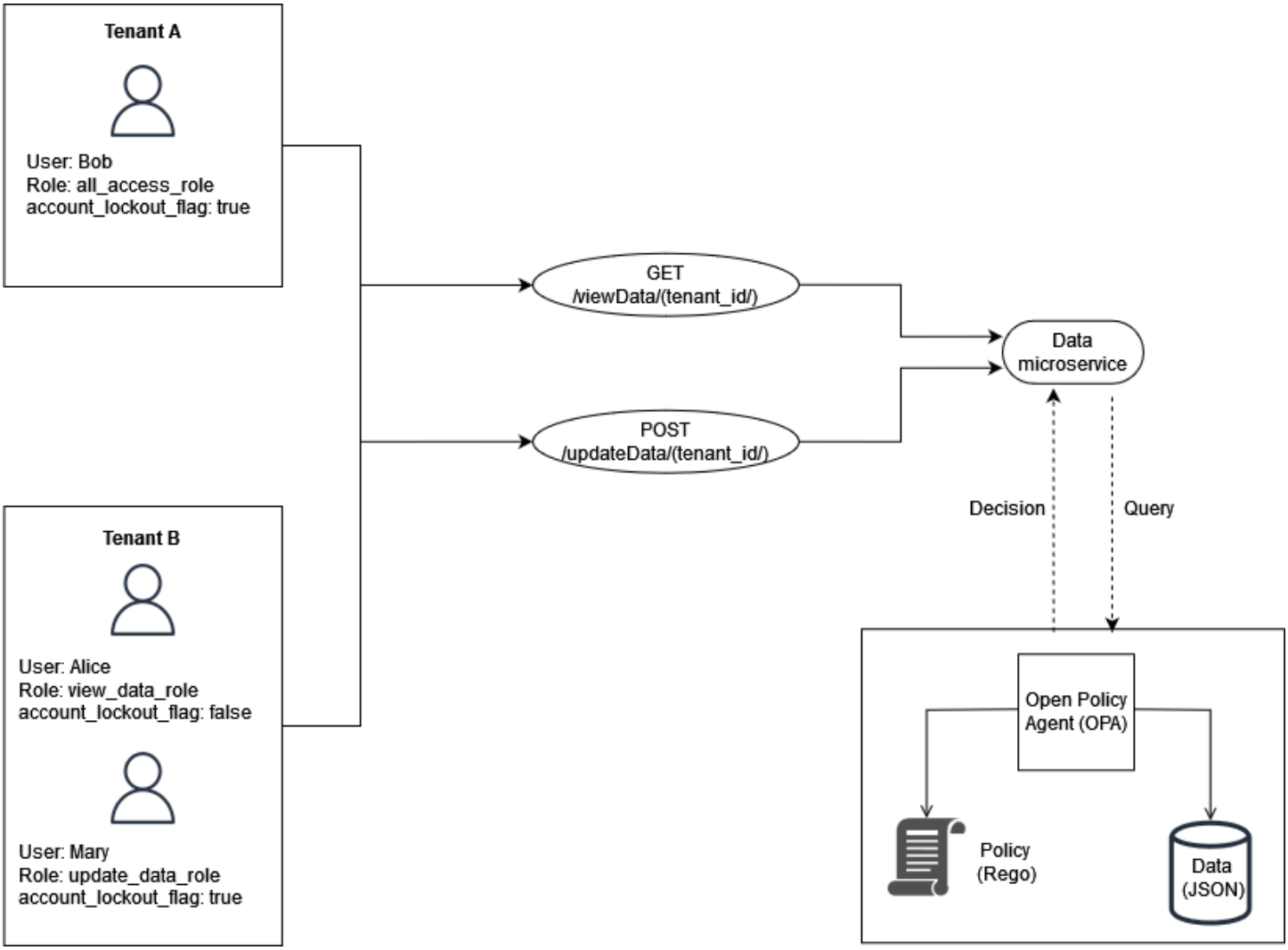
このクエリ入力は、 allowUpdateData ルールで評価されると、次の認可決定を返します。

```
{
    "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、 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"
}
```

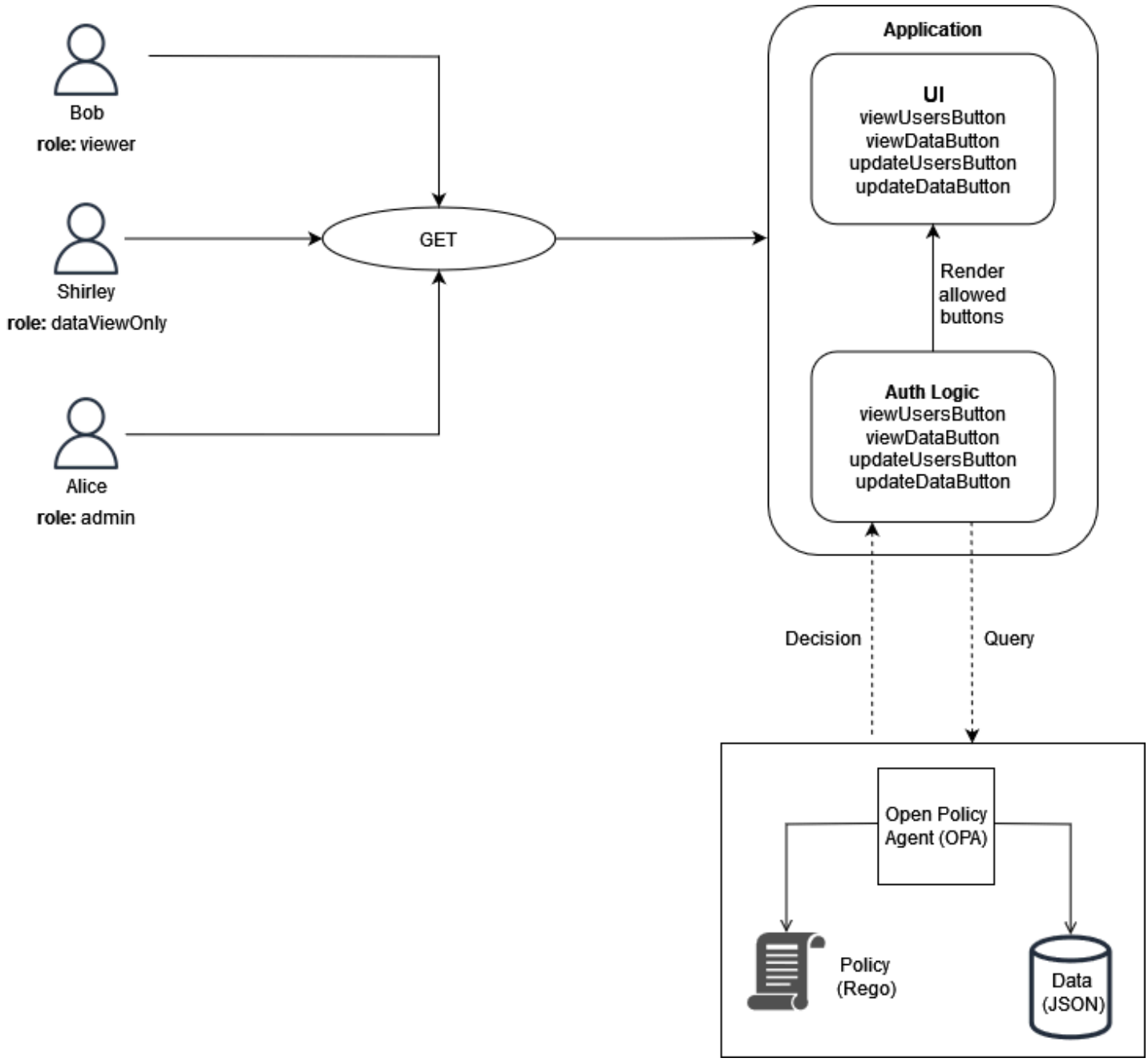
アクセス決定のためにクエリされるルールは、前の例と似ていますが、 account_lockout_flag 属性をチェックするための追加の行が含まれています。

```
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 allowViewData true用であり、Bob 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 による 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 、Shirley の 、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したいとします。UI はユーザーからリクエストを受け取ると、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 付き`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たときに 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、、updateUsersButton、および viewDataButton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 を実装する別の方法は、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承認の決定を行います。このソリューションと Verified Permissions または OPA の使用の主な違いは、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) を実装するための高レベルの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：

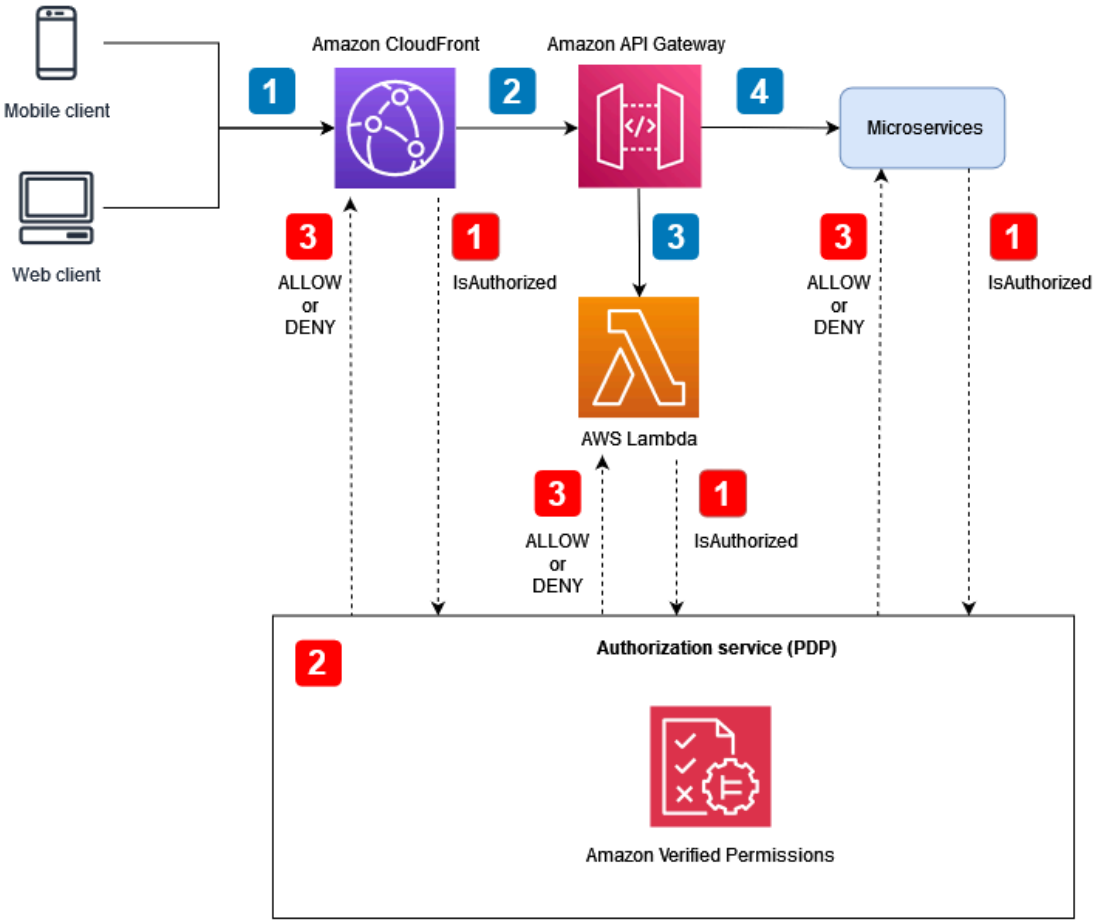
- Amazon Verified Permissions の設計モデル
- OPA の設計モデル

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、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管理できるシステムを作成するための業界のベストプラクティスに従います。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) を持つ認証済みユーザーは、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で実装されます。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の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。RESTful API または AWS SDK を使用して Verified Permissions を操作できます。

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 の利点は、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 ポリシーを一元的な場所に保存し、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。また、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、特にインターネット接続に一貫性がないために Verified Permissions へのアクセスが断続的または不可能な場合、実行可能な代替手段となる可能性があります。

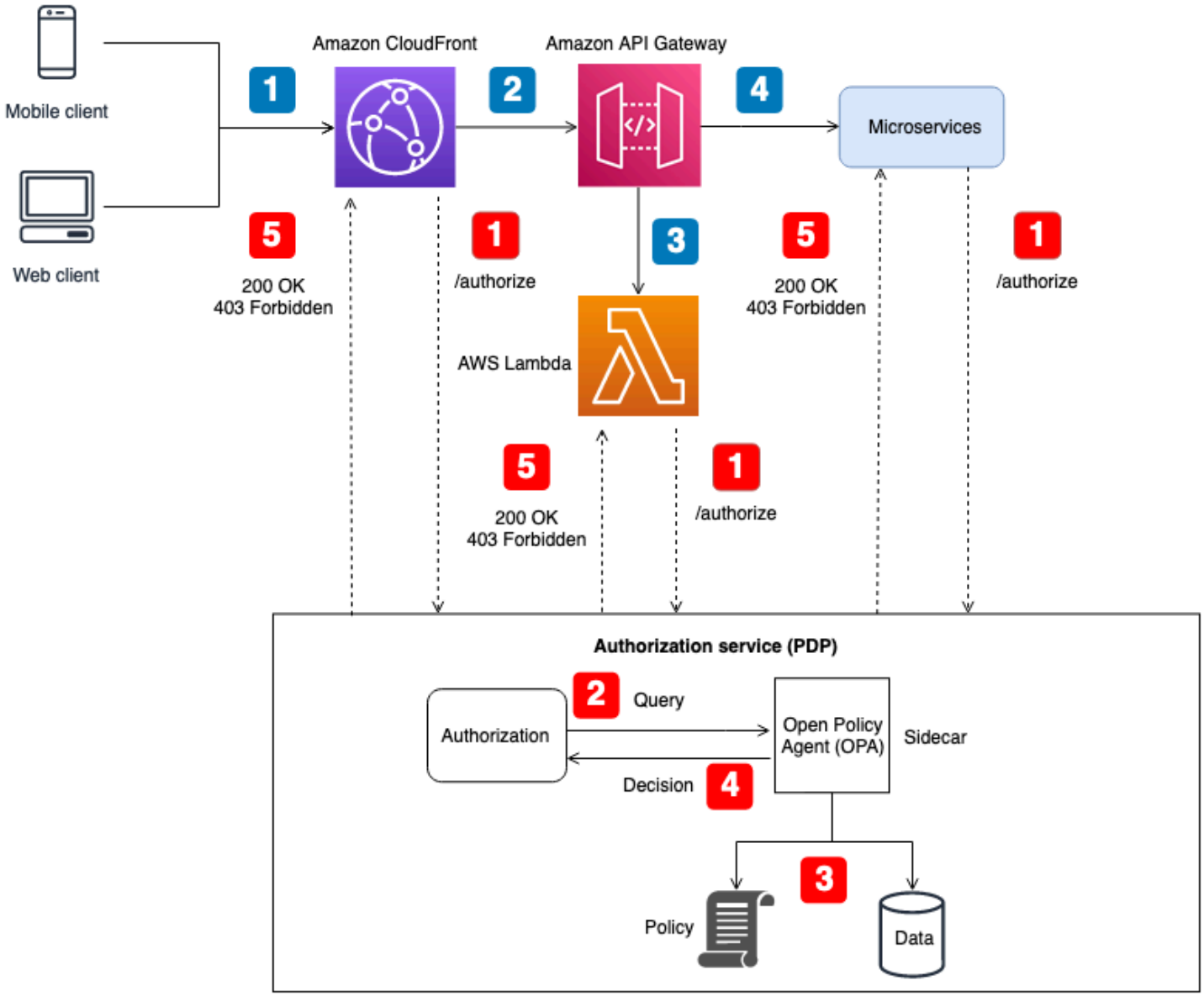
OPA の設計モデル

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、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管理できるシステムを作成するための業界のベストプラクティスに従います。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 を持つ認証済みユーザーは、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の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 間の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、承認の決定には別の 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に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

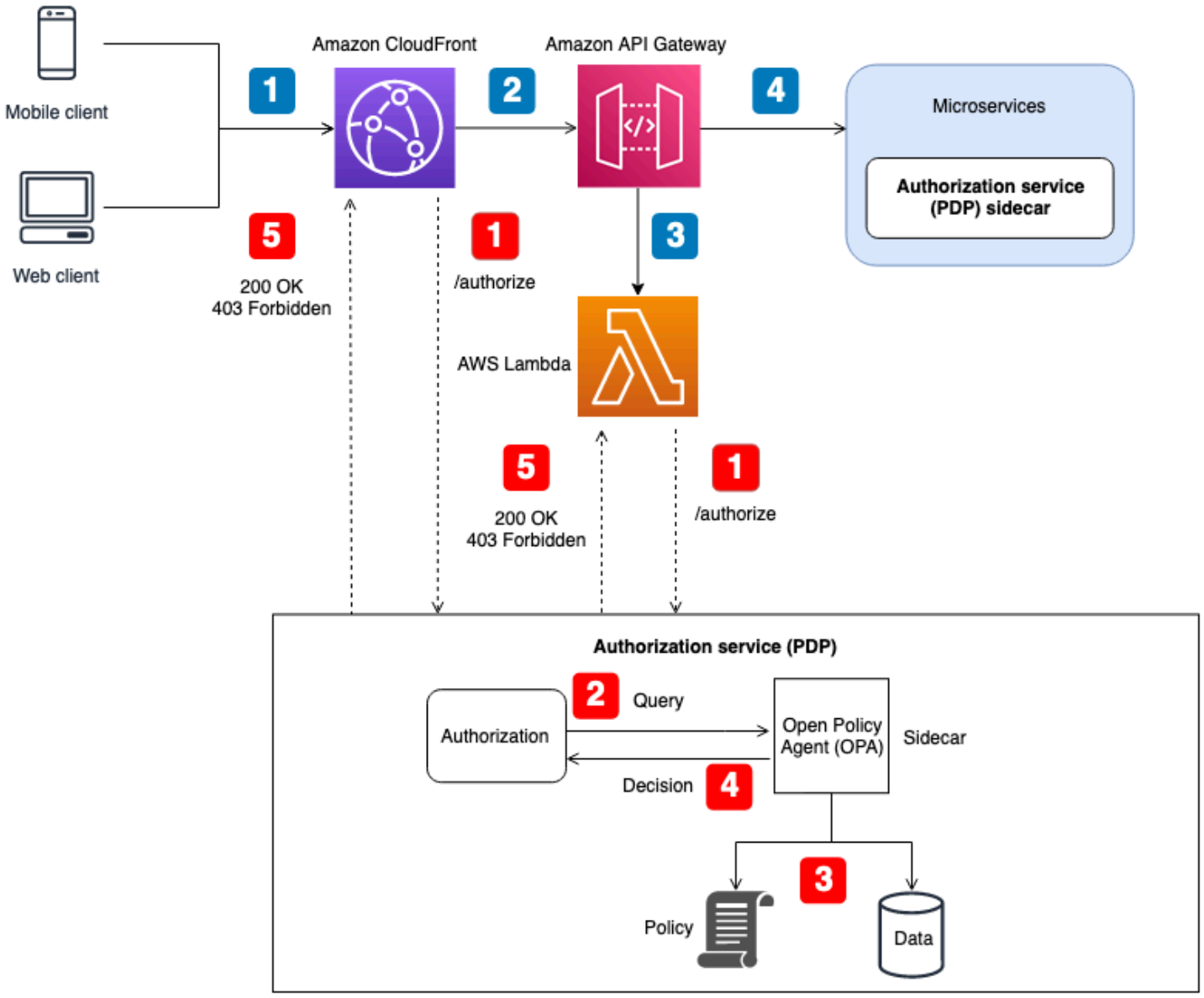
APIsモデルのポリシー適用ポイント (PEPs) を使用した分散ポリシー決定ポイント (PDP) は、API アクセスコントロールと認可のための効果的なシステムを作成するための業界のベストプラクティスに従います。APIs 上の PEPs を使用した一元化された PDP モデルと同様に、このアプローチは次の主要な原則をサポートします。

- 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。

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に疑問があるかもしれません。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、同じ入力に基づいて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。

このアプローチでは、PDPsはアプリケーションの複数の場所に実装されます。サイドカーまたは 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、PDP の決定は、一元化された PDP サービスに RESTful API コールを行うことなく、アプリケーションコンポーネントに直接統合できます。これにより、すべてのアプリケーションコンポーネントが認可の決定を得るために追加の 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ことができます。ただし、このモデルでは、Amazon CloudFront や Amazon API Gateway サービスなど、PDP を直接統合でき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 を持つ認証済みユーザーは、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の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の決定には、引き続き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 間の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。

APIs 上の PEPs を備えた分散 PDP には、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。

