멀티테넌트 SaaS 인증 및 API 액세스 제어: 구현 옵션 및 모범 사례

AWS 규범적 지침

AWS 규범적 지침: 멀티테넌트 SaaS 인증 및 API 액세스 제어: 구현 옵션 및 모범 사례

Table of Contents

멀티테넌트 SaaS 인증 및 API 액세스 제어: 구현 옵션 및 모범 사례

태비 워드, 토마스 데이비스, 기디언 랜드먼, 토마스 리하, Amazon Web Services ()AWS

2024년 5월 (문서 기록)

인증 및 API 액세스 제어는 많은 소프트웨어 애플리케이션, 특히 멀티테넌트 SaaS (Software as a Service) 애플리케이션에서 어려운 과제입니다. 보안이 필요한 마이크로서비스 API의 급증과 다양한 테넌트, 사용자 특성 및 애플리케이션 상태에서 발생하는 수많은 액세스 조건을 고려하면 이러한 복잡성이 확연히 드러납니다. 이러한 문제를 효과적으로 해결하려면 솔루션은 마이크로서비스, BFF (Backend for Fronend) 계층 및 멀티테넌트 SaaS 애플리케이션의 기타 구성 요소에서 제공하는 여러 API에 대한 액세스 제어를 적용해야 합니다. 이 접근 방식에는 여러 요인과 속성을 기반으로 복잡한 액세스 결정을 내릴 수 있는 메커니즘이 수반되어야 합니다.

기존에는 API 액세스 제어 및 권한 부여가 애플리케이션 코드의 사용자 지정 로직으로 처리되었습니다. 이 접근 방식은 오류가 발생하기 쉽고 안전하지 않았습니다. 이 코드에 액세스할 수 있는 개발자가 실수로 또는 의도적으로 권한 부여 논리를 변경하여 무단 액세스가 발생할 수 있기 때문입니다. 애플리케이션 코드에서 사용자 지정 로직으로 내린 결정을 감사하는 것은 어려웠습니다. 감사자가 특정 표준을 준수하는 데 있어 사용자 지정 로직이 효과적인지 판단하기 위해 사용자 지정 로직에 몰두해야 했기 때문입니다. 게다가 보호해야 할 API가 많지 않았기 때문에 일반적으로 API 액세스 제어는 불필요했습니다. 마이크로서비스와 서비스 지향 아키텍처를 선호하도록 애플리케이션 설계의 패러다임이 바뀌면서 권한 부여 및 액세스 제어 형식을 사용해야 하는 API의 수가 증가했습니다. 또한 멀티테넌트 SaaS 애플리케이션에서 테넌트 기반 액세스를 유지해야 하므로 테넌시를 유지하기 위한 권한 부여 문제가 추가로 발생합니다. 이 가이드에 설명된 모범 사례는 다음과 같은 몇 가지 이점을 제공합니다.

- 인증 로직을 중앙 집중화하여 프로그래밍 언어에 국한되지 않는 고급 선언적 언어로 작성할 수 있습니다.
- 권한 부여 로직은 애플리케이션 코드에서 추출되며 애플리케이션의 모든 API에 반복 가능한 패턴으로 적용할 수 있습니다.
- 추상화를 통해 개발자가 권한 부여 논리를 실수로 변경하는 것을 방지할 수 있습니다.
- SaaS 애플리케이션으로의 통합은 일관되고 간단합니다.
- 추상화를 통해 각 API 엔드포인트에 대한 사용자 지정 권한 부여 로직을 작성할 필요가 없습니다.
- 감사자가 더 이상 코드를 검토하여 권한을 결정할 필요가 없으므로 감사가 간소화됩니다.
- 이 가이드에 설명된 접근 방식은 조직의 요구 사항에 따라 여러 액세스 제어 패러다임을 사용할 수 있도록 지원합니다.

- 이 권한 부여 및 액세스 제어 접근 방식은 SaaS 애플리케이션의 API 계층에서 테넌트 데이터 격리를 유지하는 간단하고 직접적인 방법을 제공합니다.
- 모범 사례는 권한 부여와 관련하여 테넌트를 온보딩하고 오프보딩하는 일관된 접근 방식을 제공합니다.
- 이 접근 방식은 이 가이드에서 설명하는 것처럼 장단점이 모두 있는 다양한 권한 부여 배포 모델 (풀링된 모델 또는 사일로) 을 제공합니다.

목표 비즈니스 성과

이 규범적 지침은 멀티테넌트 SaaS 애플리케이션에 구현할 수 있는 권한 부여 및 API 액세스 제어를 위한 반복 가능한 설계 패턴을 설명합니다. 이 지침은 복잡한 권한 요구 사항이나 엄격한 API 액세스 제어 요구 사항이 있는 애플리케이션을 개발하는 모든 팀을 대상으로 합니다. 이 아키텍처에서는 정책 결정 지점 (PDP) 또는 정책 엔진의 생성과 API에 정책 적용 지점 (PEP) 을 통합하는 방법을 자세히 설명합니다. PDP 생성을 위한 두 가지 구체적인 옵션에 대해 설명합니다. 하나는 Cedar SDK와 함께 Amazon 검증 권한을 사용하는 것이고 다른 하나는 Rego 정책 언어와 함께 공개 정책 에이전트 (OPA) 를 사용하는 것입니다. 또한 이 가이드에서는 속성 기반 액세스 제어 (ABAC) 모델이나 역할 기반 액세스 제어 (RBAC) 모델 또는 두 모델의 조합을 기반으로 액세스 결정을 내리는 방법에 대해서도 설명합니다. 이 가이드에 제공된 설계 패턴과 개념을 사용하여 멀티테넌트 SaaS 애플리케이션의 권한 부여 및 API 액세스 제어 구현을 알리고 표준화하는 것이 좋습니다. 이 지침은 다음과 같은 비즈니스 성과를 달성하는 데 도움이 됩니다.

- 멀티테넌트 SaaS 애플리케이션을 위한 표준화된 API 권한 부여 아키텍처 — 이 아키텍처는 정책을 저장하고 관리하는 PAP (정책 관리 지점), 권한 부여 결정에 도달하기 위해 해당 정책을 평가하는 PDP (정책 결정 지점), 해당 결정을 시행하는 정책 적용 지점 (PEP) 이라는 세 가지 구성 요소를 구분합니다. 호스팅된 권한 부여 서비스인 검증된 권한은 PAP와 PDP 역할을 모두 수행합니다. 또는 Cedar 또는 OPA와 같은 오픈 소스 엔진을 사용하여 PDP를 직접 구축할 수도 있습니다.
- 응용 프로그램에서 권한 부여 로직 분리 - 권한 부여 논리를 응용 프로그램 코드에 포함하거나 임시 적용 메커니즘을 통해 구현한 경우 의도하지 않은 테넌트 간 데이터 액세스 또는 기타 보안 침해를 유발하는 우발적 또는 악의적인 변경이 발생할 수 있습니다. 이러한 가능성을 줄이려면 Verified Permissions와 같은 PAP를 사용하여 권한 부여 정책을 애플리케이션 코드와 독립적으로 저장하고 이러한 정책 관리에 강력한 거버넌스를 적용할 수 있습니다. 정책을 높은 수준의 선언적 언어로 중앙에서 관리할 수 있으므로 애플리케이션 코드의 여러 섹션에 정책을 포함할 때보다 권한 부여 논리를 훨씬 간단하게 관리할 수 있습니다. 또한 이 접근 방식을 통해 업데이트가 일관되게 적용됩니다.
- 액세스 제어 모델에 대한 유연한 접근 방식, 즉 역할 기반 액세스 제어 (RBAC), 속성 기반 액세스 제어 (ABAC) 또는 두 모델의 조합 등 모두 유효한 액세스 제어 접근 방식입니다. 이러한 모델은 다양한 접근 방식을 사용하여 비즈니스에 대한 인증 요구 사항을 충족하려고 합니다. 이 가이드에서는 조직

에 적합한 모델을 선택하는 데 도움이 되도록 이러한 모델을 비교하고 대조합니다. 또한 이 가이드에서는 이러한 모델이 OPA/rego 및 Cedar와 같은 다양한 권한 부여 정책 언어에 어떻게 적용되는지도 설명합니다. 이 가이드에서 설명하는 아키텍처를 통해 두 모델 중 하나 또는 두 모델을 모두 성공적으로 채택할 수 있습니다.

- 엄격한 API 액세스 제어 — 이 가이드에서는 최소한의 노력으로 애플리케이션에서 API를 일관되고 광범위하게 보호하는 방법을 제공합니다. 이는 애플리케이션 내 통신을 촉진하기 위해 일반적으로 많은 수의 API를 사용하는 서비스 지향 또는 마이크로서비스 애플리케이션 아키텍처에 특히 유용합니다. 엄격한 API 액세스 제어는 애플리케이션의 보안을 강화하고 공격이나 악용에 덜 취약하게 만듭니다.

테넌트 격리 및 멀티 테넌트 권한 부여

이 가이드에서는 테넌트 격리 및 멀티 테넌트 권한 부여의 개념을 설명합니다. 테넌트 격리란 SaaS 시스템에서 사용하는 명시적 메커니즘을 의미하며, 공유 인프라에서 운영되는 경우에도 각 테넌트의 리소스가 격리되도록 합니다. 멀티 테넌트 인증이란 인바운드 작업을 승인하고 잘못된 테넌트에서 구현되지 않도록 하는 것을 의미합니다. 가상의 사용자가 인증되고 권한을 부여받은 후에도 다른 테넌트의 리소스에 계속 액세스할 수 있습니다. 인증 및 권한 부여는 이러한 액세스를 차단하지 않으므로 이 목표를 달성하려면 테넌트 격리를 구현해야 합니다. 이 두 개념의 차이점에 대한 보다 광범위한 설명은 SaaS 아키텍처 기초 백서의 테넌트 격리 섹션을 참조하십시오.

액세스 제어 유형

광범위하게 정의된 두 가지 모델, 즉 역할 기반 액세스 제어 (RBAC) 와 속성 기반 액세스 제어 (ABAC) 를 사용하여 액세스 제어를 구현할 수 있습니다. 각 모델에는 장단점이 있으며, 이에 대해서는 이 섹션에서 간략하게 설명합니다. 사용해야 하는 모델은 특정 사용 사례에 따라 다릅니다. 이 가이드에서 설명하는 아키텍처는 두 모델을 모두 지원합니다.

RBAC

역할 기반 액세스 제어 (RBAC) 는 일반적으로 비즈니스 로직에 맞는 역할을 기반으로 리소스에 대한 액세스를 결정합니다. 권한은 역할에 따라 적절하게 연결됩니다. 예를 들어 마케팅 역할은 제한된 시스템 내에서 마케팅 활동을 수행할 수 있는 권한을 사용자에게 부여합니다. 이 액세스 제어 모델은 쉽게 알아볼 수 있는 비즈니스 로직에 잘 맞기 때문에 비교적 간단하게 구현할 수 있습니다.

RBAC 모델은 다음과 같은 경우 효과가 떨어집니다.

- 여러 역할을 담당하는 고유 사용자가 있습니다.
- 비즈니스 로직이 복잡하여 역할을 정의하기가 어렵습니다.
- 대규모로 확장하려면 지속적으로 관리하고 권한을 새 역할과 기존 역할에 매핑해야 합니다.
- 승인은 동적 매개변수를 기반으로 합니다.

ABAC

속성 기반 액세스 제어 (ABAC) 는 속성을 기반으로 리소스에 대한 액세스를 결정합니다. 속성은 사용자, 리소스, 환경 또는 애플리케이션 상태와 연관될 수 있습니다. 정책이나 규칙은 속성을 참조하며 기본 부울 논리를 사용하여 사용자의 작업 수행 허용 여부를 결정할 수 있습니다. 권한의 기본 예는 다음과 같습니다.

결제 시스템에서는 재무 부서의 모든 사용자가 업무 시간 */payments* 동안 API 엔드포인트에서 결제를 처리할 수 있습니다.

재무 부서의 멤버십은 액세스 권한을 결정하는 사용자 속성입니다/payments. 업무 시간에만 액세스를 허용하는 /payments API 엔드포인트와 연결된 리소스 속성도 있습니다. ABAC에서는 재무 부서 멤버십을 사용자 속성으로, 시간을 리소스 속성으로 포함하는 정책에 따라 사용자가 결제를 처리할 수 있는지 여부가 결정됩니다. /payments

ABAC 모델은 매우 유연하여 동적이고 상황에 따라 세부적인 권한 부여 결정을 내릴 수 있습니다. 그러나 ABAC 모델은 초기에 구현하기가 어렵습니다. 규칙과 정책을 정의하고 모든 관련 액세스 벡터의 속성을 열거하려면 상당한 사전 투자가 필요합니다.

RBAC-ABAC 하이브리드 접근 방식

RBAC와 ABAC를 함께 사용하면 두 모델의 장점을 일부 제공할 수 있습니다. RBAC는 비즈니스 로직과 매우 밀접하게 연계되므로 ABAC보다 구현하기가 더 간단합니다. ABAC를 RBAC와 결합하여 권한 부여 결정을 내릴 때 한층 더 세분화할 수 있습니다. 이 하이브리드 접근 방식은 사용자의 역할 (및 할당된 권한) 을 추가 속성과 결합하여 액세스 결정을 내리는 방식으로 액세스를 결정합니다. 두 모델을 모두 사용하면 권한을 간편하게 관리하고 할당할 수 있을 뿐만 아니라 권한 부여 결정과 관련된 유연성과 세분성을 높일 수 있습니다.

액세스 제어 모델 비교

다음 표에서는 앞서 설명한 세 가지 액세스 제어 모델을 비교합니다. 이 비교는 정보를 제공하고 높은 수준을 제공하기 위한 것입니다. 특정 상황에서 액세스 모델을 사용하는 것이 이 표에 나와 있는 비교와 반드시 상관관계가 있는 것은 아닙니다.

요인	RBAC	ABAC	하이브리드
유연성	중간	높음	높음
단순성	높음	낮음	중간
세부 수준	낮음	높음	중간
역동적인 결정과 규칙	아니요	예	예
상황 인식	아니요	예	어느 정도
구현 노력	낮음	높음	중간

PDP 구현하기

정책 결정 지점 (PDP) 은 정책 또는 규칙 엔진으로 특징지을 수 있습니다. 이 구성 요소는 정책 또는 규칙을 적용하고 특정 액세스의 허용 여부에 대한 결정을 반환하는 역할을 합니다. PDP는 RBAC (역할 기반 액세스 제어) 및 ABAC (속성 기반 액세스 제어) 모델과 함께 작동할 수 있지만 ABAC에는 PDP가 필요합니다. PDP를 사용하면 애플리케이션 코드의 인증 로직을 별도의 시스템으로 오프로드할 수 있습니다. 이렇게 하면 애플리케이션 코드를 단순화할 수 있습니다. 또한 API, 마이크로서비스, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내리는 데 사용할 수 있는 easy-to-use 반복 가능한 인터페이스를 제공합니다.

다음 섹션에서는 PDP를 구현하는 세 가지 방법을 설명합니다. 그러나 이 목록이 전체 목록은 아닙니다.

PDP 구현 방법:

- 아마존 검증 권한을 사용하여 PDP 구현
- OPA를 사용하여 PDP 구현
- 사용자 지정 정책 엔진 사용

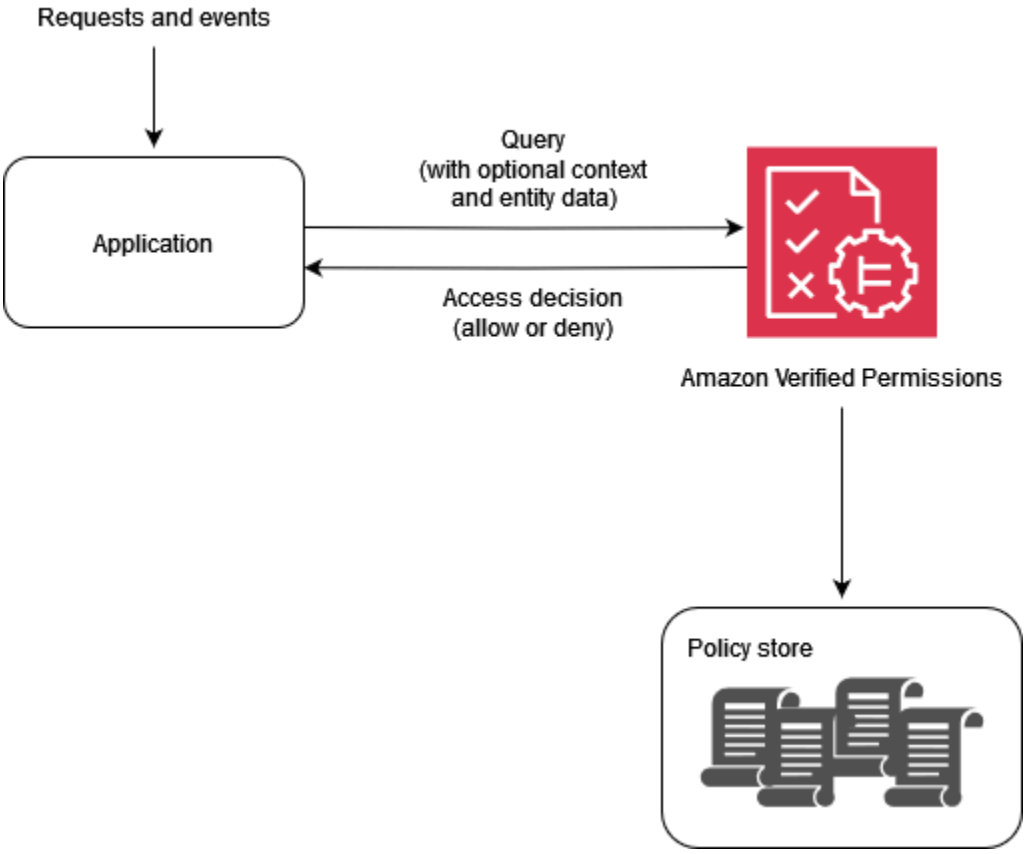
아마존 검증 권한을 사용하여 PDP 구현

Amazon Verified Permissions는 정책 결정 지점 (PDP) 을 구현하는 데 사용할 수 있는 확장 가능하고 세분화된 권한 관리 및 권한 부여 서비스입니다. 정책 엔진으로서 애플리케이션에서 사용자 작업을 실시간으로 확인하고 지나치게 권한이 있거나 유효하지 않은 권한을 강조 표시하는 데 도움이 될 수 있습니다. 권한 부여를 외부화하고 정책 관리 및 관리를 중앙 집중화하여 개발자가 보다 안전한 애플리케이션을 더 빠르게 구축할 수 있도록 지원합니다. Verified Permissions는 권한 부여 논리를 애플리케이션 로직과 분리함으로써 정책 디커플링을 지원합니다.

검증된 권한을 사용하여 PDP를 구현하고 애플리케이션 내에서 최소 권한 및 지속적인 검증을 구현함으로써 개발자는 애플리케이션 액세스를 제로 트러스트 원칙에 맞게 조정할 수 있습니다. 또한 보안 및 감사 팀은 애플리케이션 내에서 누가 어떤 리소스에 액세스할 수 있는지 더 잘 분석하고 감사할 수 있습니다. Verified Permissions는 특수 목적의 보안 우선 오픈 소스 정책 언어인 Cedar를 사용하여 RBAC (역할 기반 액세스 제어) 및 ABAC (속성 기반 액세스 제어) 를 기반으로 정책 기반 액세스 제어를 정의하여 상황에 맞는 보다 세분화된 액세스 제어를 가능하게 합니다.

검증된 권한은 Amazon Cognito, Google, Facebook과 같은 여러 자격 증명 공급자를 사용하여 멀티 테넌트 인증을 활성화하는 기능과 같이 SaaS 애플리케이션에 유용한 몇 가지 기능을 제공합니다. SaaS 애플리케이션에 특히 유용한 또 다른 검증된 권한 기능은 테넌트별 사용자 지정 역할을 지원하는 것입

니다. CRM (고객 관계 관리) 시스템을 설계하는 경우 한 테넌트가 특정 기준 세트를 기반으로 영업 기회별 액세스 세분성을 정의할 수 있습니다. 다른 테넌트에 다른 정의가 있을 수 있습니다. 검증된 권한의 기본 권한 시스템은 이러한 변형을 지원할 수 있으므로 SaaS 사용 사례에 적합합니다. 또한 Verified Permissions는 모든 테넌트에 적용되는 정책을 작성하는 기능을 지원하므로 SaaS 제공업체로서 무단 액세스를 방지하기 위한 가드레일 정책을 적용하는 것은 간단합니다.



검증된 권한을 사용하는 이유는 무엇입니까?

Amazon Cognito와 같은 ID 공급자와 함께 검증된 권한을 사용하면 애플리케이션을 위한 보다 동적인 정책 기반 액세스 관리 솔루션을 구축할 수 있습니다. 데이터의 보안, 기밀성 및 프라이버시를 유지하면서 사용자가 정보를 공유하고 협업하는 데 도움이 되는 애플리케이션을 구축할 수 있습니다. Verified Permissions는 ID 및 리소스의 역할과 특성에 따라 액세스를 강제할 수 있는 세분화된 권한 부여 시스템을 제공함으로써 운영 비용을 줄이는 데 도움이 됩니다. 정책 모델을 정의하고, 정책을 생성하여 중앙 위치에 저장하고, 액세스 요청을 밀리초 단위로 평가할 수 있습니다.

검증된 권한에서는 Cedar라는 간단하고 사람이 읽을 수 있는 선언적 언어를 사용하여 권한을 표현할 수 있습니다. Cedar로 작성된 정책은 각 팀의 애플리케이션에서 사용하는 프로그래밍 언어에 관계없이 팀 간에 공유할 수 있습니다.

검증된 권한을 사용할 때 고려할 사항

검증된 권한에서 정책을 생성하고 프로비저닝의 일환으로 정책을 자동화할 수 있습니다. 애플리케이션 로직의 일부로 런타임에 정책을 생성할 수도 있습니다. 가장 좋은 방법은 테넌트 온보딩 및 프로비저닝의 일부로 정책을 생성할 때 CI/CD (지속적 통합 및 지속적 배포) 파이프라인을 사용하여 정책 버전을 관리, 수정 및 추적하는 것입니다. 또는 애플리케이션에서 정책 버전을 관리, 수정 및 추적할 수도 있지만 애플리케이션 로직은 기본적으로 이 기능을 수행하지 않습니다. 애플리케이션에서 이러한 기능을 지원하려면 이 기능을 구현하도록 애플리케이션을 명시적으로 설계해야 합니다.

권한 부여 결정을 내리기 위해 다른 소스의 외부 데이터를 제공해야 하는 경우 권한 부여 요청의 일부로 이 데이터를 검색하여 Verified Permissions에 제공해야 합니다. 이 서비스에서는 기본적으로 추가 컨텍스트, 엔티티 및 속성을 검색하지 않습니다.

시더 개요

Cedar는 개발자가 애플리케이션 권한을 정책으로 표현할 수 있도록 지원하는 유연하고 확장 가능하며 확장 가능한 정책 기반 액세스 제어 언어입니다. 관리자와 개발자는 사용자가 애플리케이션 리소스에 대한 작업을 수행하도록 허용하거나 금지하는 정책을 정의할 수 있습니다. 단일 리소스에 여러 정책을 연결할 수 있습니다. 애플리케이션 사용자가 리소스에서 작업을 수행하려고 하면 애플리케이션은 Cedar 정책 엔진에 권한 부여를 요청합니다. Cedar는 적용 가능한 정책을 평가하고 OR 결정을 반환합니다`ALLOW`. `DENY` Cedar는 모든 유형의 주체 및 리소스에 대한 권한 부여 규칙을 지원하고, 역할 기반 액세스 제어 (RBAC) 및 속성 기반 액세스 제어 (ABAC) 를 허용하고, 자동화된 추론 도구를 통한 분석을 지원합니다.

Cedar를 사용하면 비즈니스 로직을 권한 부여 로직에서 분리할 수 있습니다. 애플리케이션 코드에서 요청하는 경우 Cedar의 권한 부여 엔진을 호출하여 요청이 승인되었는지 여부를 확인합니다. 승인된 경우 (결정에 따름`ALLOW`), 애플리케이션은 요청된 작업을 수행할 수 있습니다. 승인되지 않은 경우 (결정에 따름`DENY`), 애플리케이션은 오류 메시지를 반환할 수 있습니다. Cedar의 주요 기능은 다음과 같습니다.

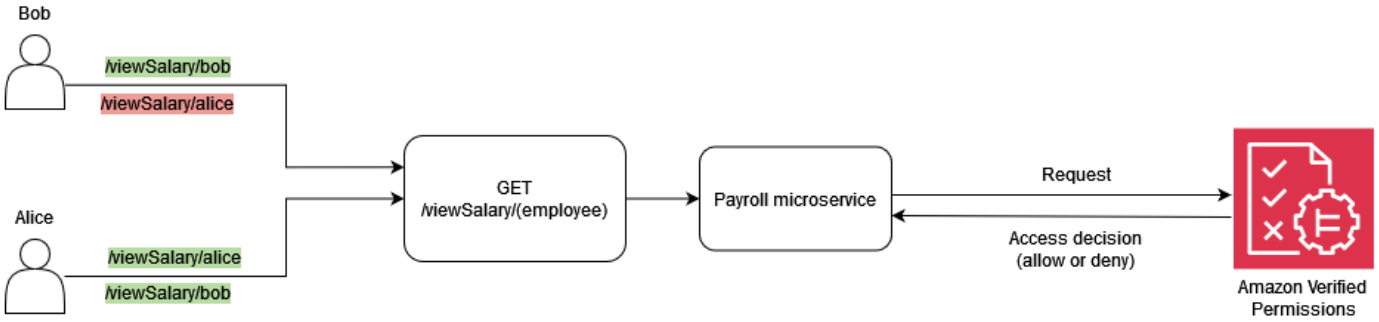
- 표현성 — Cedar는 인증 사용 사례를 지원하기 위해 특별히 제작되었으며 사람이 쉽게 읽을 수 있도록 개발되었습니다.
- 성능 — Cedar는 빠른 검색을 위한 인덱싱 정책을 지원하고 제한된 지연 시간으로 빠르고 확장 가능한 실시간 평가를 제공합니다.
- 분석 — Cedar는 정책을 최적화하고 보안 모델을 검증할 수 있는 분석 도구를 지원합니다.

자세한 내용은 Cedar 웹 사이트를 참조하십시오.

예 1: 권한이 검증된 기본 ABAC 및 Cedar

이 예제 시나리오에서는 Amazon Verified Permissions를 사용하여 가상 Payroll 마이크로서비스의 정보에 액세스할 수 있는 사용자를 결정합니다. 이 섹션에는 Cedar를 사용하여 액세스 제어 결정을 내리는 방법을 보여주는 Cedar 코드 스니펫이 포함되어 있습니다. 이 예제는 Cedar와 검증된 권한에서 제공하는 기능을 전체적으로 살펴보기 위한 것이 아닙니다. Cedar에 대한 자세한 개요는 Cedar 설명서를 참조하십시오.

다음 다이어그램에서는 이 `viewSalary GET` 방법과 관련된 두 가지 일반적인 업무 규칙을 적용하고자 합니다. 즉, 직원은 자신의 급여를 볼 수 있고 직원은 자신에게 보고한 사람의 급여를 볼 수 있다는 것입니다. 검증된 권한 정책을 사용하여 이러한 비즈니스 규칙을 적용할 수 있습니다.



직원은 자신의 급여를 볼 수 있습니다.

Cedar에서 기본 구조는 주체, 행동 또는 자원을 나타내는 주체입니다. 인증을 요청하고 검증된 권한 정책을 사용하여 평가를 시작하려면 주도자, 조치, 리소스, 엔티티 목록을 제공해야 합니다.

- 주체 (`principal`) 는 로그인한 사용자 또는 역할입니다.
- 조치 (`action`) 는 요청에 의해 평가되는 작업입니다.
- resource (`resource`) 는 액션이 액세스하는 구성 요소입니다.
- 엔티티 목록 (`entityList`) 에는 요청을 평가하는 데 필요한 모든 필수 엔티티가 포함되어 있습니다.

직원이 자신의 급여를 볼 수 있다는 비즈니스 규칙을 충족하기 위해 다음과 같은 검증된 권한 정책을 제공할 수 있습니다.

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

```
when {
    principal == resource.owner
};
```

이 정책은 요청에 포함된 리소스의 속성 소유자가 보안 주체와 ALLOW 동일한지 여부를 평가합니다. Action viewSalary 예를 들어 Bob이 급여 보고서를 요청한 로그인한 사용자이고 급여 보고서의 소유자이기도 한 경우 정책은 다음과 같이 평가됩니다. ALLOW

샘플 정책에 따라 평가되도록 다음과 같은 승인 요청이 Verified Permissions에 제출됩니다. 이 예시에서는 Bob이 viewSalary 요청을 하는 로그인한 사용자입니다. 따라서 Bob은 엔티티 유형의 주도자입니다Employee. Bob이 수행하려는 작업은 viewSalary, 이며, 표시되는 리소스는 해당 유형과 Salary-Bob 함께 viewSalary 표시됩니다Salary. Bob이 Salary-Bob 리소스를 볼 수 있는지 평가하려면 값이 Bob (주도자) 인 유형을 해당 유형이 Employee 있는 리소스의 소유자 속성에 연결하는 엔티티 구조를 제공해야 Salary 합니다. 이 구조를 a에 입력하면 소유자와 연관된 속성에 소유자가 Salary 포함되며 소유자는 유형과 Employee 값을 entityIdentifier 포함하는 소유자를 지정합니다Bob. entityList 검증된 권한은 권한 부여 요청에 principal 제공된 내용을 Salary 리소스와 연결된 owner 속성과 비교하여 결정을 내립니다.

```
{
  "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": {}
      }
    ]
  }
}
```

검증된 권한에 대한 권한 부여 요청은 다음을 출력으로 반환합니다. 여기서 속성은 `decision` `ALLOW` `DENY` 또는입니다.

```
{
    "determiningPolicies":
        [
            {
                "determiningPolicyId": "PAYROLLAPP_POLICYSTOREID"
            }
        ],
    "decision": "ALLOW",
    "errors": []
}
```

이 경우 Bob이 자신의 급여를 확인하려고 했기 때문에 Verified Permissions에 전송된 승인 요청은 다음으로 평가됩니다. `ALLOW` 하지만 우리의 목표는 검증된 권한을 사용하여 두 가지 비즈니스 규칙을 적용하는 것이었습니다. 다음과 같은 비즈니스 규칙도 준수해야 합니다.

직원은 자신에게 보고한 모든 사람의 급여를 볼 수 있습니다.

이 비즈니스 규칙을 충족하기 위해 다른 정책을 제공할 수 있습니다. 다음 정책은 작업이 `viewSalary` 수행되고 요청의 리소스가 보안 주체와 동일한 속성을 `owner.manager` 가지고 `ALLOW` 있는지 여부를 평가합니다. 예를 들어 Alice가 급여 보고서를 요청한 로그인한 사용자이고 Alice가 보고서 소유자의 관리자인 경우 정책은 다음과 같이 평가됩니다. `ALLOW`

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

샘플 정책에 따라 평가되도록 다음과 같은 승인 요청이 Verified Permissions에 제출됩니다. 이 예시에서는 Alice가 로그인한 사용자로서 `viewSalary` 요청을 보냅니다. 따라서 Alice는 주도자이고 엔티티는 다음과 같은 유형입니다. `Employee` Alice가 수행하려는 작업은 이며`viewSalary`, 표시되는 `viewSalary` 리소스는 값이 인 `Salary` 유형입니다. `Salary-Bob` Alice가 `Salary-Bob` 리소스를 볼 수 있는지 평가하려면 값이 1인 유형을 `Employee` `manager` 속성에 연결하는 엔티티 구조를 제공해야 합니다. 이 구조는 값이 1인 유형의 `owner` `Salary` 속성과 연결되어야 합니다. `Alice` `Salary-Bob` 이 구조를 a로 제공하면 관련된 속성에 소유자가 `Salary` 포함되며 소유자는 유형 `Employee` 및 값을 `entityIdentifier` Bob 포함하는 소유자를 지정합니다. `entityList` 검증된 권한은 먼저 `owner` 속성을 검사하여 `Employee` 유형과 값으로 Bob 평가됩니다. 그런 다음 검증된 권한은 연결된 `manager` 속성을 평가하고 제공된 보안 `Employee` 주체와 비교하여 권한 부여 결정을 내립니다. 이 경우 결정되는 이유는 `principal` 및 `resource.owner.manager` 속성이 `ALLOW` 동일하기 때문입니다.

```
{
  "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": []
      }
    ]
  }
```

```
}
```

지금까지 이 예제에서는 각 비즈니스 규칙의 조건을 독립적으로 충족하는 정책으로, 직원은 자신의 급여를 볼 수 있고, 직원은 자신에게 보고하는 모든 사람의 급여를 Verified Permission 정책으로 볼 수 있다는 `viewSalary` 방법과 관련된 두 가지 비즈니스 규칙을 제공했습니다. 또한 하나의 검증된 권한 정책을 사용하여 두 비즈니스 규칙의 조건을 모두 충족할 수 있습니다.

직원은 자신의 급여와 자신에게 보고한 사람의 급여를 볼 수 있습니다.

이전 권한 부여 요청을 사용하는 `ALLOW` 경우 다음 정책은 `owner.manager` 해당 조치가 `viewSalary` 조치이고 요청에 포함된 리소스의 속성이 다음과 같은지 또는 다음과 같은지 여부를 평가합니다. `principal` `owner` `principal`

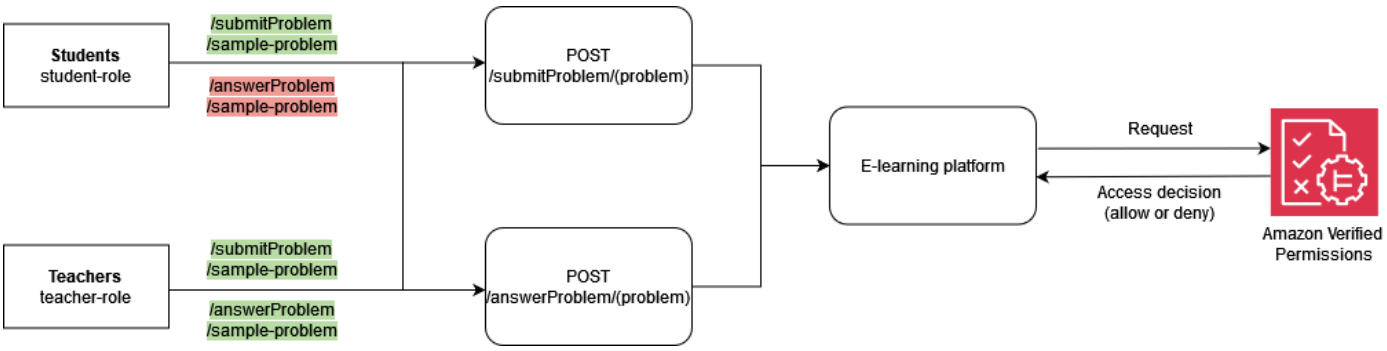
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

예를 들어 Alice가 급여 보고서를 요청하는 로그인한 사용자이고 Alice가 소유자 또는 보고서 소유자인 경우 정책은 다음과 같이 평가됩니다. `ALLOW`

Cedar 정책에서 논리 연산자를 사용하는 방법에 대한 자세한 내용은 Cedar 설명서를 참조하십시오.

예 2: 검증된 권한이 있는 기본 RBAC 및 Cedar

이 예제에서는 검증된 권한과 Cedar를 사용하여 기본 RBAC를 보여줍니다. 앞서 언급했듯이 Cedar의 기본 구조는 엔티티입니다. 개발자는 자체 엔티티를 정의하고 선택적으로 엔티티 간 관계를 만들 수 있습니다. 다음 예제에는 세 가지 유형의 엔티티 `UsersRoles`, 및 가 포함되어 `Problems` 있습니다. `Students`는 해당 유형의 엔티티로 간주될 `Teachers` `User` 수 `Role`, 있으며 각 엔티티는 0개 또는 둘 중 하나와 연결될 수 `Roles` 있습니다.



Cedar에서는 를 상위로 연결하여 이러한 관계를 표현합니다. `Role Student User Bob` 이 연결은 모든 학생 사용자를 논리적으로 한 그룹으로 그룹화합니다. Cedar의 그룹화에 대한 자세한 내용은 Cedar 설명서를 참조하십시오.