ライブラリとしての分散 PDP の使用

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、ユーザーはテナントにアクセスするためのアクセス許可を持つようになります。

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 の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 ジャーニーを経る組織やマルチテナントソリューションを実装する組織は、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頼るかもしれませんが、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に選択した分離モデルと同じではない場合があります。

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 ID の概念に従い、ユーザー ID をテナント ID 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。バインドには、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することが含まれます。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します。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`、`ps-43214321`承認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

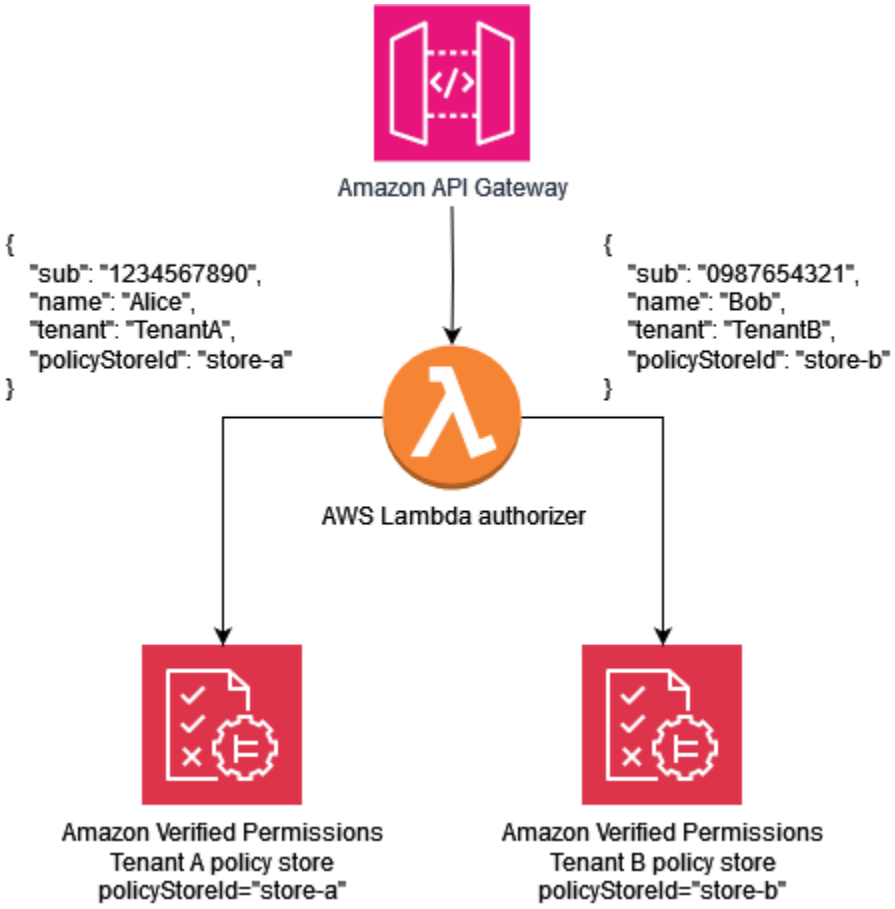
Amazon Verified Permissions のテナントごとのポリシーストア設計モデルは、SaaS アプリケーションの各テナントを独自のポリシーストアに関連付けます。このモデルは、SaaS サイロ分離モデルに似ています。どちらのモデルもテナント固有のインフラストラクチャの作成を義務付けており、同様の利点と欠点があります。このアプローチの主な利点は、インフラストラクチャが強制するテ

ナントの分離、テナントごとの一意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の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非常にサイロ化されたアプローチを提供できます。このモデルをプール分離で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ための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り、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 を 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して、リクエストを正しいポリシーストアにルーティングする方法を示します。



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 に固有です。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
);
```

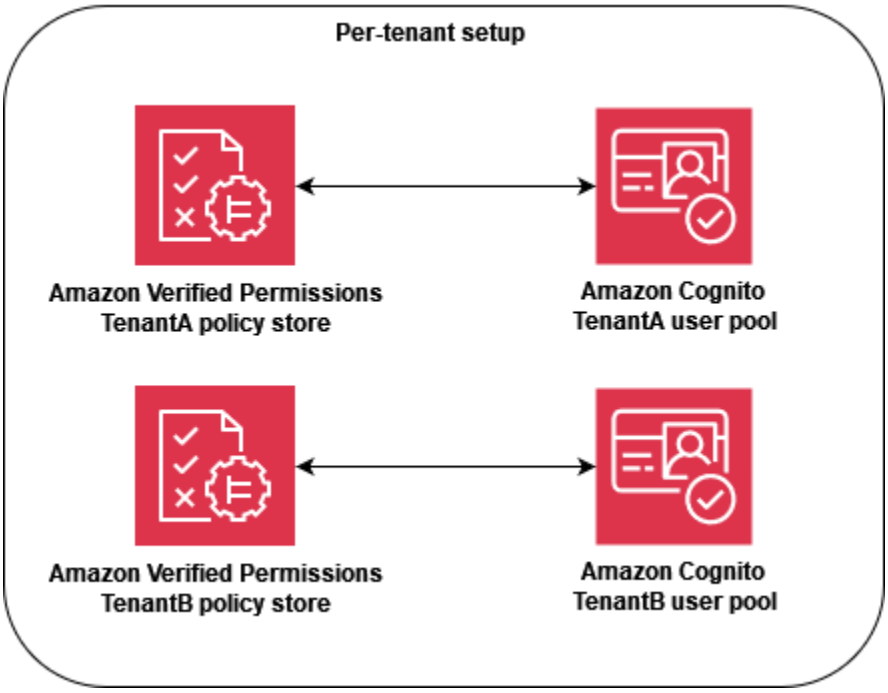
認可リクエストを行い、Verified Permissions ポリシーを使用して評価を開始するには、テナント にマッピングされた一意の ID に対応するポリシーストア ID を指定する必要があります`store-b`。

```
{
   "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、ポリシーはアイデンティティ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 と統合することを選択した場合、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。



オペレーションレベルでは、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関する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。

また、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より詳細な例については、 AWS ブログ記事SaaS 「Amazon Verified Permissions with a per-tenant policy store」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

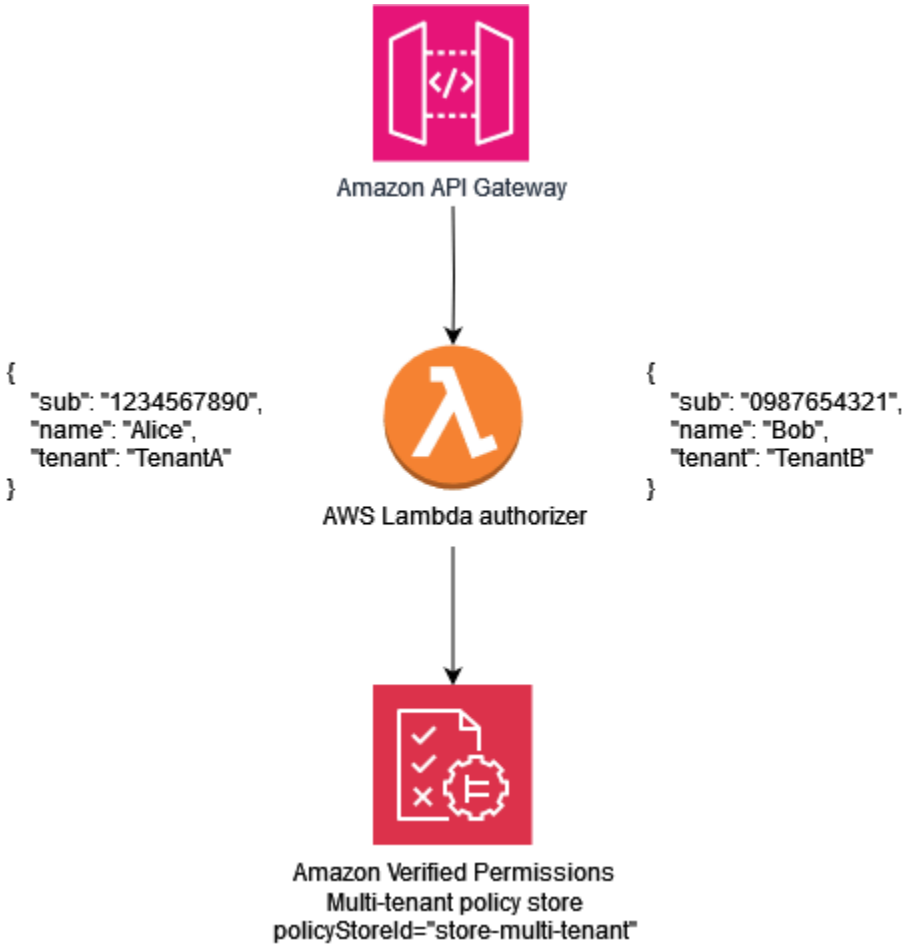
1 つの共有マルチテナントポリシーストア設計モデルは、SaaS ソリューションのすべてのテナントに対して Amazon Verified Permissions の 1 つのマルチテナントポリシーストアを使用します。このアプローチの主な利点は、管理と運用を簡素化することです。特に、テナントのオンボーディング中に追加のポリシーストアを作成する必要がないためです。このアプローチの欠点には、ポリシーの更新やデプロイの失敗やミスによる影響範囲の拡大、ノイズの多い近隣への影響の増大が含まれます。さらに、ソリューションでテナントごとに一意のポリシーが必要な場合は、このアプローチはお勧め

しません。この場合、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て、正しいテナントのポリシーが使用されていることを確認します。

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された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 1 つ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
};
```

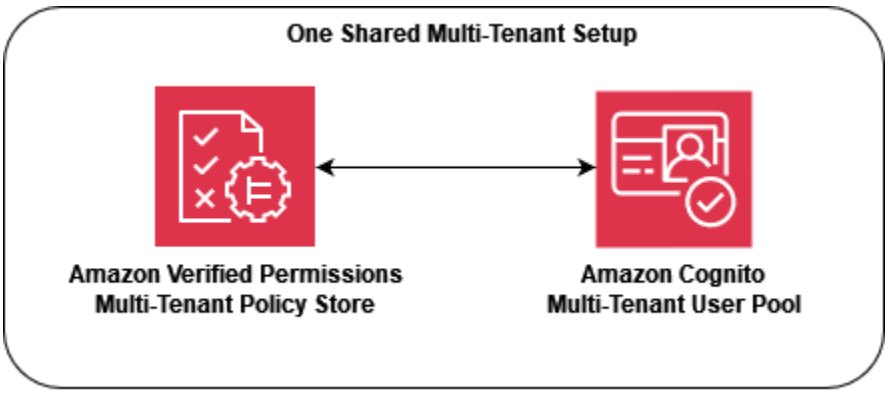
1 つの共有マルチテナントポリシーストアモデルを使用する認可リクエストの場合、ポリシーストア ID は共有ポリシーストアの識別子です。次のリクエストでは、 UserAliceには Roleの がありAdmin、リソースとプリンシパルに関連付けられたTenant属性は両方とも であるため、 へのアクセスが許可されますTenantA。

```
{
   "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、ポリシーはアイデンティティ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 と統合することを選択した場合、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、 でログに記録されたアクティビティ AWS CloudTrail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点です。これは、ログに記録された各 CloudTrail 呼び出しが同じポリシーストアを使用するためです。このシナリオでは、テナント単位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、ポリシーストアは通常、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、階層ごとに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。

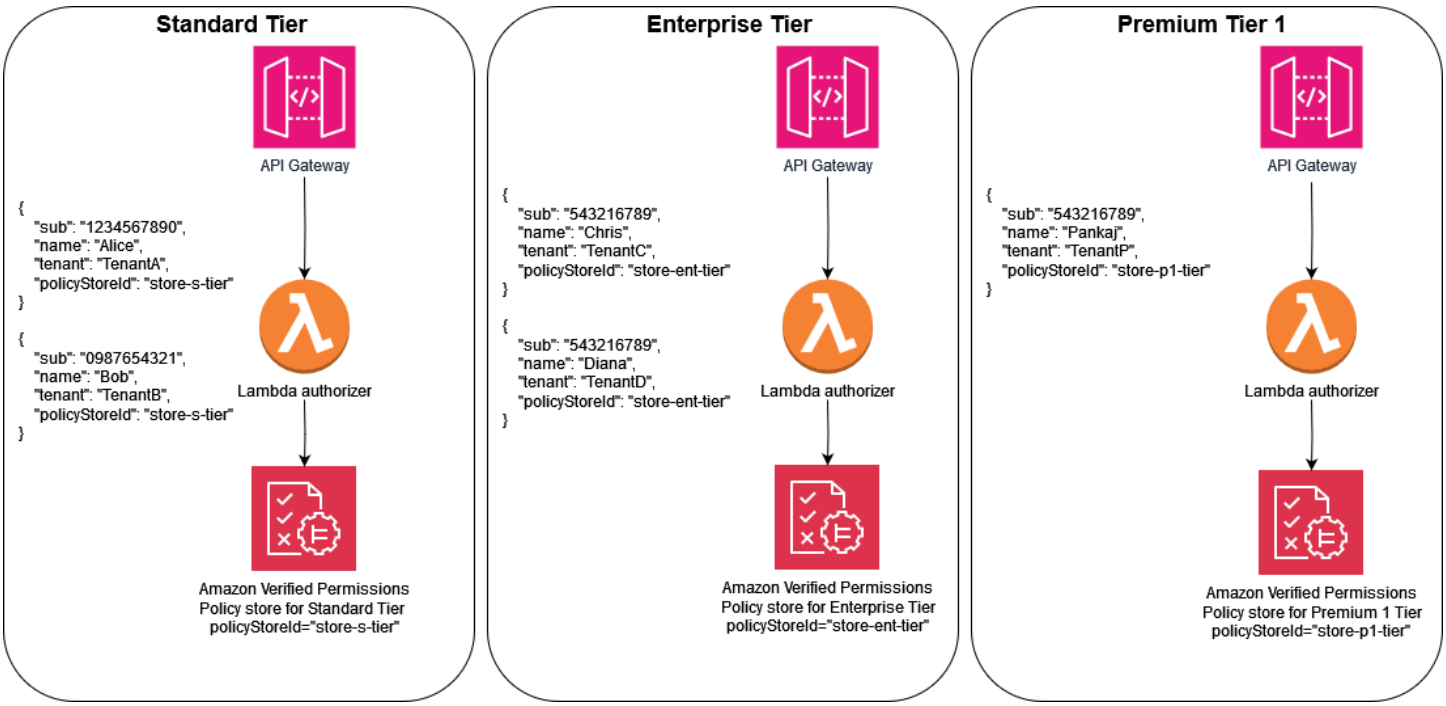