다음 정책은 해당 유형의 논리적 `ALLOW` 그룹에 연결된 모든 `submitProblem`, 주도자에 대한 조치 결정을 평가합니다. `Students Role`

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

다음 정책은 작업에 `ALLOW` 대한 결정 `submitProblem` 또는 해당 유형의 논리적 `answerProblem` 그룹에 연결된 모든 주체에 대해 평가합니다. `Teachers Role`

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

이러한 정책을 사용하여 요청을 평가하려면 평가 엔진이 권한 부여 요청 내에서 참조된 보안 주체가 적절한 그룹의 구성원인지 여부를 알아야 합니다. 따라서 응용 프로그램은 권한 부여 요청의 일부로 관련 그룹 구성원 정보를 평가 엔진에 전달해야 합니다. 이는 `entities` 속성을 통해 이루어지므로 인증 호출과 관련된 주도자 및 리소스에 대한 속성 및 그룹 구성원 데이터를 Cedar 평가 엔진에 제공할 수 있습니다. 다음 코드에서는 부모에게 전화를 `Role::"Students"` 거는 `User::"Bob"` 것으로 정의하여 그룹 구성원 자격을 나타냅니다.

```
{
  "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": []
        }
      ]
  }
}
```

이 예제에서 Bob은 answerProblem 요청을 보내는 로그인한 사용자입니다. 따라서 Bob은 주도자이고 엔티티는 다음과 같은 유형입니다User. Bob이 수행하려는 작업은 다음과 같습니

다answerProblem. Bob이 answerProblem 작업을 수행할 수 있는지 평가하려면 상위 엔티티를 User 로 나열하여 값이 인 엔티티를 연결하고 그룹 멤버십을 할당하는 엔티티 구조를 제공해야 합니다. Bob Role::"Students" 사용자 그룹의 엔티티만 작업을 수행할 수 있으므로 이 권한 부여 Role::"Students" 요청은 다음과 같이 평가됩니다. submitProblem DENY

반면, 값이 Alice 이고 그룹에 User 속하는 형식이 작업을 Role::"Teachers" 수행하려고 하면 권한 부여 요청은 수행으로 평가되는데, 이는 그룹 내 주도자가 모든 리소스에 대해 작업을 수행할 수 있도록 정책에 Role::"Teachers" 명시되어 있기 때문입니다. answerProblem ALLOW answerProblem 다음 코드는 까지 평가되는 이러한 유형의 권한 부여 요청을 보여줍니다. ALLOW

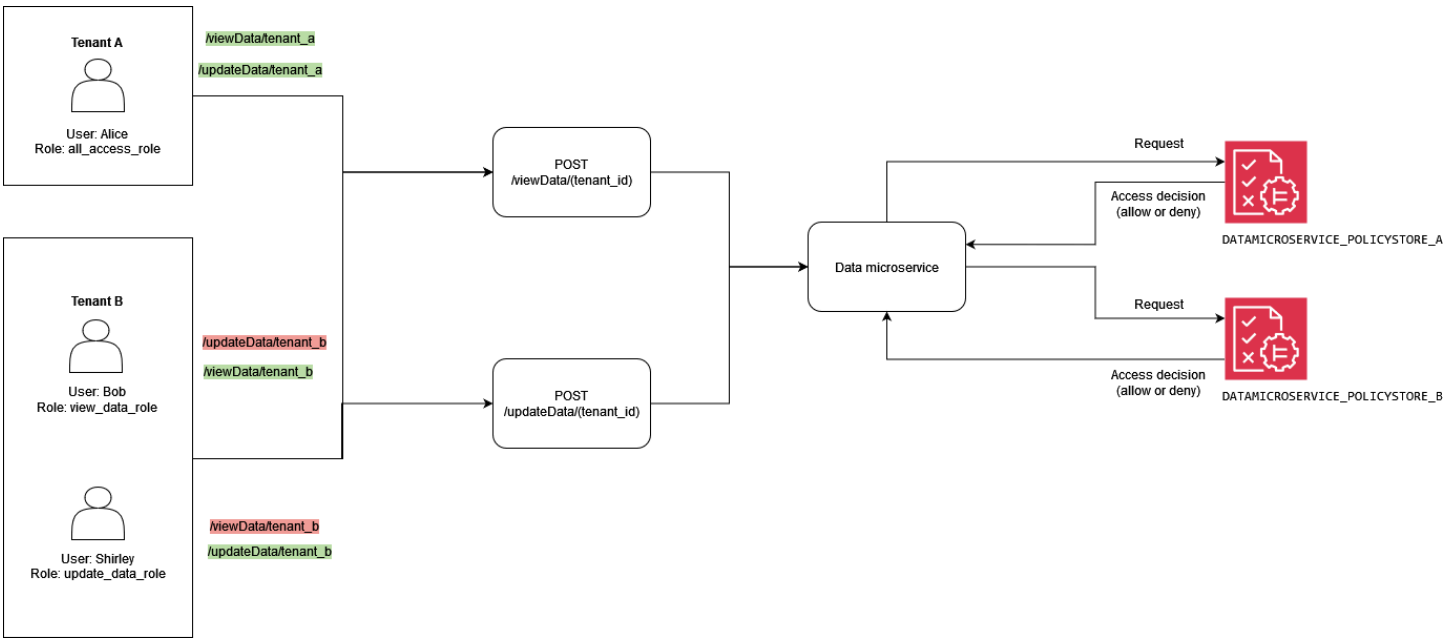
```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId":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

```
                "entityId": "SomeProblem"
            },
            "attributes": {},
            "parents": []
        }
    ]
  }
}
```

예 3: RBAC를 사용한 멀티테넌트 액세스 제어

이전 RBAC 예제를 자세히 설명하기 위해 SaaS 공급자의 일반적인 요구 사항인 SaaS 멀티 테넌시를 포함하도록 요구 사항을 확장할 수 있습니다. 멀티테넌트 솔루션에서는 항상 지정된 테넌트를 대신하여 리소스 액세스가 제공됩니다. 즉, 테넌트 A의 사용자는 데이터가 시스템에 논리적으로 또는 물리적으로 배치되어 있더라도 테넌트 B의 데이터를 볼 수 없습니다. 다음 예는 여러 개의 검증된 권한 정책 저장소를 사용하여 테넌트 격리를 구현하는 방법과 사용자 역할을 사용하여 테넌트 내에서 권한을 정의하는 방법을 보여줍니다.

검증된 권한으로 액세스 제어를 구현하면서 테넌트 격리를 유지하기 위해서는 테넌트별 정책 저장소별 디자인 패턴을 사용하는 것이 좋습니다. 이 시나리오에서는 테넌트 A와 테넌트 B 사용자 요청이 각각 별도의 정책 저장소에 `DATAMICROSERVICE_POLICYSTORE_A` 대해 검증됩니다. `DATAMICROSERVICE_POLICYSTORE_B` 멀티테넌트 SaaS 애플리케이션의 검증된 권한 설계 고려 사항에 대한 자세한 내용은 검증된 권한 다중 테넌트 설계 고려 사항 섹션을 참조하십시오.



다음 정책은 정책 저장소에 있습니다. DATAMICROSERVICE_POLICYSTORE_A 보안 주체가 해당 유형 그룹에 속하는지 확인합니다allAccessRole. Role 이 경우 주도자는 테넌트 A와 관련된 모든 리소스에 대해 viewData 및 updateData 작업을 수행할 수 있습니다.

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

정책 저장소에는 다음과 같은 DATAMICROSERVICE_POLICYSTORE_B 정책이 있습니다. 첫 번째 정책은 보안 주체가 유형 updateDataRole 그룹에 속하는지 확인합니다. Role 이 경우 주체가 테넌트 B와 연결된 리소스에서 updateData 작업을 수행할 수 있는 권한을 보안 주체에게 부여합니다.

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

이 두 번째 정책은 유형 viewDataRole 그룹에 속하는 보안 주체가 테넌트 B와 연결된 리소스에 대해 viewData 작업을 수행할 수 있도록 허용하도록 Role 요구합니다.

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

테넌트 A에서 이루어진 권한 부여 요청은 DATAMICROSERVICE_POLICYSTORE_A 정책 저장소로 전송되어 해당 저장소에 속한 정책을 통해 확인되어야 합니다. 이 경우에는 앞서 이 예제의 일부로 설명한 첫 번째 정책을 통해 확인됩니다. 이 권한 부여 요청에서는 값이 1인 유형의 User 보안 주체가 Alice viewData 작업 수행을 요청합니다. 주도자는 유형 그룹에 allAccessRole 속합니다Role. Alice가 SampleData 리소스에 대한 viewData 작업을 수행하려고 합니다. 앨리스가 allAccessRole 역할을 맡고 있기 때문에 이 평가를 통해 결정을 내리게 됩니다. ALLOW

```
{
```

```
  "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": []
      }
    ]
  }
}
```

대신 테넌트 B의 요청을 보면 다음과 같은 승인 요청이 표시됩니다. User Bob 요청은 테넌트 B에서 시작되었으므로 DATAMICROSERVICE_POLICYSTORE_B 정책 저장소로 전송됩니다. 이 요청에서 주체는 리소스에 updateData SampleData 대한 작업을 Bob 수행하려고 합니다. 하지만 Bob 은 (는)

해당 리소스의 updateData 작업에 액세스할 수 있는 그룹에 속하지 않습니다. 따라서 요청으로 인해 DENY 결정이 내려집니다.

```
{
  "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": []
      }
    ]
  }
}
```

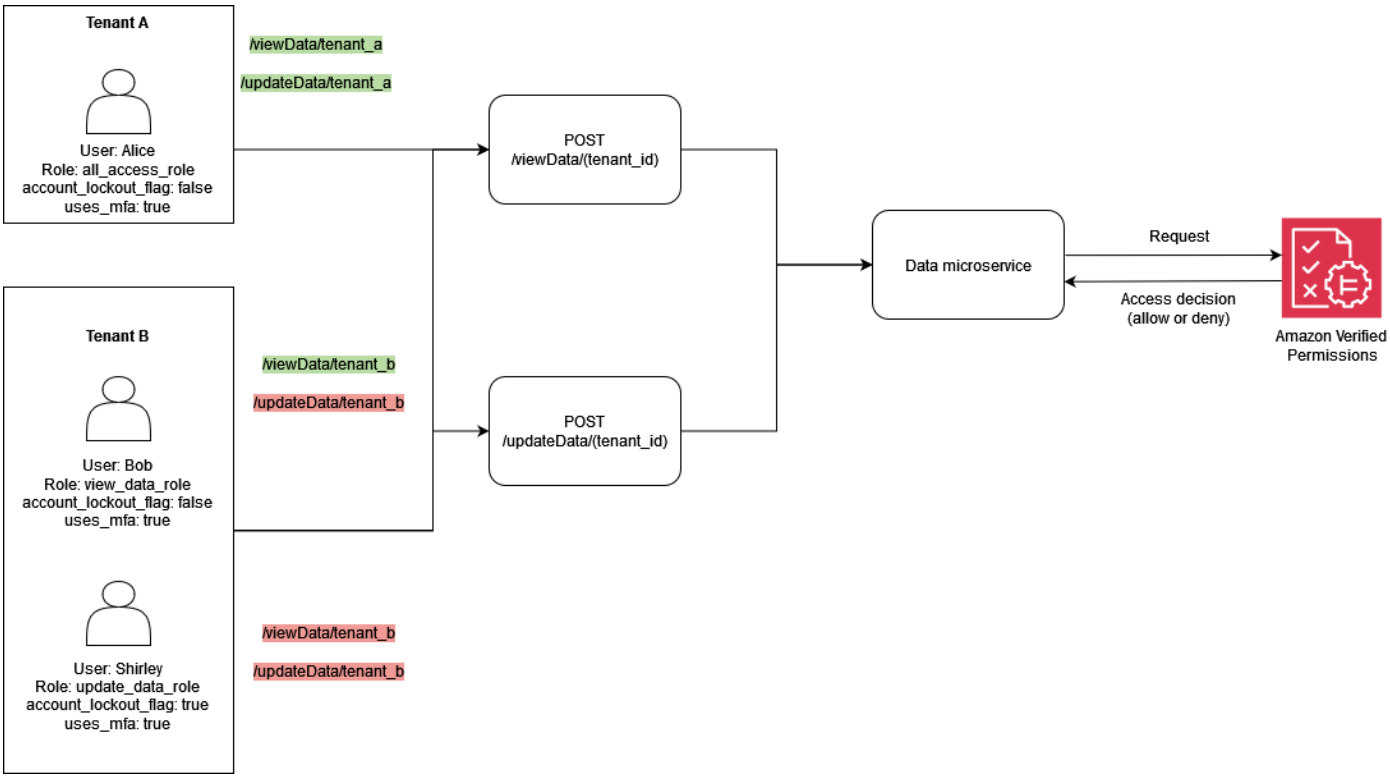
이 세 번째 예시에서는 리소스에 대한 viewData 작업을 User Alice 수행하려고 SampleData 시도합니다. 이 요청은 DATAMICROSERVICE_POLICYSTORE_A 정책 저장소로 전달됩니다. 보안 주체는 테넌트 Alice A에 Alice 속해 있기 때문에 보안 주체는 allAccessRole 해당 유형의 Role 테넌트 A에 속하므로 해당 보안 주체는 리소스에 대한 viewData 작업을 수행할 수 있습니다. 따라서 요청으로 인해 ALLOW 결정이 내려집니다.

```
{
  "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": []
      }
```

```
    ]
  }
}
```

예 4: RBAC 및 ABAC를 사용한 멀티테넌트 액세스 제어

이전 섹션의 RBAC 예제를 개선하려면 사용자에게 속성을 추가하여 멀티테넌트 액세스 제어를 위한 RBAC-ABAC 하이브리드 접근 방식을 만들 수 있습니다. 이 예제에는 이전 예제와 동일한 역할이 포함되지만 사용자 특성과 컨텍스트 매개 변수가 추가되었습니다. `account_lockout_flag` `uses_mfa` 또한 이 예제에서는 RBAC와 ABAC를 모두 사용하여 멀티테넌트 액세스 제어를 구현하는 다른 접근 방식을 취하고 각 테넌트에 대해 서로 다른 정책 저장소 대신 하나의 공유 정책 저장소를 사용합니다.



이 예는 이전 예와 마찬가지로 테넌트 A와 테넌트 B에 대한 권한 부여 결정을 제공해야 하는 멀티테넌트 SaaS 솔루션을 나타냅니다.

사용자 잠금 기능을 구현하기 위해 이 예제에서는 권한 부여 요청의 `User` 엔티티 보안 주체에 속성을 `account_lockout_flag` 추가합니다. 이 플래그는 시스템에 대한 사용자 액세스를 잠그고 잠긴 사용자에게는 `DENY` 모든 권한을 부여합니다. `account_lockout_flag`속성은 `User` 엔티티와 연결되어 있으며 여러 세션에서 플래그가 적극적으로 `User` 취소될 때까지 유효합니다. 이 예제에서는 `when` 조건을 사용하여 평가합니다. `account_lockout_flag`

이 예제에서는 요청 및 세션에 대한 세부 정보도 추가합니다. 컨텍스트 정보는 세션이 다단계 인증을 사용하여 인증되었음을 지정합니다. 이 검증을 구현하기 위해 예제에서는 when 조건을 사용하여 컨텍스트 필드의 일부로 uses_mfa 플래그를 평가합니다. 컨텍스트 추가 모범 사례에 대한 자세한 내용은 Cedar 설명서를 참조하십시오.

```
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
};
```

이 정책은 리소스가 요청 주체의 속성과 동일한 그룹에 Tenant 속하지 않는 한 리소스에 대한 액세스를 차단합니다. 테넌트 격리를 유지하기 위한 이러한 접근 방식을 단일 공유 다중 테넌트 정책 저장소 접근 방식이라고 합니다. 멀티테넌트 SaaS 애플리케이션의 검증된 권한 설계 고려 사항에 대한 자세한 내용은 검증된 권한 다중 테넌트 설계 고려 사항 섹션을 참조하십시오.

또한 이 정책은 보안 주체가 구성원으로 가입하도록 allAccessRole 보장하고 해당 주체에 대한 작업을 제한합니다. viewData updateData 또한 이 정책은 의 컨텍스트 값이 0인지, false 그리고 에 대한 컨텍스트 값이 account_lockout_flag 0인지 검증합니다uses_mfa. true

마찬가지로 다음 정책은 보안 주체와 리소스가 모두 동일한 테넌트에 연결되도록 하여 테넌트 간 액세스를 방지합니다. 또한 이 정책은 보안 주체가 구성원이 되도록 viewDataRole 보장하고 활동을 제한합니다. viewData 또한 의 컨텍스트 값이 0인지, 의 account_lockout_flag 컨텍스트 값이 로 uses_mfa 평가되는지도 확인합니다. false 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
};
```

세 번째 정책은 이전 정책과 비슷합니다. 정책에 따라 리소스는 가 대표하는 엔티티에 해당하는 그룹의 구성원이어야 principal.Tenant 합니다. 이렇게 하면 보안 주체와 리소스가 모두 테넌트 B와 연결되므로 테넌트 간 액세스가 방지됩니다. 이 정책은 보안 주체가 구성원이 되도록 updateDataRole 하고 해당 주체의 작업을 제한합니다. updateData 또한 이 정책은 의 컨텍스트 값이 account_lockout_flag 0인지false, 의 컨텍스트 값이 T로 uses_mfa 평가되는지도 확인합니다. 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
};
```

다음 권한 부여 요청은 이 섹션의 앞부분에서 설명한 세 가지 정책에 따라 평가됩니다. 이 권한 부여 요청에서는 유형 User 및 값이 인 보안 Alice 주체가 역할을 updateData allAccessRole 요청합니다. AliceTenant값이 인 속성을 가집니다Tenant::"TenantA". 수행하려는 작업은 Alice 이고 이 updateData, 작업이 적용될 리소스는 다음과 같은 SampleData 유형입니다Data. SampleData상위 TenantA 엔티티로 있습니다.

정책 저장소의 첫 번째 정책에 따라 <DATAMICROSERVICE_POLICYSTOREID> 정책 when 조항의 조건이 충족되는 경우 리소스에 대한 작업을 수행할 Alice 수 있습니다. updateData 첫 번째 조건에서는 평가할 principal.Tenant 속성이 필요합니다. TenantA 두 번째 조건에서는 주도자의 속성이 다음과 account_lockout_flag 같아야 false 합니다. 최종 조건을 uses_mfa 충족하려면 컨텍스트가 다음과 같아야 true 합니다. 세 가지 조건이 모두 충족되었으므로 요청은 ALLOW 결정을 반환합니다.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
```

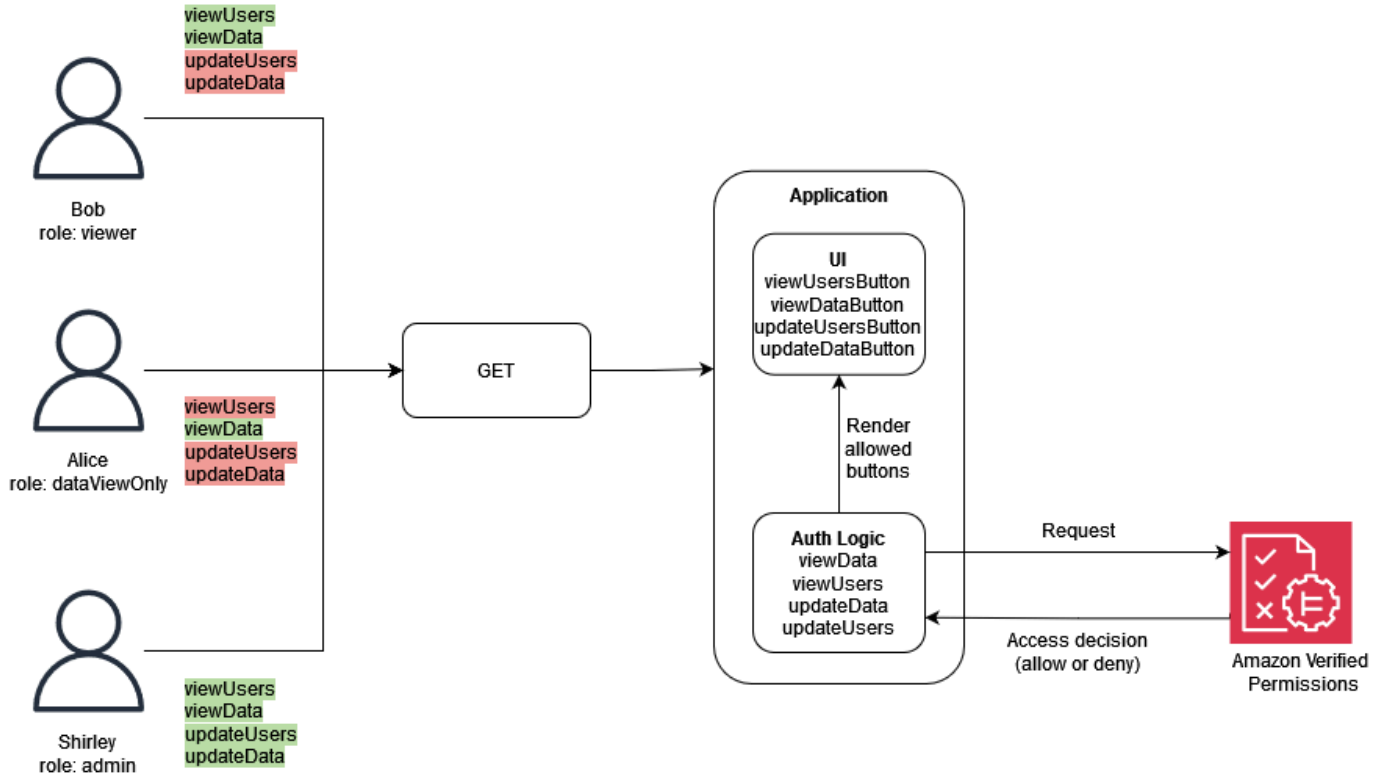
```
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "uses_mfa": {
            "boolean": true
        }
    }
  },
  "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"
            }
        ]
      }
    ]
  }
}
```

예 5: 검증된 권한 및 Cedar를 사용한 UI 필터링

또한 검증된 권한을 사용하여 승인된 작업을 기반으로 UI 요소의 RBAC 필터링을 구현할 수 있습니다. 이는 멀티테넌트 SaaS 애플리케이션의 경우 특정 사용자 또는 테넌트와 연결될 수 있는 상황에 맞는 UI 요소가 있는 애플리케이션에 매우 유용합니다.

다음 예에서는 이들 `Users` 중 2명이 업데이트를 수행할 수 `Role` `viewer` 없습니다. 이러한 사용자의 경우 UI에서 업데이트 버튼을 렌더링해서는 안 됩니다.



이 예제에서 단일 페이지 웹 애플리케이션에는 네 개의 버튼이 있습니다. 표시되는 버튼은 현재 애플리케이션에 로그인한 사용자에 따라 달라집니다. Role 단일 페이지 웹 애플리케이션은 UI를 렌더링할 때 Verified Permissions를 쿼리하여 사용자가 수행할 수 있는 작업을 확인한 다음 권한 부여 결정에 따라 버튼을 생성합니다.

다음 정책은 값이 1인 Role 유형에서 사용자와 데이터를 모두 볼 viewer 수 있도록 지정합니다. 이 정책에 대한 ALLOW 권한 부여 결정에는 viewData 또는 viewUsers 작업이 필요하며 Data 또는 유형과 관련된 리소스도 필요합니다Users. ALLOW결정에 따라 UI는 두 개의 버튼, 즉 viewDataButton 및 viewUsersButton 버튼을 렌더링할 수 있습니다.

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

다음 정책은 값이 1인 Role 유형만 데이터를 볼 viewerDataOnly 수 있도록 지정합니다. 이 정책에 대한 ALLOW 권한 부여 결정에는 viewData 조치가 필요하며 해당 유형과 관련된 리소스도 필요합니다Data. ALLOW결정을 내리면 UI가 버튼을 viewDataButton 렌더링할 수 있습니다.

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

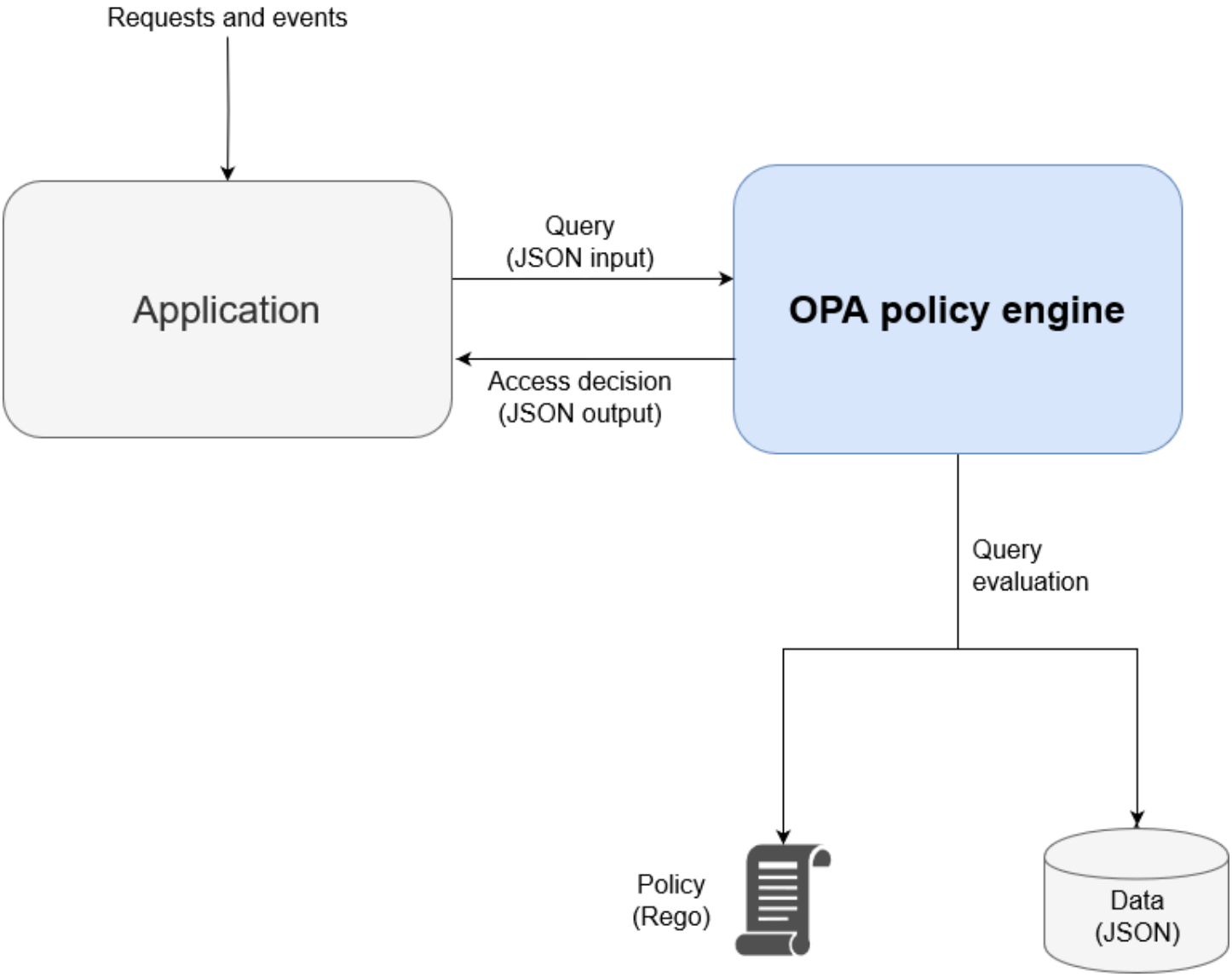
다음 정책은 값이 1인 유형이 Role 데이터 및 사용자를 편집하고 볼 admin 수 있도록 지정합니다. 이 정책에 대한 ALLOW 권한 부여 결정에는 updateData updateUsersviewUsers, viewData, 또는 조치가 필요하며, Data 또는 유형과 관련된 리소스도 필요합니다Users. ALLOW결정을 내리면 UI에서 네 개의 버튼 (updateDataButton, updateUsersButtonviewDataButton, 및viewUsersButton) 을 모두 렌더링할 수 있습니다.

```
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"]
};
```

OPA를 사용하여 PDP 구현

개방형 정책 에이전트 (OPA) 는 오픈 소스의 범용 정책 엔진입니다. OPA에는 많은 사용 사례가 있지만 PDP 구현과 관련된 사용 사례는 애플리케이션에서 권한 부여 논리를 분리하는 기능입니다. 이를 정책 디커플링이라고 합니다. OPA는 여러 가지 이유로 PDP를 구현하는 데 유용합니다. Rego라는 고급 선언적 언어를 사용하여 정책과 규칙의 초안을 작성합니다. 이러한 정책 및 규칙은 애플리케이션과는 별도로 존재하며 애플리케이션별 로직 없이도 권한 부여 결정을 내릴 수 있습니다. 또한 OPA는 권한 부여 결정을 간단하고 간단하게 검색할 수 있도록 RESTful API를 제공합니다. 권한 부여 결정을 내리기 위해 애플리케이션은 JSON 입력으로 OPA를 쿼리하고, OPA는 지정된 정책을 기준으로 입력을 평가하여 액세스 결정을 JSON으로 반환합니다. 또한 OPA는 권한 부여 결정과 관련이 있을 수 있는 외부 데이터를 가져올 수 있습니다.



OPA는 사용자 지정 정책 엔진에 비해 몇 가지 장점이 있습니다.

- OPA와 Rego를 통한 정책 평가 기능은 권한 부여 결정에 필요한 정책과 데이터를 삽입하기만 하면 되는 유연하고 사전 구축된 정책 엔진을 제공합니다. 이 정책 평가 로직은 맞춤형 정책 엔진 솔루션에서 다시 만들어야 할 것입니다.
- OPA는 정책을 선언적 언어로 작성하여 권한 부여 논리를 단순화합니다. 애플리케이션 개발 기술 없이도 애플리케이션 코드와 독립적으로 이러한 정책과 규칙을 수정하고 관리할 수 있습니다.
- OPA는 정책 적용 지점 (PEP) 과의 통합을 간소화하는 RESTful API를 제공합니다.
- OPA는 JSON 웹 토큰 (JWT) 의 검증 및 디코딩을 위한 기본 지원을 제공합니다.
- OPA는 인정된 권한 부여 표준이므로 특정 문제를 해결하기 위해 지원이나 연구가 필요한 경우 문서와 예제가 풍부합니다.

- OPA와 같은 권한 부여 표준을 채택하면 팀 응용 프로그램에서 사용하는 프로그래밍 언어에 관계없이 Rego로 작성된 정책을 팀 간에 공유할 수 있습니다.

OPA가 자동으로 제공하지 않는 두 가지가 있습니다.

- OPA에는 정책 업데이트 및 관리를 위한 강력한 컨트롤 플레인이 없습니다. OPA는 관리 API를 노출하여 정책 업데이트, 모니터링 및 로그 집계를 구현하기 위한 몇 가지 기본 패턴을 제공하지만 이 API와의 통합은 OPA 사용자가 처리해야 합니다. 가장 좋은 방법은 지속적 통합 및 지속적 배포 (CI/CD) 파이프라인을 사용하여 OPA에서 정책 버전을 관리, 수정 및 추적하고 정책을 관리하는 것입니다.
- OPA는 기본적으로 외부 소스에서 데이터를 검색할 수 없습니다. 권한 부여 결정을 위한 외부 데이터 소스는 사용자 속성을 보관하는 데이터베이스일 수 있습니다. 외부 데이터를 OPA에 제공하는 방식에는 어느 정도 유연성이 있습니다. 즉, 사전에 로컬에 캐시하거나 권한 부여 결정 요청 시 API에서 동적으로 검색할 수 있습니다. 하지만 이 정보를 가져오는 작업은 OPA가 대신 수행할 수 있는 작업이 아닙니다.

레고 개요

Rego는 범용 정책 언어이므로 스택의 모든 계층과 모든 도메인에서 작동합니다. Rego의 주요 목적은 JSON/YAML 입력 및 데이터를 수용하여 인프라 리소스, ID 및 운영에 대한 정책 기반 결정을 내리는 것입니다. Rego를 사용하면 언어를 변경하거나 확장할 필요 없이 스택 또는 도메인의 모든 계층에 대한 정책을 작성할 수 있습니다. Rego가 내릴 수 있는 결정의 몇 가지 예는 다음과 같습니다.

- 이 API 요청이 허용되나요 아니면 거부되나요?
- 이 애플리케이션의 백업 서버 호스트 이름은 무엇입니까?
- 제안된 인프라 변경의 위험 점수는 얼마입니까?
- 고가용성을 위해 이 컨테이너를 어떤 클러스터에 배포해야 합니까?
- 이 마이크로서비스에 사용해야 하는 라우팅 정보는 무엇입니까?

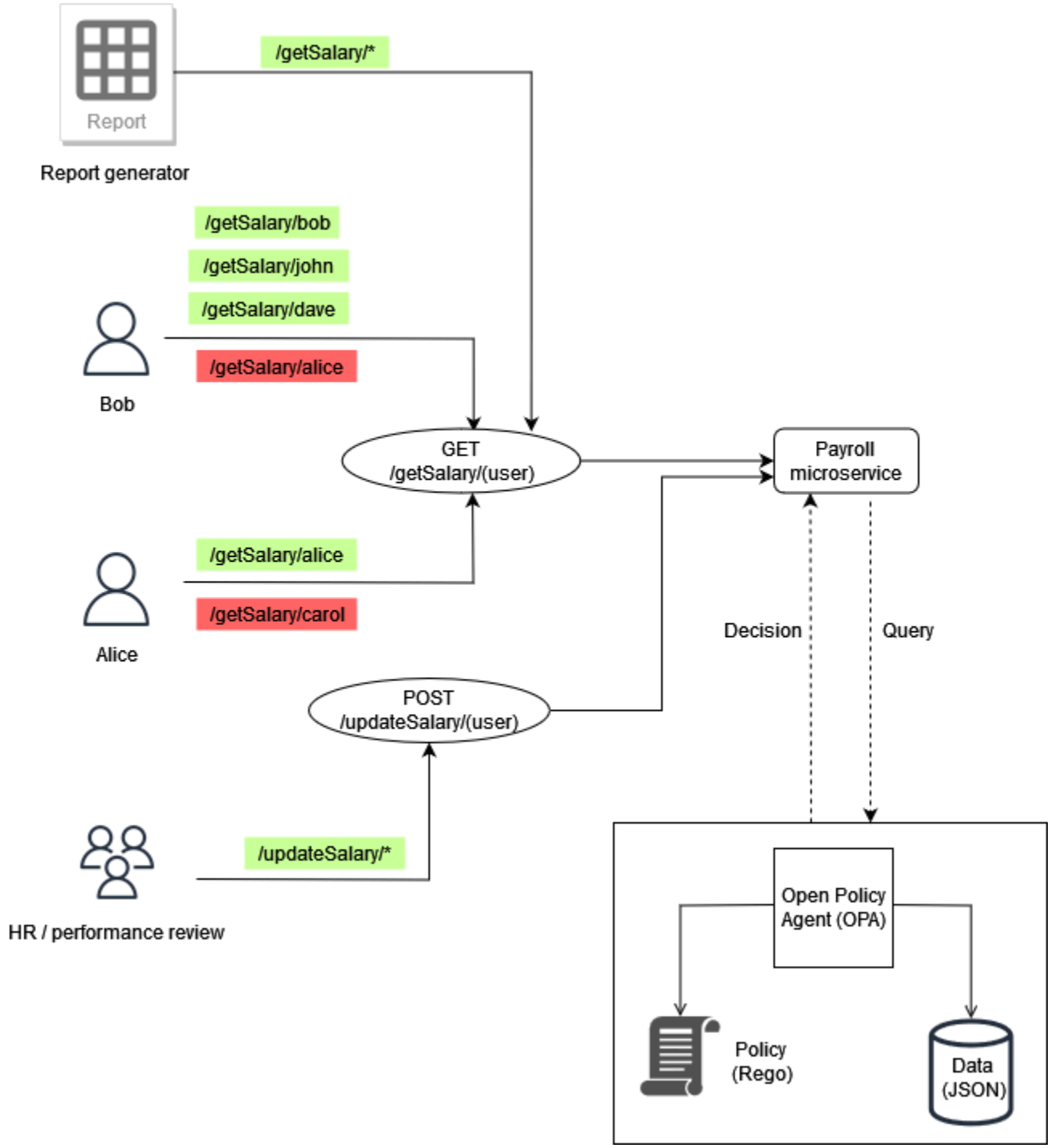
이러한 질문에 답하기 위해 Rego는 이러한 결정을 내리는 방법에 대한 기본 철학을 사용합니다. Rego에서 정책 초안을 작성할 때의 두 가지 주요 원칙은 다음과 같습니다.

- 모든 리소스, ID 또는 작업을 JSON 또는 YAML 데이터로 표시할 수 있습니다.
- 정책은 데이터에 적용되는 로직입니다.

Rego는 JSON/YAML 데이터 입력 평가 방법에 대한 로직을 정의하여 소프트웨어 시스템이 권한 부여 결정을 내리는 데 도움이 됩니다. C, Java, Go, Python과 같은 프로그래밍 언어가 이 문제에 대한 일반적인 해결책이지만 Rego는 시스템을 나타내는 데이터와 입력, 그리고 이 정보를 사용하여 정책을 결정하는 로직에 초점을 맞추도록 설계되었습니다.

예 1: OPA와 Rego를 사용한 기본 ABAC

이 섹션에서는 OPA를 사용하여 가상의 Payroll 마이크로서비스에 있는 정보에 액세스할 수 있는 사용자에 대한 액세스 결정을 내리는 시나리오를 설명합니다. Rego를 사용하여 액세스 제어 결정을 내리는 방법을 보여주기 위해 Rego 코드 스니펫이 제공됩니다. 이러한 예는 Rego 및 OPA 기능에 대한 완전한 설명도 아니고 전체적으로 살펴본 것도 아닙니다. Rego에 대한 더 자세한 개요를 보려면 OPA 웹 사이트의 Rego 설명서를 참조하는 것이 좋습니다.



기본 OPA 규칙 예제

이전 다이어그램에서 Payroll 마이크로서비스에 대해 OPA에서 적용하는 액세스 제어 규칙 중 하나는 다음과 같습니다.

직원은 자신의 급여를 확인할 수 있습니다.

Bob이 자신의 급여를 확인하기 위해 Payroll 마이크로서비스에 액세스하려고 하면 Payroll 마이크로서비스가 API 호출을 OPA RESTful API로 리디렉션하여 액세스 결정을 내릴 수 있습니다. 급여 서비스는 다음과 같은 JSON 입력을 사용하여 OPA에 결정을 쿼리합니다.