ソリューションがプールされた分離モデルを主に使用する場合は、追加の分離またはカスタマイズが必要になる場合があります。例えば、「プレミアム階層」を作成して、各テナントが独自のテナント階層インフラストラクチャを取得できます。これにより、テナントが 1 つしかないプールされたインスタンスをデプロイすることでサイロ化されたモデルが作成されます。これは、ポリシーストアを含む完全に分離された「プレミアム階層テナント A」インフラストラクチャと「プレミアム階層テナント B」インフラストラクチャの形式になる可能性があります。このアプローチにより、最高レベルの顧客に対してサイロ化された分離モデルになります。

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 、 Standard TierTenantC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、独自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。

このセクションの説明は 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。この一般的な分離の原則は、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 を使用した集中型 PDP APIs の使用PEPs APIs」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ために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 はコンテナまたはデーモンとしてデプロイされ、RESTful API は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 が呼び出されて、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です。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ないことです。OPA のデプロイは 1 つだけで

あるため、OPA が参照す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を OPA が利用できる必要があります。この方法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は、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、各認可の決定をローカルで行うことができるため、分散デプロイパターンのレイテンシーは集中型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場合は、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。DevOps と自動化によってこの複雑さの一部を管理できますが、一元化されたデプロイパターンと比較して、さらに多くの労力が必要になります。

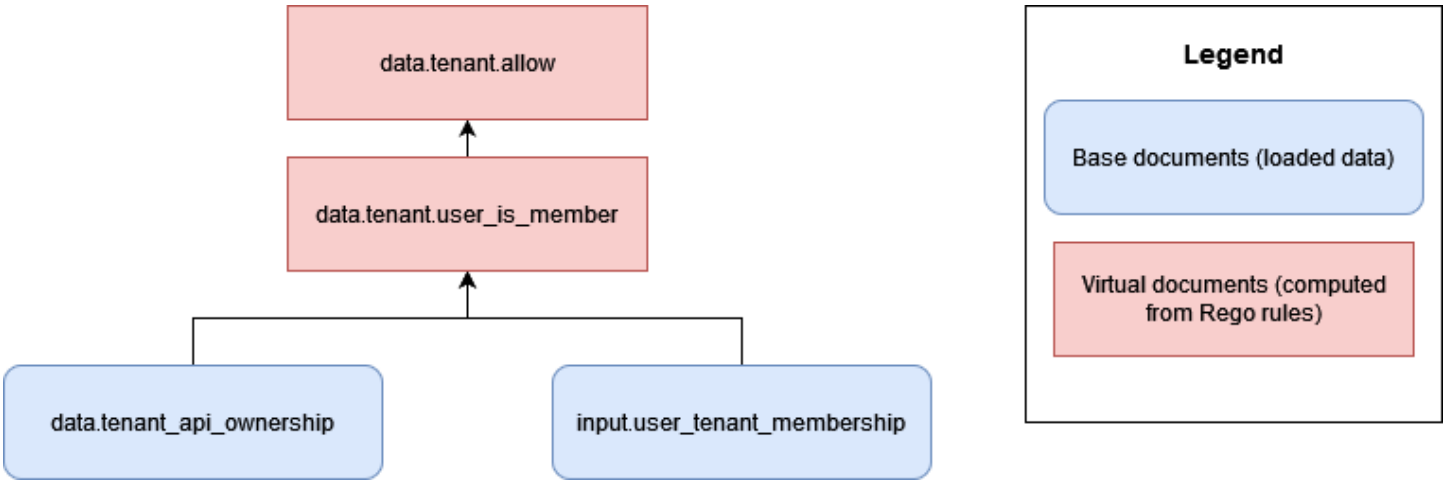
マルチテナントアプリケーションがサイロ化された SaaS モデルを使用している場合は、分散 OPA デプロイパターンを使用して、テナントデータ分離に対するサイロ化されたアプローチを模倣できます。これは、各テナント固有のアプリケーションサービスと一緒に OPA を実行する場合、各 OPA デプロイをカスタマイズして、そのテナントに関連付けられているデータのみを含めることができるためです。一元化された OPA デプロイパターンで OPA データをサイロ化することはできません。一元化されたデプロイパターンまたは分散パターンをプールされた SaaS モデルと組み合わせて使用する場合、テナントデータの分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

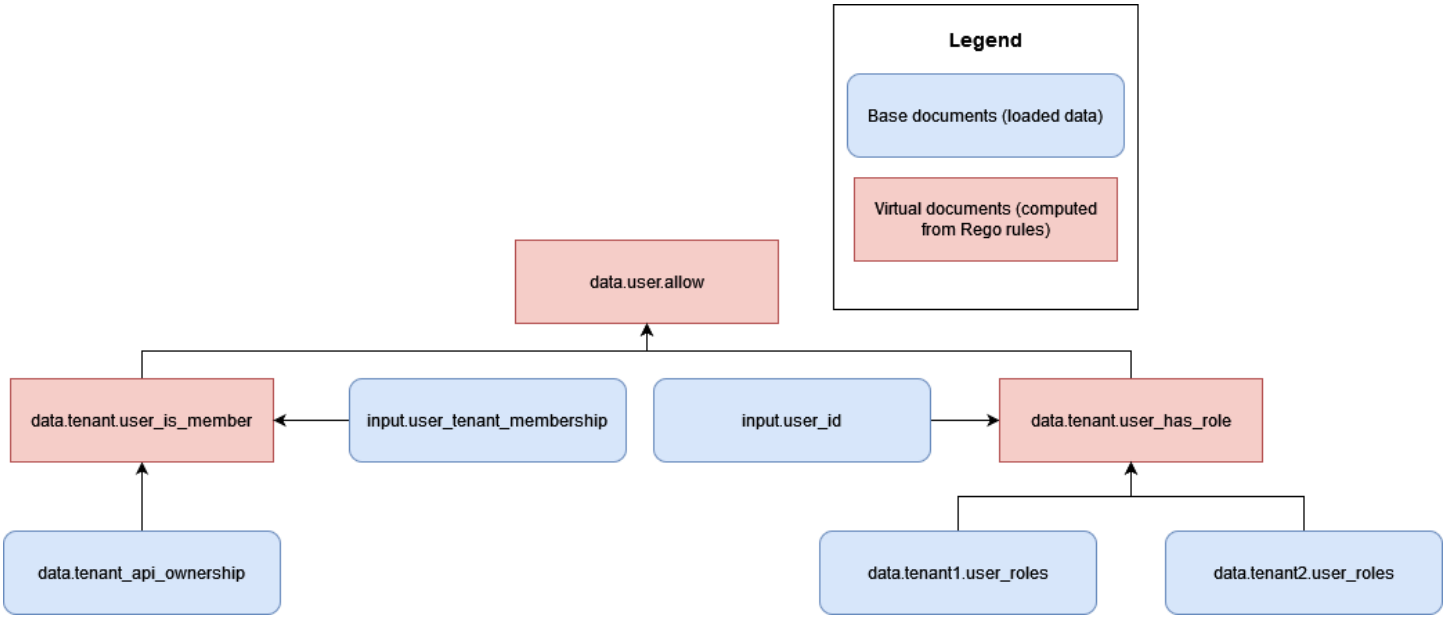
ントには、外部からのデータが含まれています。これには、OPA に直接提供されるデータ、OPA リクエストに関するデータ、および入力として OPA に渡される可能性のあるデータが含まれます。仮想ドキュメントはポリシーによって計算され、OPA ポリシーとルールが含まれます。詳細については、OPA ドキュメントを参照してください。

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 でテナント固有のデータを決定する必要はありません。次の例は、入力ドキュメントに示されているように、API を所有しているテナントと、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、どのテナントが API を所有しているかに関する情報を除き、OPA は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は懸念はありません。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。これは、各テナントが独立したリソースを所有するためです。

ただし、多くの RBAC 認可アプローチでは、クロステナントで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、承認の決定を行うために、と 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 にアクセスできるようにするためdata.tenant2.user_roles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、認可ロジックをリジッドにし、継続的な更新も必要になります。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。

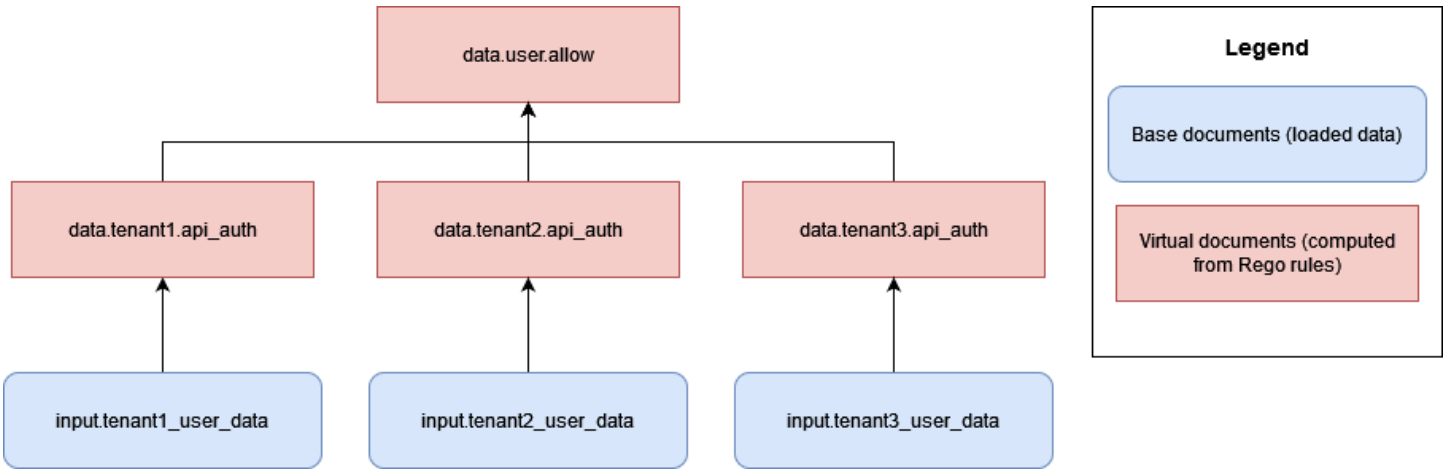
テナント固有のデータを非同期ベースドキュメントとして保存しないことで、OPA でのテナント分離を簡単に維持できます。非同期ベースドキュメントは、メモリに保存され、OPA で定期的に更新できるデータです。OPA 入力などの他の基本ドキュメントは同期的に渡され、決定時にのみ使用できます。例えば、クエリへの OPA 入力の一部としてテナント固有のデータを提供することは、テナント分離の違反にはなりません。これは、そのデータが決定プロセス中に同期的にしか利用できないためです。

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単に行えるようにする必要があります。パッケージを使用して OPA ドキュメントモデル階層に仮想ドキュメントを整理でき、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

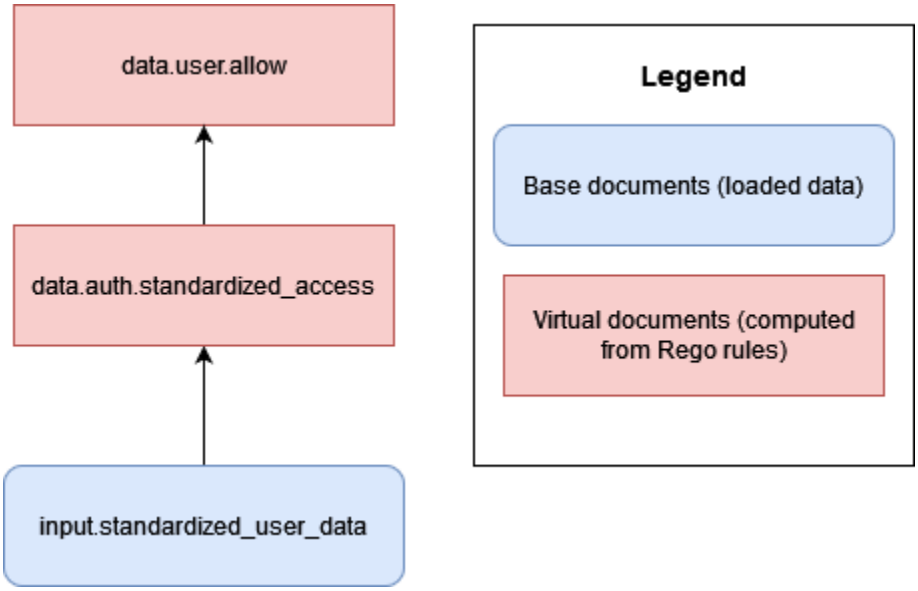
ナントアプリケーションの OPA ドキュメントモデルを計画するときは、まず OPA が決定を行うために必要なデータを決定します。データを入力として提供したり、OPA に事前ロードしたり、決定時または定期的に外部データソースから提供したりできます。OPA で外部データを使用する方法の詳細については、このガイドの後半にある「OpA での PDP の外部データの取得」セクションを参照してください。

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 ルールを実装して、そのデータで決定を行う方法を検討してください。例えば、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、テナント固有のルールの OPA パッケージを実装できます。



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。これは煩雑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承認を決定し、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容易にすることができます。