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

OPA는 쿼리를 기반으로 정책을 하나 또는 여러 개 선택합니다. 이 경우 Rego로 작성된 다음 정책이 JSON 입력을 평가합니다.

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

이 정책은 기본적으로 액세스를 거부합니다. 그런 다음 쿼리의 입력을 글로벌 변수에 바인딩하여 평가합니다. `input` 이 변수와 함께 도트 연산자를 사용하여 변수 값에 액세스합니다. 규칙의 표현식도 참이면 Rego 규칙이 true를 `allow` 반환합니다. Rego 규칙은 `method` 입력의 가 GET과 같은지 확인합니다. 그런 다음 목록의 첫 번째 요소가 있는지 확인한 다음 목록의 `path` 두 번째 요소를 변수에 할당합니다. `getSalary` `user` 마지막으로 요청한 내용이 변수와 일치하는지 `/getSalary/bob` 확인하여 액세스 중인 경로가 맞는지 확인합니다. `user` `input.user` `user` 이 규칙은 `allow` 출력에 표시된 것처럼 if-then 논리를 적용하여 부울 값을 반환합니다.

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

외부 데이터를 사용하는 일부 규칙

추가 OPA 기능을 시연하기 위해 적용 중인 액세스 규칙에 요구 사항을 추가할 수 있습니다. 위 그림의 맥락에서 이 액세스 제어 요구 사항을 적용하려는 경우를 가정해 보겠습니다.

직원은 자신에게 보고하는 모든 사람의 급여를 확인할 수 있습니다.

이 예에서 OPA는 액세스 결정을 내리는 데 도움이 되도록 가져올 수 있는 외부 데이터에 액세스할 수 있습니다.

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

OPA에서 고정 응답 대신 값 집합을 반환하는 부분 규칙을 생성하여 임의의 JSON 응답을 생성할 수 있습니다. 다음은 부분 규칙의 예입니다.

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

이 규칙은 값 (이 경우) 으로 보고하는 모든 사용자 집합을 반환합니다bob. `input.user [_]`규칙의 구문은 집합의 값을 반복하는 데 사용됩니다. 규칙의 출력은 다음과 같습니다.

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

이 정보를 검색하면 사용자가 관리자의 직속 직원인지 여부를 판단하는 데 도움이 될 수 있습니다. 일부 애플리케이션의 경우 단순 Boolean 응답을 반환하는 것보다 동적 JSON을 반환하는 것이 더 좋습니다.

모두 통합

마지막 액세스 요구 사항은 두 요구 사항에 지정된 조건을 결합하므로 처음 두 액세스 요구 사항보다 더 복잡합니다.

직원은 자신의 급여와 자신에게 보고한 사람의 급여를 확인할 수 있습니다.

이 요구 사항을 충족하기 위해 다음 Rego 정책을 사용할 수 있습니다.

```
default allow = false
```

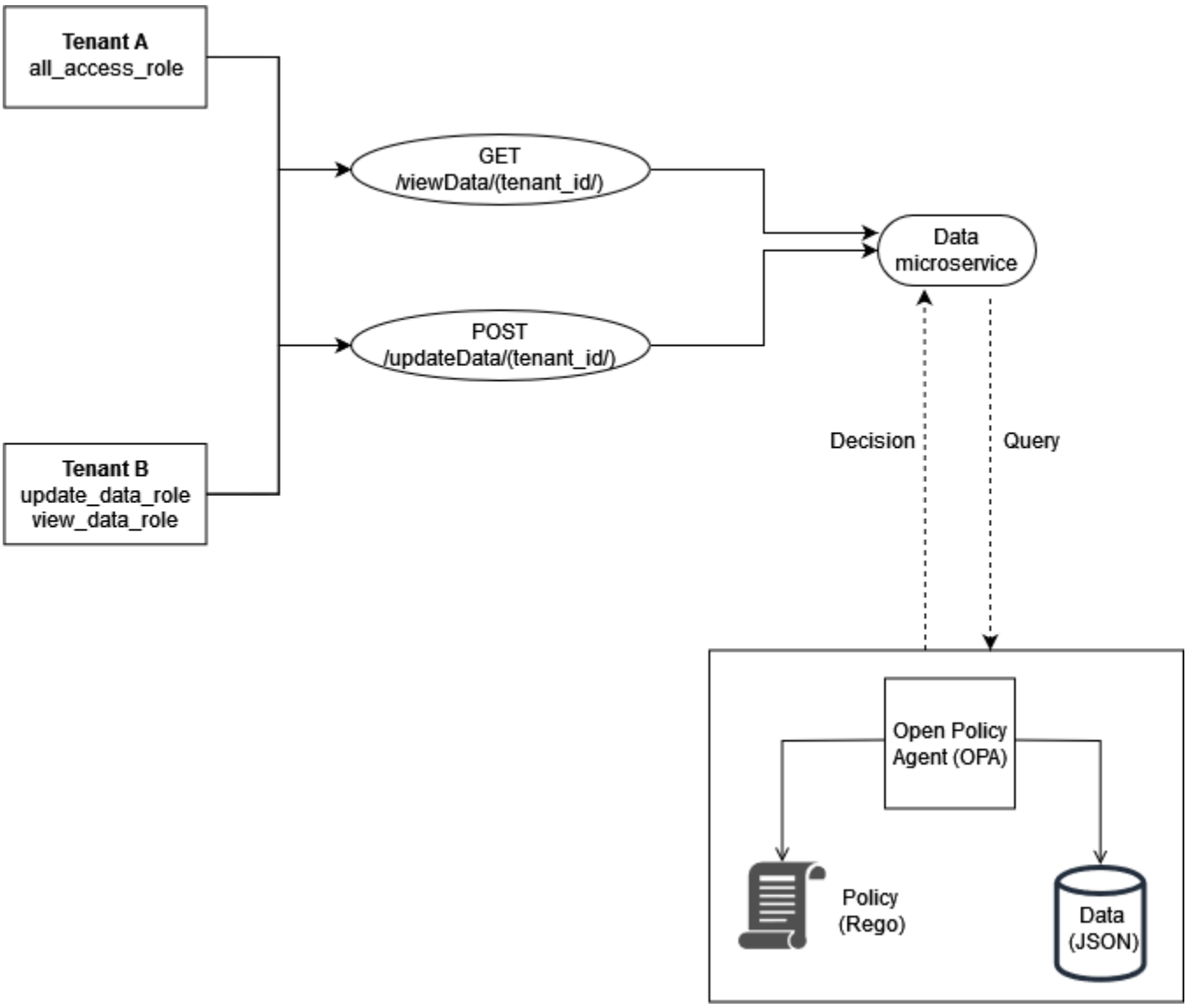
```
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

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

정책의 첫 번째 규칙은 앞에서 설명한 것처럼 자신의 급여 정보를 보려고 하는 모든 사용자에게 액세스를 허용합니다. 이름이 같은 규칙을 두 개 사용하면 `allow` Rego에서 논리 OR 연산자 역할을 합니다. 두 번째 규칙은 `input.user` (이전 다이어그램의 데이터에서) 관련된 모든 부하 직원의 목록을 검색하고 이 목록을 변수에 할당합니다. `managers` 마지막으로, 이 규칙은 사용자 이름이 변수에 포함되어 있는지 `input.user` 확인하여 급여를 보려는 사용자가 부하 직원인지 여부를 확인합니다. `managers`

이 섹션의 예는 매우 기본적이며 Rego 및 OPA의 기능에 대한 완전하거나 철저한 설명을 제공하지는 않습니다. 자세한 내용은 OPA 설명서를 검토하고 OPA GitHub README 파일을 참조하고 Rego 플레이그라운드에서 실험해 보십시오.

예 2: OPA와 Rego를 사용한 멀티테넌트 액세스 제어 및 사용자 정의 RBAC

이 예제에서는 OPA와 Rego를 사용하여 테넌트 사용자가 정의한 사용자 지정 역할이 있는 다중 테넌트 애플리케이션의 API에서 액세스 제어를 구현하는 방법을 보여줍니다. 또한 테넌트를 기반으로 액세스를 제한하는 방법도 보여줍니다. 이 모델은 OPA가 상위 역할에서 제공되는 정보를 기반으로 세분화된 권한 결정을 내리는 방법을 보여줍니다.



테넌트의 역할은 OPA에 대한 액세스 결정을 내리는 데 사용되는 외부 데이터 (RBAC 데이터) 에 저장됩니다.

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

```
}
```

테넌트 사용자가 정의하는 이러한 역할은 외부 데이터 소스 또는 ID 공급자 (IdP) 에 저장해야 합니다. IdP (ID 공급자) 는 테넌트가 정의한 역할을 권한 및 테넌트 자체에 매핑할 때 신뢰할 수 있는 소스 역할을 할 수 있습니다.

이 예에서는 OPA의 두 정책을 사용하여 권한 부여를 결정하고 이러한 정책이 테넌트 격리를 적용하는 방법을 살펴봅니다. 이러한 정책은 앞서 정의한 RBAC 데이터를 사용합니다.

```
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")
}
```

이 규칙이 어떻게 작동하는지 확인하려면 다음과 같은 입력이 포함된 OPA 쿼리를 고려해 보십시오.

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

이 API 호출에 대한 권한 부여는 RBAC 데이터, OPA 정책 및 OPA 쿼리 입력을 결합하여 다음과 같이 결정됩니다.

1. 에서 사용자가 `Tenant A` API를 호출합니다. `/viewData/tenant_a`
2. 데이터 마이크로서비스는 호출을 수신하고 `allowViewData` 규칙을 쿼리하여 OPA 쿼리 입력 예제에 표시된 입력을 전달합니다.
3. OPA는 OPA 정책의 쿼리된 규칙을 사용하여 제공된 입력을 평가합니다. 또한 OPA는 RBAC 데이터의 데이터를 사용하여 입력을 평가합니다. OPA는 다음을 수행합니다.
   a. API 호출에 사용된 메서드가 인지 확인합니다. `GET`
   b. 요청된 경로가 맞는지 확인합니다. `viewData`

c. 경로에 tenant_id 있는 가 사용자와 input.tenant_id 관련된 것과 같은지 확인합니다. 이렇게 하면 테넌트 격리가 유지됩니다. 역할이 같더라도 다른 테넌트는 이 API 호출을 수행할 권한을 부여받을 수 없습니다.

d. 역할의 외부 데이터에서 역할 권한 목록을 가져와 변수에 할당합니다. role_permissions 이 목록은 에서 사용자와 연결된 테넌트 정의 역할을 사용하여 검색됩니다. input.role.

e. 권한이 role_permissions 포함되어 있는지 확인합니다. viewData.

4. OPA는 다음 결정을 데이터 마이크로서비스에 반환합니다.

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

이 프로세스는 RBAC 및 테넌트 인식이 OPA를 통한 권한 부여 결정에 어떻게 기여할 수 있는지를 보여줍니다. 이 점을 더 자세히 설명하려면 다음 쿼리 입력을 /viewData/tenant_b 사용한 API 호출을 고려해 보십시오.

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

이 규칙은 역할이 다른 다른 테넌트에 대한 결과이긴 하지만 OPA 쿼리 입력과 동일한 출력을 반환합니다. 이는 이 호출이 /tenant_b 목적이고 RBAC view_data_role 내 데이터에 여전히 관련 viewData 권한이 있기 때문입니다. 에 대해 동일한 유형의 액세스 제어를 적용하려면 유사한 / updateData OPA 규칙을 사용할 수 있습니다.

```
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")
}
```

이 규칙은 기능적으로는 규칙과 allowViewData 동일하지만 다른 경로와 입력 방법을 검증합니다. 규칙은 여전히 테넌트 격리를 보장하고 테넌트 정의 역할이 API 호출자 권한을 부여하는지 확인합니다. 이를 어떻게 적용할 수 있는지 알아보려면 API 호출에 대한 다음 쿼리 입력을 살펴보세요. /updateData/tenant_b

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

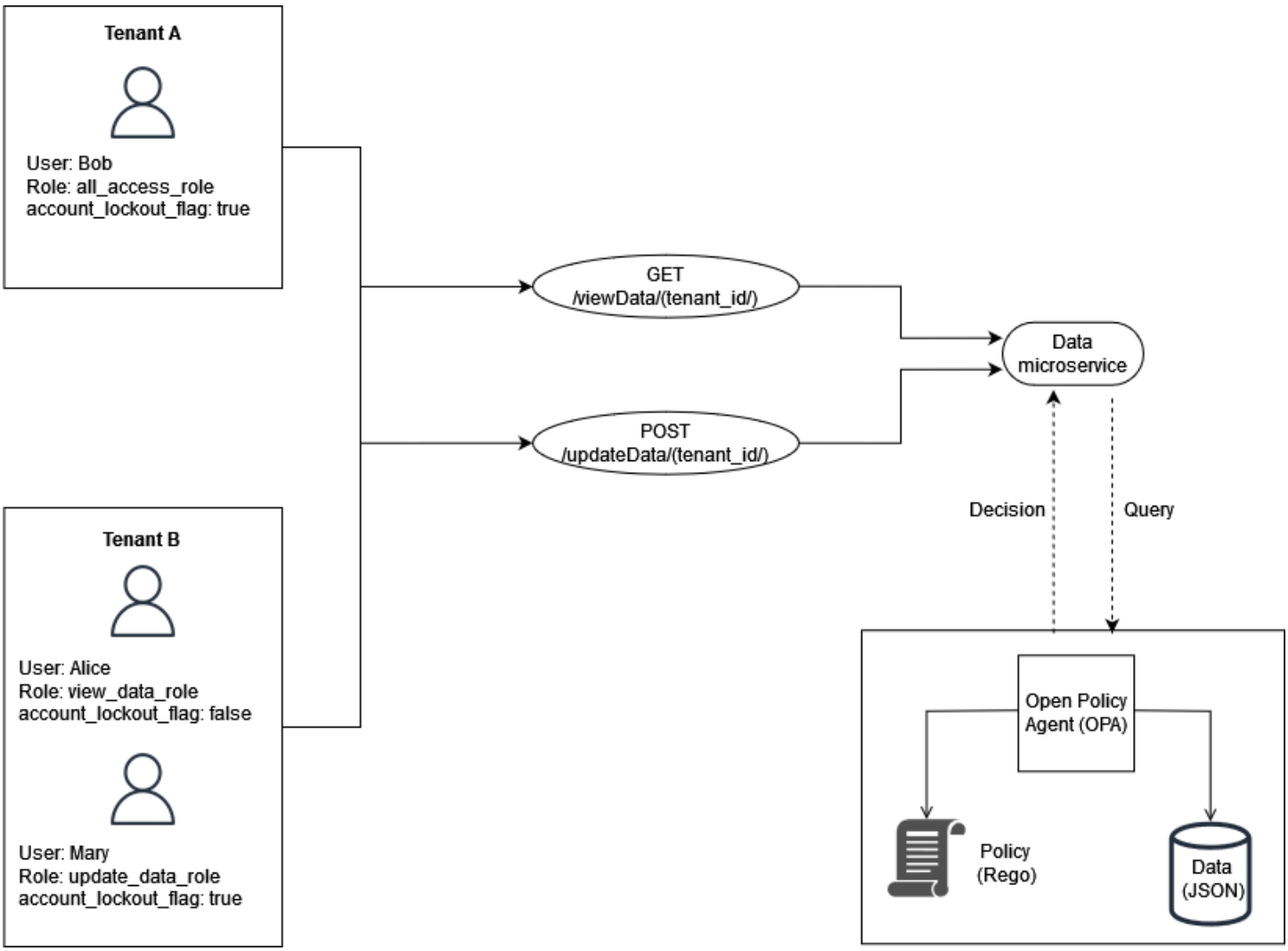
이 쿼리 입력을 allowUpdateData 규칙으로 평가하면 다음과 같은 권한 부여 결정이 반환됩니다.

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

이 호출은 승인되지 않습니다. API 호출자가 올바른 tenant_id 호출자와 연결되어 승인된 메서드를 사용하여 API를 호출하더라도 input.role view_data_role 테넌트가 정의한 호출자입니다. 에는 updateData 권한이 없으므로 호출이 허용되지 view_data_role 않습니다. /updateData 이 통화는 tenant_b 권한이 있는 사용자라면 성공했을 update_data_role 것입니다.

예 3: OPA와 Rego를 사용한 RBAC 및 ABAC에 대한 멀티테넌트 액세스 제어

이전 섹션의 RBAC 예제를 개선하려면 사용자에게 속성을 추가할 수 있습니다.



이 예제는 이전 예제와 동일한 역할을 포함하지만 사용자 속성을 추가합니다. `account_lockout_flag` 이는 특정 역할과 관련이 없는 사용자별 특성입니다. 이 예제에서 이전에 사용한 것과 동일한 RBAC 외부 데이터를 사용할 수 있습니다.

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

account_lockout_flag사용자 Bob에 /viewData/tenant_a 대한 OPA 쿼리 입력의 일부로 사용자 속성을 데이터 서비스에 전달할 수 있습니다.

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

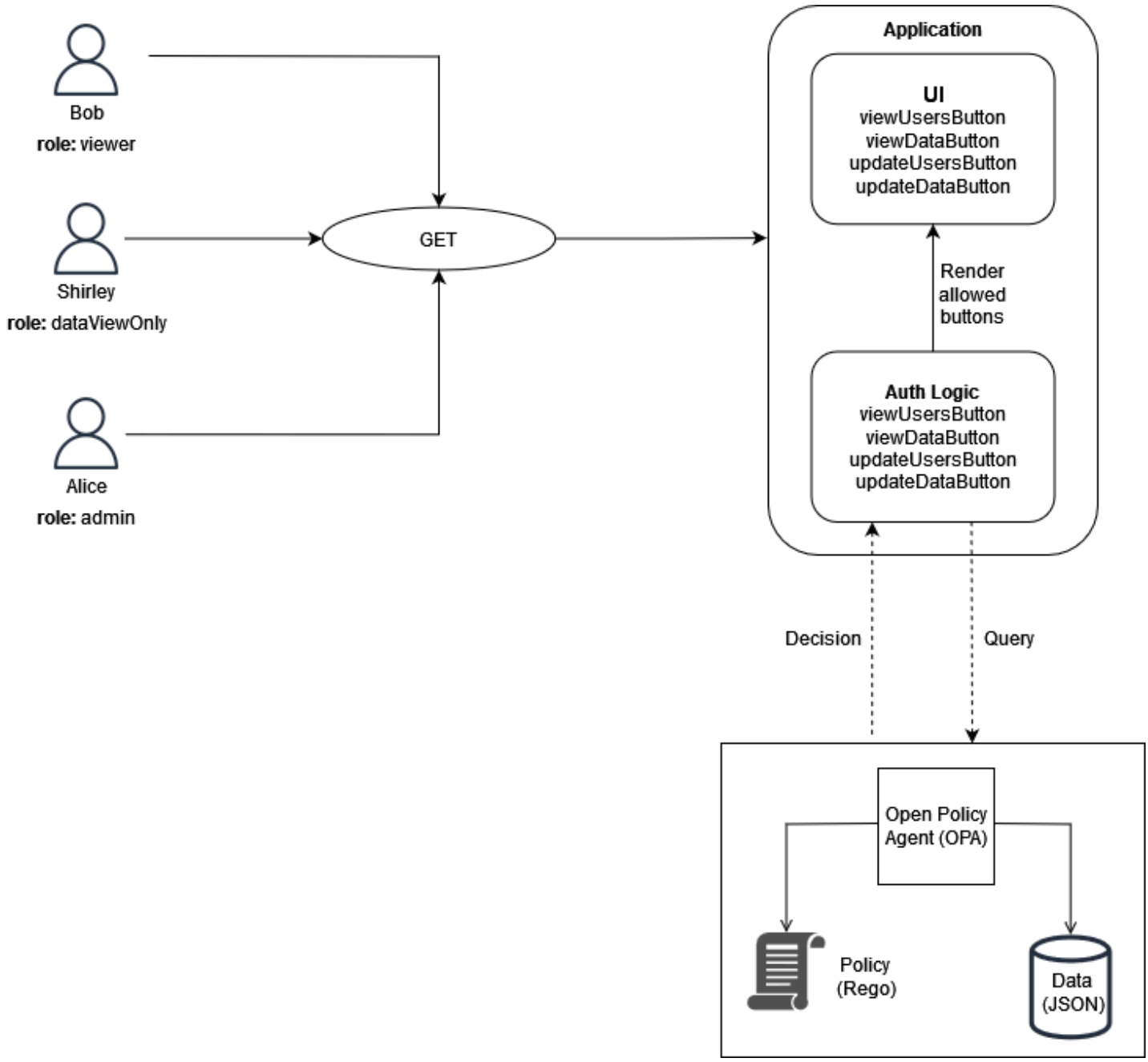
액세스 결정을 위해 쿼리되는 규칙은 이전 예와 비슷하지만 속성을 확인하기 위한 추가 행이 포함되어 있습니다. account_lockout_flag