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。このアプローチでは、OPA がルールを通じてどのように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。テナントrules-per-tenantモデルを導入する必要があるのは、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のみです。

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の認可ロジックの削除を達成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迅速に対応できるように 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、 を通じて 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 は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新しいバージョンのポリシーバンドルをプルしたり、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 オペレータによって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、ポリシーのバージョニング、検証、更新のための堅牢な 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。

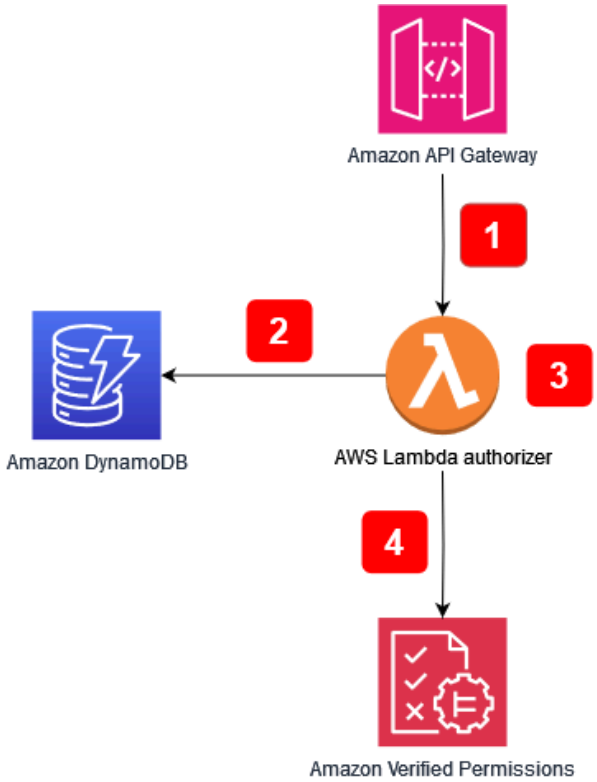
トピック

- 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
- テナントのデータの分離と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 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。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 に追加の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し、これらのサービスに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、データを取得したりする必要がある場合があります。



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て、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、認可の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、オーバーロード入力アプローチと呼ばれる任意の 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、プッシュデータ (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、外部データ取得の以下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る場合、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、外部データ取得の次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。

バンドルアプローチのこの代替方法では、データは定期的に OPA によって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にはバンドルアプローチと同じデータサイズの制限があります。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代わりに、OPA のデータを差分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。

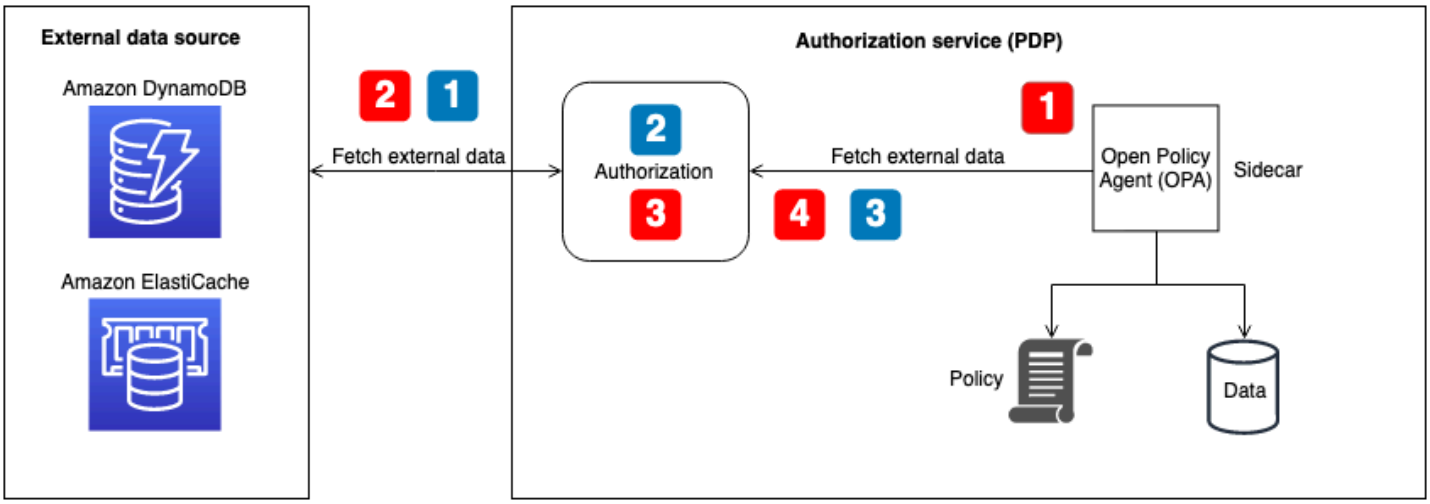
OPA 動的データ取得

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 に提供するためにカスタム 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、データの取得速度は極めて重要です。そのため、Amazon DynamoDB AWS のサービス などの を使用して、可能な場合は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 (図では赤の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。 バンドルの更新により、データとポリシーの両方が OPA のキャッシュですべて置き換えられます)。
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、それを予想される 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、設定された頻度で実行することも、外部ソースのデータ更新に応答して実行することもできます。
2. 認可サービスは、返されたデータで必要な変換を実行して、期待される 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、ルールを選択的に更新できます。

テナントのデータの分離と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、承認の決定を行うのに役立ついくつかのアプローチについて説明しました。可能な場合は、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代わりに、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、承認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、承認の決定を行うために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 からの 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、各テナントに論理的に分離された独自の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、認可ロジックをリジッドにし、継続的な更新も必要になります。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。これは、各テナントには論理的に分離された独自の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。このデータが認可の決定を行うために必要な場合は、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、認可の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。例えば、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 で提供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、認可の決定を行うため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。