```
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"
}
```

이 쿼리는 권한 부여 결정을 반환합니다. false 이는 Bob을 true 위한 account_lockout_flag attribute 것이고 Rego 규칙은 Bob에게 올바른 역할과 테넌트가 있음에도 불구하고 액세스를 allowViewData 거부하기 때문입니다.

예 4: OPA와 Rego를 사용한 UI 필터링

OPA와 Rego의 유연성은 UI 요소를 필터링하는 기능을 지원합니다. 다음 예제는 OPA 부분 규칙이 RBAC를 사용하여 UI에 표시되어야 하는 요소에 대한 권한 부여 결정을 내리는 방법을 보여줍니다. 이 방법은 OPA로 UI 요소를 필터링할 수 있는 다양한 방법 중 하나입니다.



이 예제에서 단일 페이지 웹 애플리케이션에는 네 개의 버튼이 있습니다. Bob's, Shirley의 UI를 필터링하여 역할에 해당하는 버튼만 볼 수 있도록 하려고 한다고 가정해 보겠습니다. UI는 사용자로부터 요청을 받으면 OPA 부분 규칙을 쿼리하여 UI에 표시할 버튼을 결정합니다. 역할을 `viewer` 가진 Bob이 UI에 요청을 보내면 쿼리는 다음을 OPA에 입력으로 전달합니다.

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

OPA는 RBAC용으로 구조화된 외부 데이터를 사용하여 액세스 결정을 내립니다.

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

OPA 부분 규칙은 외부 데이터와 입력을 모두 사용하여 허용된 작업 목록을 생성합니다.

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

부분 규칙에서 OPA는 `input.role` 지정된 항목을 쿼리의 일부로 사용하여 표시할 버튼을 결정합니다. Bob이 역할을 담당하고 외부 데이터에 따르면 뷰어에게는 두 가지 권한`viewer`, 즉 `viewUsers` 및 `viewData` 권한이 부여됩니다. 따라서 Bob (및 뷰어 역할을 가진 다른 모든 사용자) 에 대한 이 규칙의 출력은 다음과 같습니다.

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

`dataViewOnly`역할을 가진 Shirley의 출력에는 권한 버튼이 포함됩니다. `viewData` `admin`역할을 맡은 Alice의 출력에는 이러한 권한이 모두 포함됩니다. 이러한 응답은 OPA를 쿼리하면 UI로 반환됩니다. `user_permissions` 그러면 애플리케이션은 이 응답을 사용하여`viewUsersButton`, `viewDataButton`, `updateUsersButton`, 및 를 숨기거나 표시할 수 있습니다. `updateDataButton`

사용자 지정 정책 엔진 사용

PDP를 구현하는 또 다른 방법은 사용자 지정 정책 엔진을 만드는 것입니다. 이 정책 엔진의 목표는 응용 프로그램에서 권한 부여 논리를 분리하는 것입니다. 사용자 지정 정책 엔진은 Verified Permissions 또는 OPA와 마찬가지로 정책 분리를 달성하기 위한 권한 부여 결정을 담당합니다. 이 솔루션과 검증된 권한 또는 OPA 사용 간의 주요 차이점은 정책을 작성하고 평가하는 로직이 사용자 지정 정책 엔진

에 맞게 사용자 정의되어 있다는 것입니다. 엔진과의 모든 상호 작용은 API 또는 기타 방법을 통해 노출되어야 애플리케이션에 권한 부여 결정을 내릴 수 있습니다. 모든 프로그래밍 언어로 사용자 지정 정책 엔진을 작성하거나 CEL (Common Expression Language) 과 같은 정책 평가를 위한 다른 메커니즘을 사용할 수 있습니다.

PEP 구현하기

정책 적용 지점 (PEP) 은 평가를 위해 PDP (정책 결정 지점) 에 전송되는 권한 부여 요청을 수신하는 역할을 합니다. PEP는 데이터와 리소스를 보호해야 하거나 권한 부여 논리가 적용되는 애플리케이션 어디에나 있을 수 있습니다. PEP는 PDP에 비해 비교적 간단합니다. PEP는 권한 부여 결정을 요청하고 평가하는 역할만 담당하며 권한 부여 논리는 필요하지 않습니다. PDP와 달리 PEP는 SaaS 애플리케이션에서 중앙 집중화될 수 없습니다. 애플리케이션 및 액세스 포인트 전체에 권한 부여 및 액세스 제어를 구현해야 하기 때문입니다. PEP는 API, 마이크로서비스, BFF (Backend for Fronend) 계층 또는 액세스 제어가 필요하거나 필요한 애플리케이션 내 모든 지점에 적용할 수 있습니다. PEP를 애플리케이션에 널리 보급하면 여러 지점에서 권한 부여를 자주 독립적으로 검증할 수 있습니다.

PEP를 구현하기 위한 첫 번째 단계는 애플리케이션에서 액세스 제어를 적용해야 하는 위치를 결정하는 것입니다. PEP를 애플리케이션에 통합해야 하는 위치를 결정할 때는 다음 원칙을 고려하십시오.

애플리케이션이 API를 노출하는 경우 해당 API에 대한 권한 부여 및 액세스 제어가 있어야 합니다.

마이크로서비스 지향 또는 서비스 지향 아키텍처에서 API는 다양한 애플리케이션 함수를 구분하는 역할을 하기 때문입니다. 액세스 제어를 애플리케이션 함수 간의 논리적 체크포인트로 포함하는 것이 합리적입니다. SaaS 애플리케이션의 각 API에 액세스하기 위한 사전 요구 사항으로 PEP를 포함하는 것이 좋습니다. 애플리케이션의 다른 지점에서 권한 부여를 통합할 수도 있습니다. 모놀리식 애플리케이션에서는 애플리케이션 자체의 로직 내에 PEP를 통합해야 할 수도 있습니다. PEP를 포함해야 하는 단일 위치는 없지만 API 원칙을 출발점으로 삼는 것을 고려해 보십시오.

승인 결정 요청

PEP는 PDP에 승인 결정을 요청해야 합니다. 요청은 여러 형식을 취할 수 있습니다. 권한 부여 결정을 요청하는 가장 쉽고 접근하기 쉬운 방법은 PDP에 의해 노출되는 RESTful API (OPA 또는 검증된 권한) 에 권한 부여 요청 또는 쿼리를 보내는 것입니다. 검증된 권한을 사용하는 경우 AWS SDK를 사용하여 IsAuthorized메서드를 호출하여 권한 부여 결정을 검색할 수도 있습니다. 이 패턴에서 PEP의 유일한 기능은 승인 요청 또는 쿼리에 필요한 정보를 전달하는 것입니다. 이는 API에서 수신한 요청을 PDP에 입력으로 전달하는 것만큼 간단할 수 있습니다. PEP를 생성하는 다른 방법도 있습니다. 예를 들어, API를 사용하는 대신 Go 프로그래밍 언어로 작성된 응용 프로그램과 OPA PDP를 라이브러리로 로컬로 통합할 수 있습니다.

권한 부여 결정 평가

PEP에는 승인 결정의 결과를 평가하는 로직이 포함되어야 합니다. PDP가 API로 노출되는 경우 권한 부여 결정은 대부분 JSON 형식으로 이루어지며 API 호출을 통해 반환됩니다. PEP는 이 JSON 코드를

평가하여 취해진 조치가 승인되었는지 여부를 결정해야 합니다. 예를 들어 PDP가 부울 방식의 허용 또는 거부 권한 부여 결정을 제공하도록 설계된 경우 PEP는 단순히 이 값을 확인한 다음 허용에 대해서는 HTTP 상태 코드 200을, 거부를 위해서는 HTTP 상태 코드 403을 반환할 수 있습니다. API에 액세스하기 위한 전제 조건으로 PEP를 통합하는 이 패턴은 SaaS 애플리케이션 전반에 액세스 제어를 구현하기 위한 쉽게 구현되고 매우 효과적인 패턴입니다. 더 복잡한 시나리오에서는 PEP가 PDP에서 반환한 임의의 JSON 코드를 평가해야 할 수도 있습니다. PDP가 반환하는 권한 부여 결정을 해석하는 데 필요한 모든 논리를 포함하도록 PEP를 작성해야 합니다. PEP는 응용 프로그램의 다양한 위치에 구현될 가능성이 높으므로 PEP 코드를 선택한 프로그래밍 언어의 재사용 가능한 라이브러리 또는 아티팩트로 패키징하는 것이 좋습니다. 이렇게 하면 재작업을 최소화하면서 애플리케이션의 어느 시점에서든 PEP를 쉽게 통합할 수 있습니다.

멀티테넌트 SaaS 아키텍처를 위한 설계 모델

API 액세스 제어 및 권한 부여를 구현하는 방법은 여러 가지가 있습니다. 이 가이드에서는 멀티테넌트 SaaS 아키텍처에 효과적인 세 가지 설계 모델에 중점을 둡니다. 이러한 설계는 PDP (정책 결정 지점) 및 PEP (정책 적용 지점) 구현을 위한 상위 수준의 참조 역할을 하여 애플리케이션을 위한 일관되고 어디에나 있는 권한 부여 모델을 형성합니다.

설계 모델:

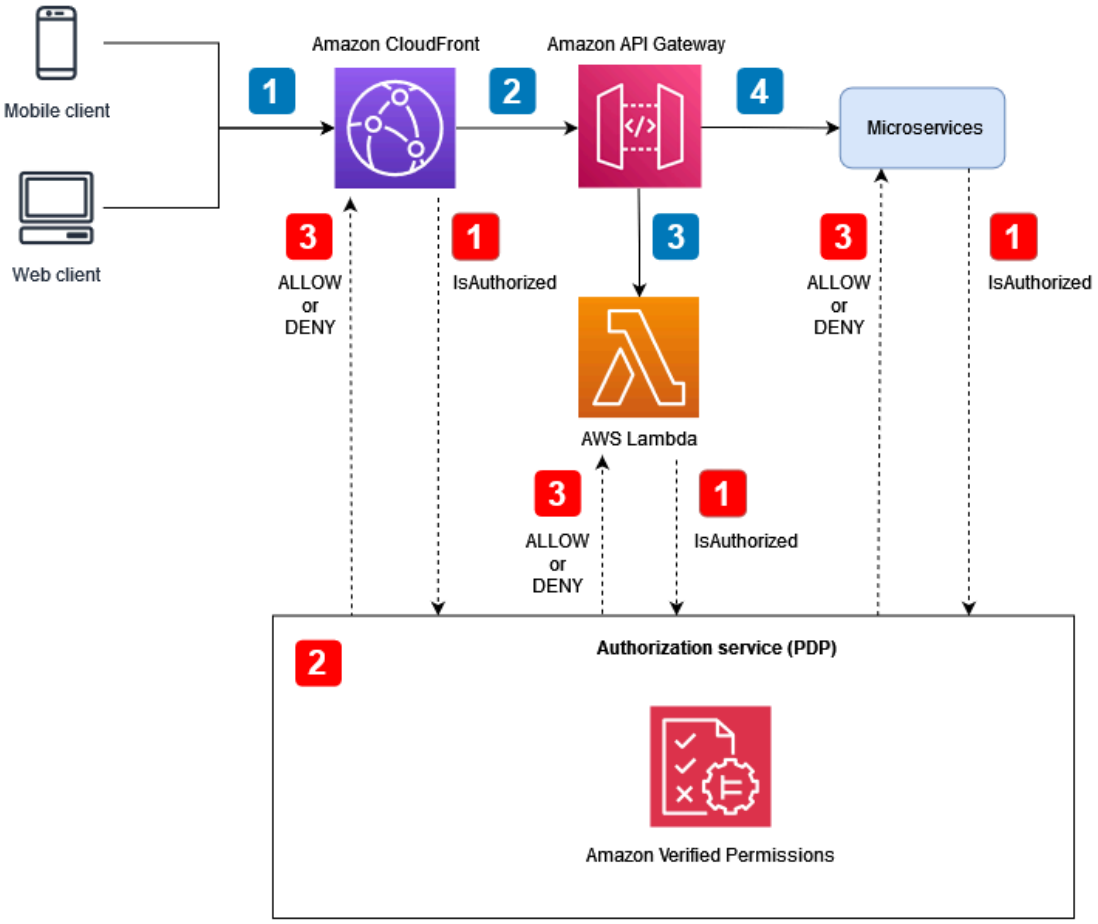
- Amazon 검증 권한을 위한 디자인 모델
- OPA용 설계 모델

Amazon 검증 권한을 위한 디자인 모델

API에서 PEP가 포함된 중앙 집중식 PDP 사용

API에 정책 적용 지점 (PEP) 이 포함된 중앙 집중식 정책 결정 지점 (PDP) 모델은 업계 모범 사례를 따라 API 액세스 제어 및 권한 부여를 위한 효과적이고 쉽게 유지 관리할 수 있는 시스템을 만듭니다. 이 접근 방식은 다음과 같은 몇 가지 주요 원칙을 지원합니다.

- 권한 부여 및 API 액세스 제어는 애플리케이션의 여러 지점에 적용됩니다.
- 권한 부여 로직은 애플리케이션과 독립적입니다.
- 액세스 제어 결정은 중앙 집중식으로 이루어집니다.



애플리케이션 흐름 (다이어그램에서 파란색 번호가 매겨진 콜아웃으로 표시됨):

1. JSON 웹 토큰 (JWT) 을 가진 인증된 사용자가 Amazon에 HTTP 요청을 생성합니다. CloudFront
2. CloudFront 요청을 CloudFront 오리진으로 구성된 Amazon API Gateway로 전달합니다.
3. API Gateway 사용자 지정 권한 부여자가 호출되어 JWT를 확인합니다.
4. 마이크로서비스는 요청에 응답합니다.

권한 부여 및 API 액세스 제어 흐름 (다이어그램에서 빨간색 번호가 매겨진 콜아웃으로 표시됨):

1. PEP는 권한 부여 서비스를 호출하고 JWT를 포함한 요청 데이터를 전달합니다.
2. 권한 부여 서비스 (PDP) (이 경우 Verified Permissions) 는 요청 데이터를 쿼리 입력으로 사용하고 쿼리에 지정된 관련 정책을 기반으로 요청 데이터를 평가합니다.
3. 승인 결정은 PEP에 반환되고 평가됩니다.

이 모델은 중앙 집중식 PDP를 사용하여 권한 부여 결정을 내립니다. PEP는 여러 지점에서 구현되어 PDP에 권한 부여 요청을 보냅니다. 다음 다이어그램은 가상의 멀티테넌트 SaaS 애플리케이션에서 이 모델을 구현하는 방법을 보여줍니다.

이 아키텍처에서 PEP는 Amazon CloudFront 및 Amazon API Gateway와 각 마이크로서비스에 대한 서비스 엔드포인트에서 승인 결정을 요청합니다. 승인 결정은 권한 부여 서비스인 Amazon 검증 권한(PDP) 에서 내립니다. 검증된 권한은 완전 관리형 서비스이므로 기본 인프라를 관리할 필요가 없습니다. RESTful API 또는 AWS SDK를 사용하여 검증된 권한과 상호 작용할 수 있습니다.

이 아키텍처를 사용자 지정 정책 엔진과 함께 사용할 수도 있습니다. 하지만 검증된 권한을 통해 얻을 수 있는 이점은 사용자 지정 정책 엔진에서 제공하는 로직으로 대체해야 합니다.

API에 PEP를 사용하는 중앙 집중식 PDP를 사용하면 API에 대한 강력한 권한 부여 시스템을 쉽게 만들 수 있습니다. 이는 권한 부여 프로세스를 간소화하고 API, 마이크로서비스 easy-to-use, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내리는 데 사용할 수 있는 반복 가능한 인터페이스를 제공합니다.

Cedar SDK 사용

Amazon 검증 권한은 Cedar 언어를 사용하여 사용자 지정 애플리케이션의 세분화된 권한을 관리합니다. 검증된 권한을 사용하면 Cedar 정책을 중앙 위치에 저장하고, 밀리초 단위의 처리로 짧은 지연 시간을 활용하고, 다양한 애플리케이션에서 권한을 감사할 수 있습니다. 또한 검증된 권한을 사용하지 않고도 Cedar SDK를 애플리케이션에 직접 통합하여 권한 부여 결정을 내릴 수도 있습니다. 이 옵션을 사용하려면 사용 사례에 맞는 정책을 관리하고 저장하기 위한 추가 사용자 지정 애플리케이션 개발이 필요합니다. 그러나 특히 인터넷 연결이 일관되지 않아 Verified Permissions에 대한 액세스가 간헐적이거나 불가능한 경우에는 실행 가능한 대안이 될 수 있습니다.

OPA용 설계 모델

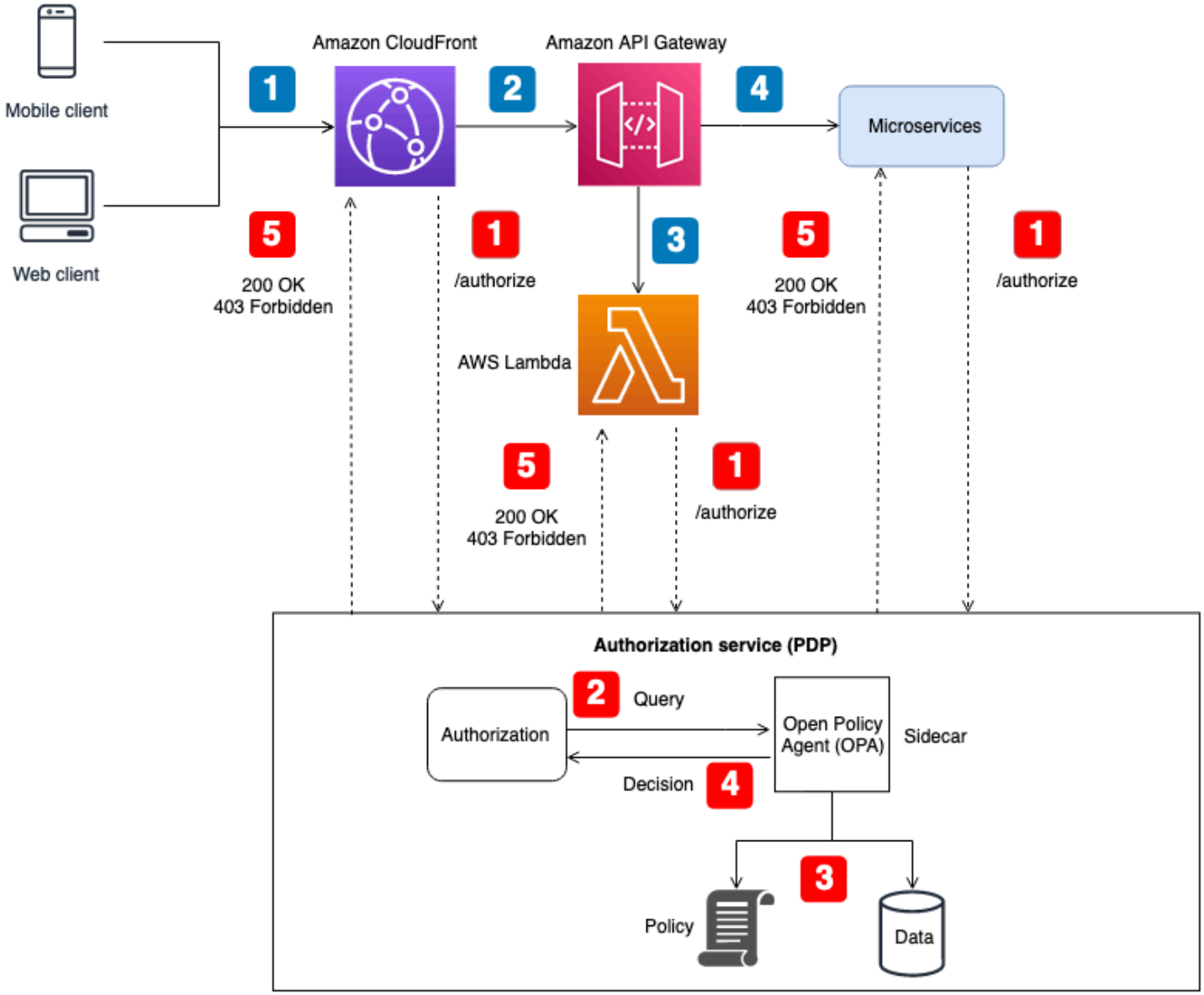
API에 PEP가 포함된 중앙 집중식 PDP 사용

API에 정책 적용 지점 (PEP) 이 포함된 중앙 집중식 정책 결정 지점 (PDP) 모델은 업계 모범 사례를 따라 API 액세스 제어 및 권한 부여를 위한 효과적이고 쉽게 유지 관리할 수 있는 시스템을 만듭니다. 이 접근 방식은 다음과 같은 몇 가지 주요 원칙을 지원합니다.

- 권한 부여 및 API 액세스 제어는 애플리케이션의 여러 지점에 적용됩니다.
- 권한 부여 로직은 애플리케이션과 독립적입니다.

- 액세스 제어 결정은 중앙 집중식으로 이루어집니다.

이 모델은 중앙 집중식 PDP를 사용하여 권한 부여 결정을 내립니다. PEP는 모든 API에 구현되어 PDP에 권한 부여 요청을 보냅니다. 다음 다이어그램은 가상의 멀티테넌트 SaaS 애플리케이션에서 이 모델을 구현하는 방법을 보여줍니다.



애플리케이션 흐름 (다이어그램에서 파란색 번호가 매겨진 콜아웃으로 표시됨):

1. JWT를 사용하는 인증된 사용자는 Amazon에 HTTP 요청을 생성합니다. CloudFront
2. CloudFront 요청을 CloudFront 오리진으로 구성된 Amazon API Gateway로 전달합니다.
3. API Gateway 사용자 지정 권한 부여자가 호출되어 JWT를 확인합니다.

4. 마이크로서비스는 요청에 응답합니다.

권한 부여 및 API 액세스 제어 흐름 (다이어그램에서 빨간색 번호가 매겨진 콜아웃으로 표시됨):

1. PEP는 권한 부여 서비스를 호출하고 JWT를 포함한 요청 데이터를 전달합니다.
2. 권한 부여 서비스 (PDP) 는 요청 데이터를 가져와 사이드카로 실행되는 OPA 에이전트 REST API를 쿼리합니다. 요청 데이터는 쿼리의 입력 역할을 합니다.
3. OPA는 쿼리에 지정된 관련 정책을 기반으로 입력을 평가합니다. 필요한 경우 권한 부여 결정을 내리기 위해 데이터를 가져옵니다.
4. OPA는 승인 서비스에 결정을 반환합니다.
5. 승인 결정은 PEP에 반환되고 평가됩니다.