アプリケーションに適した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変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 を使用すると、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 はソース管理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、更新をより簡単に OPA エージェントに分散できるため、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、集計用のログを収集し、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、 を通じ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。この作業により、Verified Permissions リソースに変更を加えると、手動作業がなくなり、オペレーターエラーが発生する可能性が低くなります。

承認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 ウェブトークン (JWT) に含まれるデータのみに基づいて承認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メソッドは、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します。
- 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: でオンプレミスの Oracle データベースを Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 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: カスタマーリレーションシップ管理 (CRM) システムを Salesforce.com に移行します。
- 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: オンプレミスの Oracle データベースを の EC2 インスタンス上の Oracle に移行します AWS クラウド。
- 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。

- 使用停止 — 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

ABAC

「属性ベースのアクセスコントロール」を参照してください。

抽象化されたサービス

「 マネージドサービス」を参照してください。

ACID

「アトミック性」、「整合性」、「分離」、「耐久性」を参照してください

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。より柔軟ですが、アクティブ/パッシブ移行よりも多くの作業が必要です。

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の方法。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。

集計関数

行のグループで動作し、グループの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 などがありますMAX。

AI

「人工知能」を参照してください。

AIOps

「人工知能オペレーション」を参照してください。

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。

アプリケーションコントロール

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 の需要要素であり、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?」を参照してください。

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

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)

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は、組織がクラウドに成功するための効率的で効果的な計画を立て AWS るのに役立ちます。 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整理します。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

個人または組織に損害を与えることを目的としたボット。

BCP

「事業継続計画」を参照してください。

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Data in a behavior graph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。エンディアン性も参照してください。

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

2 つの異なる同一の環境を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 (青) 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 (緑) 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、ボットハーダーまたはボットオペレーターとして知られる単一関係者の管理下にあるボットのネットワーク。ボットは、ボットとその影響をスケールするための最もよく知られているメカニズムです。

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なります。詳細については、 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 AWS 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、ホワイトペーパー AWSでのコンテナ化されたマイクロサービスの実行 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。

Canary デプロイ

エンドユーザーへのバージョンのスローリリースと増分リリース。確信が持てば、新しいバージョンをデプロイし、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

「データキャプチャの変更」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します。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。

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テクノロジー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 に移行するときに通常実行する 4 つのフェーズ AWS クラウド :

- 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。AWS 移行戦略との関連性については、「移行準備ガイド」を参照してください。

CMDB

「設定管理データベース」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または が含まれますBitbucket Cloud。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などのビジュアル形式から情報を分析および抽出する AI の分野。例えば、Amazon SageMaker AI は CV 用の画像処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定状態から変化します。これにより、ワークロードが非準拠になる可能性があり、通常は段階的かつ意図的ではありません。

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

一元管理とガバナンスを備えた分散型の分散データ所有権を提供す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。信頼された ID のみが、期待されるネットワークから信頼された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、通常、大量の履歴データが含まれており、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る。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。

ディープラーニング

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。

デプロイ

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、Implementing security controls on AWSのDetective controls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

スタースキーマでは、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する

離散数値です。これらの属性は、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一般的に使用されます。

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)

災害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に使用する戦略とプロセス。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。例えば、 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

「開発値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、応答時間を短縮できます。

電子データ交換 (EDI)

組織間の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プロセス。

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

星スキーマの中央テーブル。事業運営に関する量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 つのタイプの列が含まれます。

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で段階的なテストを使用する哲学。これはアジャイルアプローチの重要な部分です。

障害分離の境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を高めるのに役立つアベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界。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

同様のタスクの実行を求める前に、タスクと必要な出力を示す少数の例を LLM に提供します。この手法は、プロンプトに埋め込まれた例 (ショット) からモデルが学習するコンテキスト内学習のアプリケーションです。少数ショットプロンプトは、特定のフォーマット、推論、またはドメインの知識を必要とする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コントロール」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

段階的なアプローチを使用する代わりに、変更データキャプチャによる継続的なデータレプリケーションを使用して、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)

一般化およびラベル付けされていないデータの大規模なデータセットでトレーニングされている大規模な深層学習ニューラルネットワーク。FMs は、言語の理解、テキストと画像の生成、自然言語の会話など、さまざまな一般的なタスクを実行できます。詳細については、「基盤モデルとは」を参照してください。

G

生成 AI

大量のデータでトレーニングされ、シンプルなテキストプロンプトを使用してイメージ、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る AI モデルのサブセット。詳細については、「生成 AI とは」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 ワークフローはレガシーと見なされ、トランクベースのワークフローはモダンで推奨されるアプローチです。

ゴールデンイメージ

そのシステムまたはソフトウェアの新しいインスタンスをデプロイするためのテンプレートとして使用されるシステムまたはソフトウェアのスナップショット。例えば、製造では、ゴールデンイメージを使用して複数のデバイスにソフトウェアをプロビジョニングし、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、、 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。

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。

ハイアベイラビリティ (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に使用されるデータセットから保留される、ラベル付きの履歴データの一部。モデル予測をホールドアウトデータと比較することで、ホールドアウトデータを使用してモデルのパフォーマンスを評価できます。

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

「産業用モノのインターネット」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

既存のインフラストラクチャを更新、パッチ適用、または変更する代わりに、本番環境のワークロード用に新しいインフラストラクチャをデプロイするモデル。イミュータブルインフラストラクチャは、本質的に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、予測性が高くなります。詳細については、 AWS 「 Well-Architected フレームワーク」の「イミュータブルインフラストラクチャを使用したデプロイ」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 によって導入された用語で、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ゼーションを指します。

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。

産業分野における 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、「Building an industrial Internet of Things (IIoT) digital transformation strategy」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 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

「モノのインターネット」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)

大量のデータに対して事前トレーニングされた深層学習 AI モデル。LLM は、質問への回答、ドキュメントの要約、テキストの他の言語への翻訳、文の完了など、複数のタスクを実行できます。詳細については、LLMs」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