이 아키텍처에서 PEP는 Amazon CloudFront 및 Amazon API Gateway, 그리고 각 마이크로서비스에 대한 서비스 엔드포인트에서 승인 결정을 요청합니다. 권한 부여 결정은 OPA 사이드카가 있는 권한 부여 서비스 (PDP) 에서 내립니다. 이 권한 부여 서비스를 컨테이너 또는 기존 서버 인스턴스로 운영할 수 있습니다. OPA 사이드카는 RESTful API를 로컬로 노출하므로 권한 부여 서비스에서만 API에 액세스할 수 있습니다. 권한 부여 서비스는 PEP가 사용할 수 있는 별도의 API를 제공합니다. 권한 부여 서비스가 PEP와 OPA 사이에서 중개자 역할을 하도록 하면 PEP와 OPA 간에 필요할 수 있는 모든 변환 로직을 PEP와 OPA 간에 삽입할 수 있습니다. 예를 들어 PEP의 권한 부여 요청이 OPA에서 예상한 쿼리 입력과 일치하지 않는 경우.

이 아키텍처를 사용자 지정 정책 엔진과 함께 사용할 수도 있습니다. 하지만 OPA를 통해 얻을 수 있는 모든 이점을 사용자 지정 정책 엔진에서 제공하는 로직으로 대체해야 합니다.

API에 PEP를 사용하는 중앙 집중식 PDP를 사용하면 API에 대한 강력한 권한 부여 시스템을 쉽게 만들 수 있습니다. 구현이 간단하고 API, 마이크로서비스 easy-to-use, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내리는 데 사용할 수 있는 반복 가능한 인터페이스도 제공합니다. 그러나 권한 부여 결정에는 별도의 API 호출이 필요하기 때문에 이 접근 방식을 사용하면 애플리케이션에 너무 많은 지연이 발생할 수 있습니다. 네트워크 지연 시간이 문제라면 분산형 PDP를 고려해 볼 수 있습니다.

API에서 PEP가 포함된 분산 PDP 사용

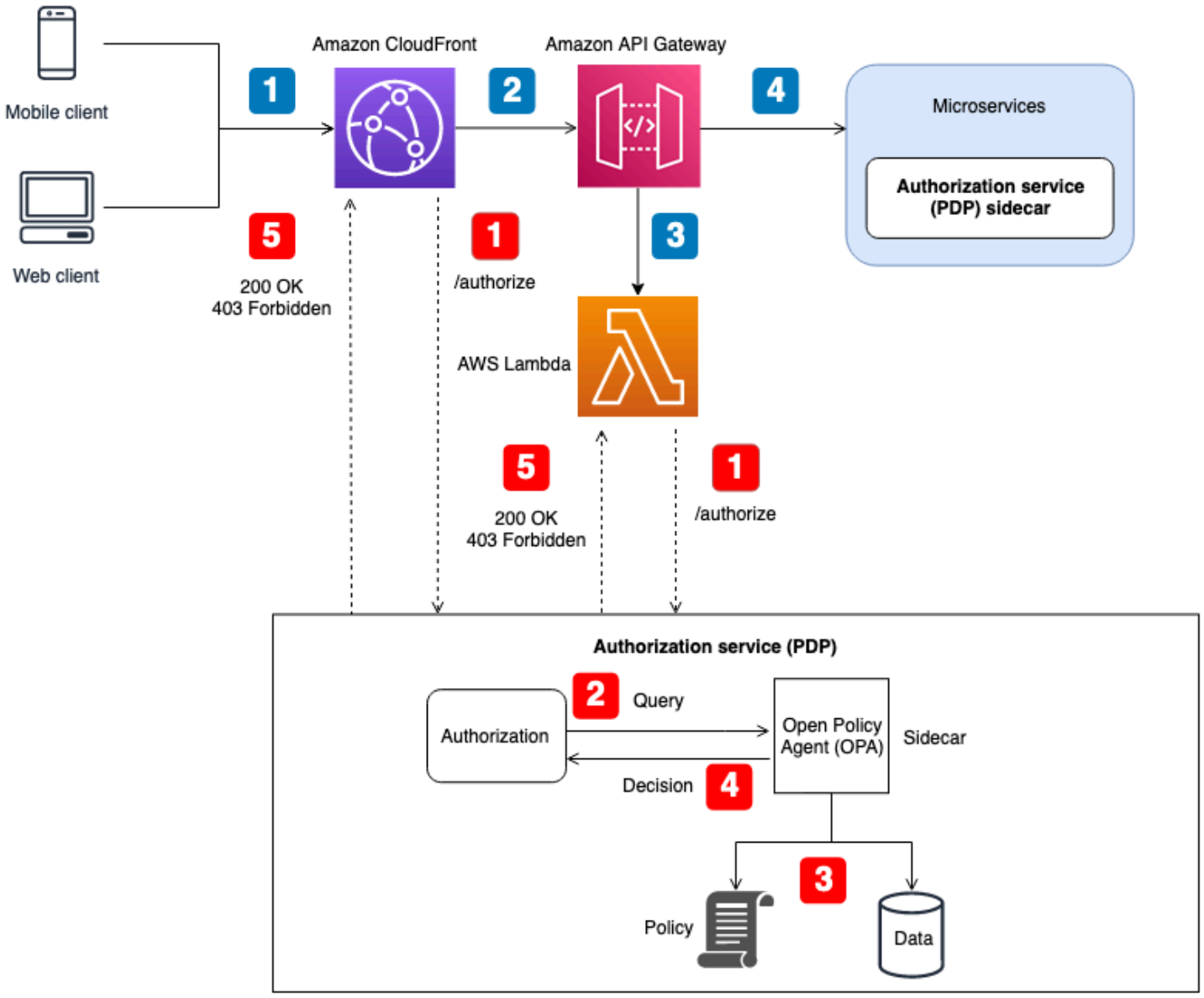
API에 정책 적용 지점 (PEP) 이 포함된 분산 정책 결정 지점 (PDP) 모델은 업계 모범 사례를 따라 API 액세스 제어 및 권한 부여를 위한 효과적인 시스템을 구축합니다. API 기반 PEP를 사용하는 중앙 집중식 PDP 모델과 마찬가지로 이 접근 방식은 다음과 같은 주요 원칙을 지원합니다.

- 권한 부여 및 API 액세스 제어는 애플리케이션의 여러 지점에 적용됩니다.
- 권한 부여 로직은 애플리케이션과 독립적입니다.
- 액세스 제어 결정은 중앙 집중식으로 이루어집니다.

PDP가 배포될 때 액세스 제어 결정이 중앙 집중화되는 이유가 궁금할 것입니다. PDP가 응용 프로그램의 여러 위치에 있을 수 있지만 액세스 제어 결정을 내리려면 동일한 권한 부여 논리를 사용해야 합니다. 모든 PDP는 동일한 입력이 주어지면 동일한 액세스 제어 결정을 제공합니다. PEP는 모든 API에 구현되어 PDP에 권한 부여 요청을 보냅니다. 다음 그림은 가상의 멀티테넌트 SaaS 애플리케이션에서 이 분산 모델을 구현하는 방법을 보여줍니다.

이 접근 방식에서는 PDP를 애플리케이션의 여러 위치에 구현합니다. 사이드카를 사용하는 컨테이너식 서비스 또는 Amazon Elastic Compute Cloud (Amazon EC2) 인스턴스와 같이 OPA를 실행하고 PDP를 지원하는 온보드 컴퓨팅 기능을 갖춘 애플리케이션 구성 요소의 경우 중앙 집중식 PDP 서비스에 RESTful API를 호출하지 않고도 PDP 결정을 애플리케이션 구성 요소에 직접 통합할 수 있습니다. 이렇게 하면 중앙 집중식 PDP 모델에서 발생할 수 있는 지연 시간을 줄일 수 있다는 이점이 있습니다. 모든 애플리케이션 구성 요소가 권한 부여 결정을 내리기 위해 추가 API 호출을 수행할 필요는 없기 때문입니다. 그러나 Amazon 또는 Amazon CloudFront API Gateway 서비스와 같이 PDP를 직접 통합할 수 있는 온보드 컴퓨팅 기능이 없는 애플리케이션 구성 요소의 경우 이 모델에서 중앙 집중식 PDP가 여전히 필요합니다.

다음 다이어그램은 가상의 멀티테넌트 SaaS 애플리케이션에서 중앙 집중식 PDP와 분산 PDP의 조합을 구현하는 방법을 보여줍니다.



애플리케이션 흐름 (다이어그램에서 파란색 번호가 매겨진 콜아웃으로 표시됨):

1. JWT를 사용하는 인증된 사용자는 Amazon에 HTTP 요청을 생성합니다. CloudFront
2. CloudFront 요청을 CloudFront 오리진으로 구성된 Amazon API Gateway로 전달합니다.
3. API Gateway 사용자 지정 권한 부여자가 호출되어 JWT를 확인합니다.
4. 마이크로서비스는 요청에 응답합니다.

권한 부여 및 API 액세스 제어 흐름 (다이어그램에서 빨간색 번호가 매겨진 콜아웃으로 표시됨):

1. PEP는 권한 부여 서비스를 호출하고 JWT를 포함한 요청 데이터를 전달합니다.

2. 권한 부여 서비스 (PDP) 는 요청 데이터를 가져와 사이드카로 실행되는 OPA 에이전트 REST API를 쿼리합니다. 요청 데이터는 쿼리의 입력 역할을 합니다.
3. OPA는 쿼리에 지정된 관련 정책을 기반으로 입력을 평가합니다. 필요한 경우 권한 부여 결정을 내리기 위해 데이터를 가져옵니다.
4. OPA는 승인 서비스에 결정을 반환합니다.
5. 승인 결정은 PEP에 반환되고 평가됩니다.

이 아키텍처에서 PEP는 API Gateway의 서비스 엔드포인트와 각 마이크로서비스에 대한 CloudFront 권한 부여 결정을 요청합니다. 마이크로서비스에 대한 권한 부여 결정은 애플리케이션 구성 요소와 함께 사이드카 역할을 하는 권한 부여 서비스 (PDP) 에 의해 이루어집니다. 이 모델은 컨테이너 또는 Amazon Elastic Compute Cloud (Amazon EC2) 인스턴스에서 실행되는 마이크로서비스 (또는 서비스) 에 사용할 수 있습니다. API Gateway와 CloudFront 같은 서비스에 대한 권한 부여 결정은 여전히 외부 권한 부여 서비스에 문의해야 합니다. 그럼에도 불구하고 권한 부여 서비스는 PEP가 사용할 수 있는 API를 공개합니다. 권한 부여 서비스가 PEP와 OPA 사이에서 중개자 역할을 하도록 하면 PEP와 OPA 간에 필요할 수 있는 모든 변환 로직을 PEP와 OPA 간에 삽입할 수 있습니다. 예를 들어 PEP의 권한 부여 요청이 OPA에서 예상한 쿼리 입력과 일치하지 않는 경우.

이 아키텍처를 사용자 지정 정책 엔진과 함께 사용할 수도 있습니다. 하지만 OPA를 통해 얻을 수 있는 모든 이점을 사용자 지정 정책 엔진에서 제공하는 로직으로 대체해야 합니다.

API에 PEP를 사용하는 분산 PDP는 API에 대한 강력한 권한 부여 시스템을 만들 수 있는 옵션을 제공합니다. 구현이 간단하고 API, 마이크로서비스 easy-to-use, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내리는 데 사용할 수 있는 반복 가능한 인터페이스를 제공합니다. 또한 이 접근 방식은 중앙 집중식 PDP 모델에서 발생할 수 있는 지연 시간을 줄일 수 있다는 이점도 있습니다.

분산 PDP를 라이브러리로 사용

응용 프로그램 내에서 사용할 수 있도록 라이브러리 또는 패키지로 제공되는 PDP에 권한 부여 결정을 요청할 수도 있습니다. OPA는 Go 타사 라이브러리로 사용할 수 있습니다. 다른 프로그래밍 언어의 경우 일반적으로 이 모델을 채택하면 사용자 지정 정책 엔진을 만들어야 합니다.

Amazon 검증 권한 멀티 테넌트 설계 고려 사항

멀티테넌트 SaaS 솔루션에서 Amazon 검증 권한을 사용하여 권한 부여를 구현할 때는 몇 가지 설계 옵션을 고려해야 합니다. 이러한 옵션을 살펴보기 전에 멀티테넌트 SaaS 컨텍스트에서 격리와 권한 부여의 차이점을 명확히 설명하겠습니다. 테넌트를 격리하면 인바운드 및 아웃바운드 데이터가 잘못된 테넌트에 노출되는 것을 방지할 수 있습니다. 권한 부여는 사용자가 테넌트에 액세스할 수 있는 권한을 갖도록 합니다.

검증된 권한에서 정책은 정책 저장소에 저장됩니다. Verified Permissions 설명서에 설명된 대로 각 테넌트에 대해 별도의 정책 저장소를 사용하여 테넌트의 정책을 격리하거나 모든 테넌트에 대해 단일 정책 저장소를 사용하여 테넌트가 정책을 공유하도록 허용할 수 있습니다. 이 섹션에서는 이러한 두 가지 격리 전략의 장단점을 설명하고 계층형 배포 모델을 사용하여 배포할 수 있는 방법을 설명합니다. 추가 컨텍스트는 검증된 권한 설명서를 참조하십시오.

이 섹션에서 설명하는 기준은 검증된 권한에 초점을 맞추고 있지만, 일반적인 개념은 격리 사고방식과 이 사고방식이 제공하는 지침에 기반을 두고 있습니다. SaaS 애플리케이션은 항상 테넌트 격리를 설계의 일부로 고려해야 하며, 이러한 일반 격리 원칙은 SaaS 애플리케이션에 Verified Permission을 포함하는 데까지 확장됩니다. 이 섹션에서는 사일로화된 SaaS 모델 및 풀링된 SaaS 모델과 같은 핵심 SaaS 격리 모델도 참조합니다. 자세한 내용은 AWS Well-Architected 프레임워크인 SaaS Lens의 핵심 격리 개념을 참조하십시오.

멀티테넌트 SaaS 솔루션을 설계할 때 주요 고려 사항은 테넌트 격리 및 테넌트 온보딩입니다. 테넌트 격리는 보안, 개인 정보 보호, 탄력성 및 성능에 영향을 미칩니다. 테넌트 온보딩은 운영 오버헤드 및 관찰 가능성과 관련하여 운영 프로세스에 영향을 미칩니다. SaaS 여정을 진행하거나 멀티테넌트 솔루션을 구현하는 조직은 항상 SaaS 애플리케이션에서 테넌시를 처리하는 방법의 우선 순위를 정해야 합니다. SaaS 솔루션이 특정 격리 모델에 치우칠 수 있지만 전체 SaaS 솔루션에서 일관성이 반드시 필요한 것은 아닙니다. 예를 들어 애플리케이션의 프런트엔드 구성 요소에 대해 선택한 격리 모델은 마이크로서비스나 권한 부여 서비스에 대해 선택한 격리 모델과 동일하지 않을 수 있습니다.

설계 고려 사항:

- 테넌트 온보딩 및 사용자 테넌트 등록
- 테넌트별 정책 저장소
- 공유 멀티테넌트 정책 스토어 1개
- 계층형 배포 모델

테넌트 온보딩 및 사용자 테넌트 등록

SaaS 애플리케이션은 SaaS ID의 개념을 준수하고 사용자 ID를 테넌트 ID에 바인딩하는 일반적인 모범 사례를 따릅니다. 바인딩에는 테넌트 식별자를 ID 공급자의 사용자에 대한 클레임 또는 속성으로 저장하는 작업이 포함됩니다. 이렇게 하면 각 애플리케이션에서 테넌트에 ID를 매핑하는 책임이 사용자 등록 프로세스로 이전됩니다. 그러면 인증된 각 사용자는 JWT (JSON 웹 토큰) 의 일부로서 올바른 테넌트 ID를 갖게 됩니다.

마찬가지로 권한 부여 요청에 대한 올바른 정책 저장소의 선택은 애플리케이션 로직에 의해 결정되어서는 안 됩니다. 특정 권한 부여 요청에서 사용해야 하는 정책 저장소를 결정하려면 사용자를 정책 저장소에 매핑하거나 테넌트를 정책 저장소에 매핑하십시오. 이러한 매핑은 일반적으로 애플리케이션이 참조하는 Amazon DynamoDB 또는 Amazon RDS (Amazon RDS) 와 같은 데이터 스토어에서 유지 관리됩니다. ID 공급자 (IdP) 의 데이터로 이러한 매핑을 제공하거나 보완할 수도 있습니다. 그러면 일반적으로 권한 부여 요청에 필요한 모든 관계가 포함된 JWT를 통해 테넌트, 사용자 및 정책 저장소 간의 관계가 사용자에게 제공됩니다.

이 예제는 테넌트에 `TenantA` 속하고 권한 부여를 위해 정책 저장소 `ps-43214321` ID가 있는 정책 저장소를 사용하는 사용자에게 `Alice` JWT가 어떻게 표시되는지를 보여줍니다.

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

테넌트별 정책 저장소

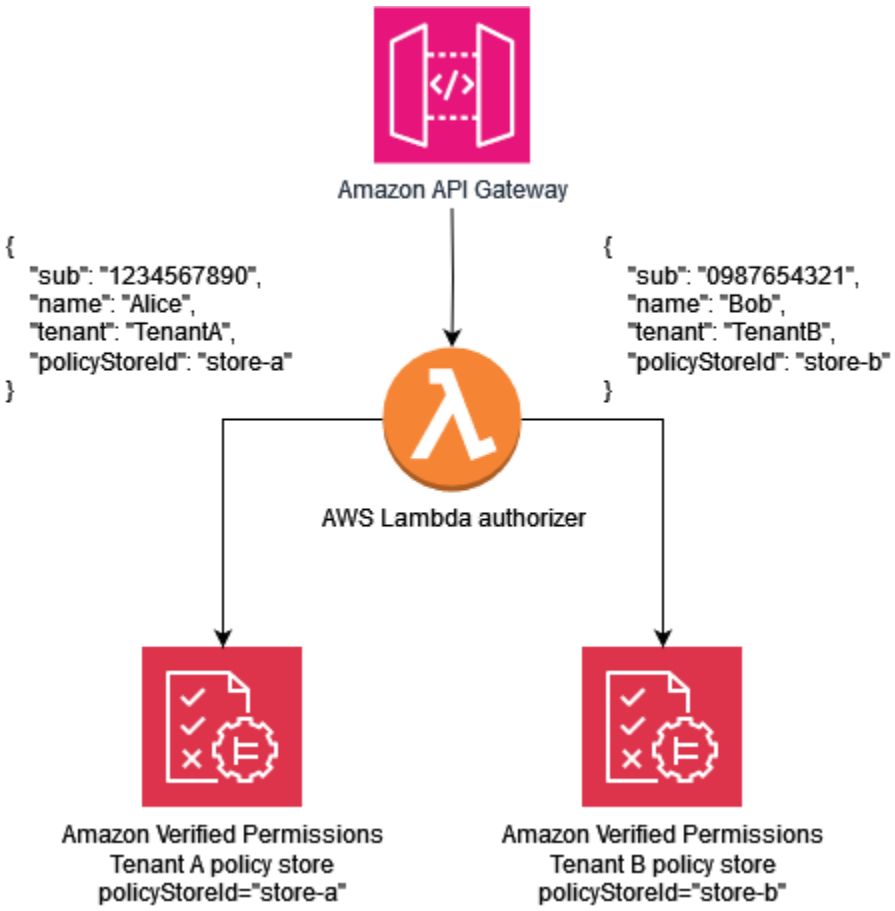
Amazon Verified Permissions의 테넌트별 정책 스토어 설계 모델은 SaaS 애플리케이션의 각 테넌트를 자체 정책 스토어와 연결합니다. 이 모델은 SaaS 사일로 격리 모델과 유사합니다. 두 모델 모두 테넌트별 인프라 생성을 의무화하고 있으며 장점과 단점이 비슷합니다. 이 접근 방식의 주요 이점은 인프라를 통한 테넌트 격리, 테넌트별 고유 권한 부여 모델 지원, 시끄러운 이웃 문제 제거, 정책 업데이트 또는 배포 실패의 영향 범위 감소 등입니다. 이 접근 방식의 단점으로는 더 복잡한 테넌트 온보딩 프로세스, 배포 및 운영이 있습니다. 솔루션에 테넌트별로 고유한 정책이 있는 경우 테넌트별 정책 저장소가 권장되는 접근 방식입니다.

테넌트별 정책 저장소 모델은 SaaS 애플리케이션에 필요한 경우 테넌트 격리에 대한 고도로 사일로화된 접근 방식을 제공할 수 있습니다. 이 모델을 풀 격리와 함께 사용할 수도 있지만 Verified

Permissions 구현에서는 관리 및 운영 단순화와 같은 광범위한 풀 격리 모델의 표준적 이점을 공유할 수 없습니다.

테넌트별 정책 저장소에서는 앞서 설명한 대로 사용자 등록 프로세스 중에 테넌트의 정책 저장소 식별자를 사용자의 SaaS ID에 매핑하여 테넌트를 격리합니다. 이 접근 방식은 테넌트의 정책 저장소를 사용자 주체와 강력하게 연결하고 전체 SaaS 솔루션에서 매핑을 공유하는 일관된 방법을 제공합니다. SaaS 애플리케이션을 IdP의 일부로 유지 관리하거나 DynamoDB와 같은 외부 데이터 소스에서 유지 관리하여 SaaS 애플리케이션에 매핑을 제공할 수 있습니다. 또한 이렇게 하면 보안 주체가 테넌트의 일부가 되고 테넌트의 정책 저장소가 사용되도록 할 수 있습니다.

이 예제에서는 사용자의 손을 포함하는 JWT가 API 엔드포인트에서 권한 `tenant` 부여자의 정책 평가 지점으로 전달되는 방식을 보여줍니다. AWS Lambda 권한 부여자는 요청을 올바른 정책 저장소로 라우팅합니다. `policyStoreId`



다음 샘플 정책은 테넌트별 정책 스토어 설계 패러다임을 보여줍니다. 또한 `TenantA`. The policyStoreId `store-a` 에 `Alice` 속하는 사용자는 테넌트 ID에 `Alice`, 매핑되어 올바른 정책 저장소를 사용하도록 강제합니다. 이렇게 하면 의 정책이 사용될 수 `TenantA` 있습니다.

> **Note**
>
> 테넌트별 정책 저장소 모델은 테넌트의 정책을 분리합니다. 권한 부여는 사용자가 데이터에 대해 수행할 수 있는 작업을 적용합니다. 이 모델을 사용하는 가상 애플리케이션과 관련된 리소스는 AWS Well-Architected Framework, SaaS Lens 설명서에 정의된 대로 다른 격리 메커니즘을 사용하여 격리해야 합니다.

이 정책에서 `Alice` 는 모든 리소스의 데이터를 볼 수 있는 권한을 가집니다.