「ラベルベースのアクセスコントロール」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。エンディアン性も参照してください。

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または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、不正アクセスにつながる可能性があります。マルウェアの例としては、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 は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 AWS を運用し、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。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、マネージドサービスの例です。これら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、制御するためのソフトウェアシステムで、原材料を工場の完成製品に変換します。

MAP

「移行促進プログラム」を参照してください。

メカニズム

ツールを作成し、ツールの導入を推進し、調整を行うために結果を検査する完全なプロセス。メカニズムは、動作中にそれ自体を強化して改善するサイクルです。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。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、パブリッシュ/サブスクライブパターンに基づく軽量 machine-to-machine (M2M) 通信プロトコル。

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)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。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 に移行するために使用するアプローチ AWS クラウド。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、「」の「アプリケーションをモダナイズするための戦略 AWS クラウド」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、『』の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」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

「移行ポートフォリオ評価」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。

ミュータブルインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。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

「オリジンアクセスコントロール」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

「 オペレーションの統合」を参照してください。

OLA

「運用レベルの契約」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。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業用オートメーション用のmachine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

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)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問とそれに関連するベストプラクティスのチェックリスト。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するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、Industry 4.0 変換の主な焦点です。

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。

オリジンアクセスコントロール (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コントロールが可能です。

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

「プログラム可能な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

アクセス許可を定義 (アイデンティティベースのポリシーを参照)、アクセス条件を指定 (リソースベースのポリシーを参照)、または の組織内のすべてのアカウントに対する最大アクセス許可を定義 AWS Organizations （ サービスコントロールポリシーを参照) できるオブジェクト。

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。詳細については、マイクロサービスでのデータ永続性の有効化 を参照してください。

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、「移行準備状況ガイド」を参照してください。

述語

`true` または を返すクエリ条件。一般的に`false`は `WHERE`句にあります。

述語プッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、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、Implementing security controls on AWSのPreventative controls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内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じてプライバシーを考慮するシステムエンジニアリングアプローチ。

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を防ぐ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、プロビジョニング前にリソースをスキャンします。リソースがコントロールに準拠していない場合、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)

設計、開発、発売から成長と成熟まで、製品のデータとプロセスのライフサイクル全体にわたる管理。

本番環境

「環境」を参照してください。

プログラム可能なロジックコントローラー (PLC)

製造では、マシンをモニタリングし、製造プロセスを自動化する、信頼性の高い適応可能なコンピュータです。

プロンプトの連鎖

1 つの LLM プロンプトの出力を次のプロンプトの入力として使用して、より良いレスポンスを生成します。この手法は、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善または拡張したりするために使用されます。これにより、モデルのレスポンスの精度と関連性が向上し、より詳細でパーソナライズされた結果が得られます。

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。

パブリッシュ/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。スケーラビリティと応答性を向上させます。たとえば、マイクロサービスベースの MES では、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できるチャネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

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る手順などの一連のステップ。

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

責任、説明責任、相談、通知 (RACI) を参照してください。

RAG

「取得拡張生成」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

責任、説明責任、相談、情報 (RACI) を参照してください。

RCAC

「行と列のアクセスコントロール」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。

再設計

「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、AWS リージョン 「アカウントで使用できる を指定する」を参照してください。

回帰

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。

プラットフォーム変更

「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

中断に抵抗または回復するアプリケーションの機能。高可用性とディザスタリカバリは、 で回復性を計画する際の一般的な考慮事項です AWS クラウド。詳細については、AWS クラウド「レジリエンス」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。サポートを含めると、そのマトリックスは RASCI マトリックスと呼ばれ、サポートを除外すると 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、Implementing security controls on AWSのResponsive controls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。

取得拡張生成 (RAG)

LLM がレスポンスを生成する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を参照する生成 AI テクノロジー。たとえば、RAG モデルは、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する場合があります。詳細については、「RAG とは」を参照してください。

ローテーション

攻撃者が認証情報にアクセスすることをより困難にするために、シークレットを定期的に更新するプロセス。

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、 AWS Management Console にログインしたり 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

「監視コントロールとデータ取得」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。シークレット値は、バイナリ、単一の文字列、または複数の文字列にすることができます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」を参照してください。

設計によるセキュリティ

開発プロセス全体でセキュリティを考慮するシステムエンジニアリングアプローチ。

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、予防的、検出的、応答的、プロアクティブの 4 つの主なタイプがあります。

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に自動的に応答または修復するように設計された、事前定義された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。自動応答アクションの例としては、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

送信先にあるデータの、それ AWS のサービス を受け取る による暗号化。

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、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータによって測定される、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。

SIEM

セキュリティ情報とイベント管理システムを参照してください。

単一障害点 (SPOF)

システムを中断する可能性のあるアプリケーションの 1 つの重要なコンポーネントの障害。

SLA

「サービスレベルの契約」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、『』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 AWS クラウド」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

1 つの大きなファクトテーブルを使用してトランザクションデータまたは測定データを保存し、1 つ以上の小さなディメンションテーブルを使用してデータ属性を保存するデータベース組織構造。この構造は、データウェアハウスまたはビジネスインテリジェンスの目的で使用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。

監視制御とデータ収集 (SCADA)

製造では、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稼働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潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。Amazon CloudWatch Synthetics を使用して、これらのテストを作成できます。

システムプロンプト

LLM にコンテキスト、指示、またはガイドラインを提供して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、ユーザーとのやり取りのルールを確立するのに役立ちます。

T

tags

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。

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 枚のピザで養う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。詳細については、深層学習システムにおける不確実性の定量化 ガイドを参照してください。

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する行のグループに対して計算を実行する SQL 関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどの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 「ワークロード認定フレームワーク」を参照してください。

Write Once, Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。承認されたユーザーは、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

ゼロデイ脆弱性を利用する攻撃、通常はマルウェア。

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 にタスクを実行する手順を提供しますが、タスクのガイドに役立つ例 (ショット) はありません。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。