```
permit (
    principal == MultiTenantApp::User::"Alice",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

권한 부여를 요청하고 Verified Permissions 정책으로 평가를 시작하려면 테넌트에 매핑된 고유 ID에 해당하는 정책 저장소 ID를 제공해야 합니다. `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":[]
            }
        ]
    ]
  }
}
```

사용자는 테넌트 B에 Bob 속하며 테넌트 ID에도 매핑되므로 올바른 정책 저장소를 사용해야 합니다. policyStoreId `store-b` Bob 이렇게 하면 테넌트 B의 정책을 사용할 수 있습니다.

이 Bob 정책에는 모든 리소스의 데이터를 사용자 지정할 수 있는 권한이 있습니다. 이 예에서는 테넌트 B에만 적용되는 `customizeData` 작업일 수 있으므로 정책은 테넌트 B에만 고유합니다. 테넌트별 정책 저장소 모델은 기본적으로 테넌트별로 사용자 지정 정책을 지원합니다.

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

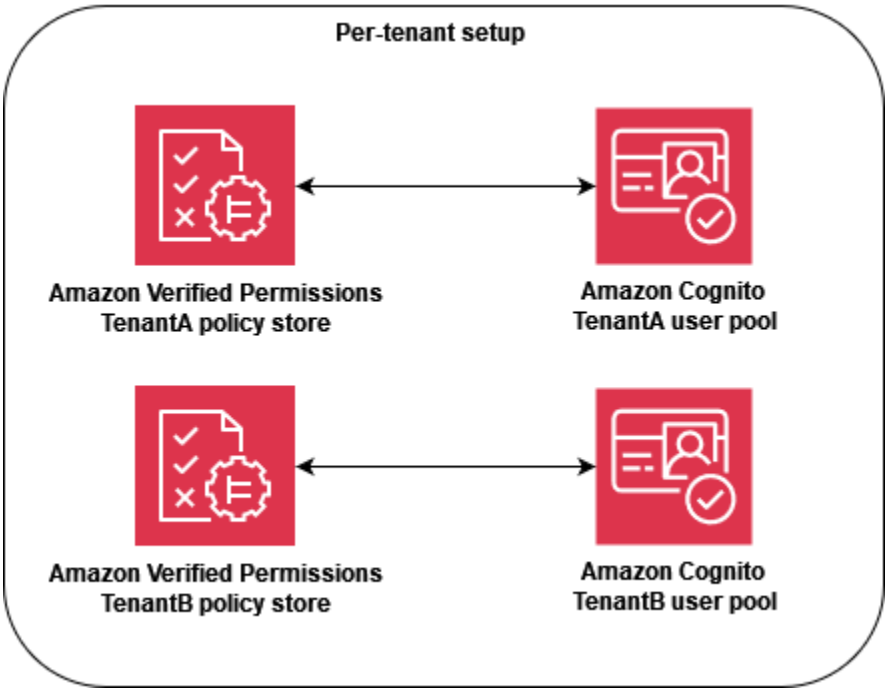
권한 부여를 요청하고 Verified Permissions 정책을 사용하여 평가를 시작하려면 테넌트에 매핑된 고유 ID에 해당하는 정책 저장소 ID를 제공해야 합니다. `store-b`

```
{
   "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":[]
             }
          ]
       ]
    }
}
```

검증된 권한을 사용하면 IdP를 정책 저장소와 통합할 수 있지만 필수는 아닙니다. 이 통합을 통해 정책에서는 ID 저장소의 보안 주체를 정책의 보안 주체로 명시적으로 참조할 수 있습니다. 검증된 권한을 위해 Amazon Cognito와 IdP로 통합하는 방법에 대한 자세한 내용은 검증된 권한 설명서 및 Amazon Cognito 설명서를 참조하십시오.

정책 저장소를 IdP와 통합하는 경우 정책 저장소당 하나의 ID 소스만 사용할 수 있습니다. 예를 들어 검증된 권한을 Amazon Cognito와 통합하기로 선택한 경우 검증된 권한 정책 스토어와 Amazon Cognito 사용자 풀의 테넌트 격리에 사용되는 전략을 미러링해야 합니다. 정책 스토어와 사용자 풀도 같은 위치에 있어야 합니다. AWS 계정



운영 수준에서 테넌트별 정책 저장소는 각 테넌트에 대해AWS CloudTrail 개별적으로 로그인한 활동을 쉽게 쿼리할 수 있으므로 감사 이점이 있습니다. 하지만 여전히 테넌트별 차원에 대한 추가 사용자 지정 지표를 CloudWatch Amazon에 기록하는 것이 좋습니다.

또한 테넌트별 정책 저장소 접근 방식에서는 SaaS 솔루션 운영에 방해가 되지 않도록 두 개의 검증된 권한 할당량에 세심한 주의를 기울여야 합니다. 이러한 할당량은 계정별 지역별 정책 저장소와 계정별 지역별 초당 IsAuthorized 요청입니다. 두 할당량 모두에 대한 증가를 요청할 수 있습니다.

테넌트별 정책 스토어 모델을 구현하는 방법에 대한 자세한 예는 테넌트별 정책 스토어와 함께 Amazon Verified Permissions를 사용한 SaaS 액세스 제어 AWS 블로그 게시물을 참조하십시오.

공유 멀티테넌트 정책 스토어 1개

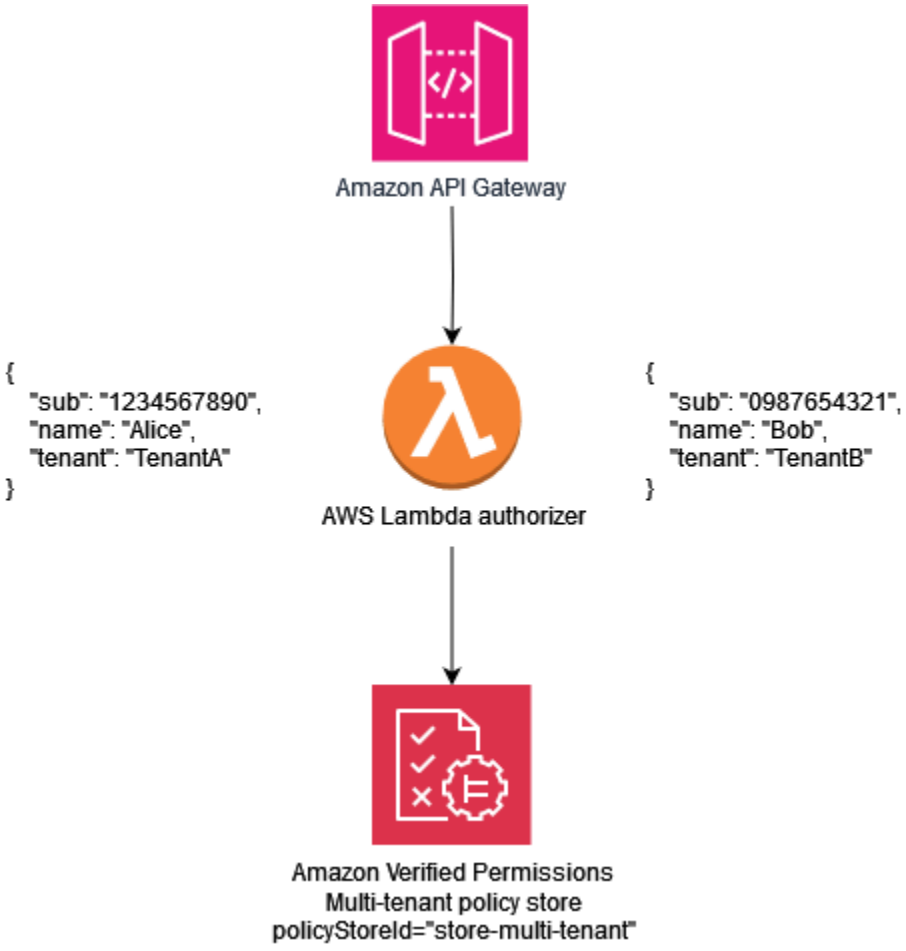
단일 공유 멀티테넌트 정책 스토어 설계 모델은 SaaS 솔루션의 모든 테넌트에 대해 Amazon Verified Permissions의 단일 멀티테넌트 정책 스토어를 사용합니다. 이 접근 방식의 주요 이점은 관리 및 운영이 단순하다는 것입니다. 특히 테넌트 온보딩 중에 추가 정책 저장소를 생성할 필요가 없기 때문입니다. 이 접근 방식의 단점은 정책 업데이트 또는 배포의 실패 또는 실수로 인한 영향의 범위가 확대되고 노이즈가 발생하는 이웃 영향에 대한 노출이 커진다는 점입니다. 또한 솔루션에 테넌트별로 고유한 정책이 필요한 경우에는 이 방법을 사용하지 않는 것이 좋습니다. 이 경우 올바른 테넌트의 정책이 사용되도록 하려면 테넌트별 정책 저장소 모델을 대신 사용하십시오.

단일 공유 멀티테넌트 정책 저장소 접근 방식은 SaaS 풀링 격리 모델과 유사합니다. SaaS 애플리케이션에 필요한 경우 테넌트 격리에 대한 풀링된 접근 방식을 제공할 수 있습니다. SaaS 솔루션이 마이

크로서비스에 사일로 격리를 적용하는 경우에도 이 모델을 사용할 수 있습니다. 모델을 선택할 때는 SaaS 애플리케이션에 필요한 테넌트 데이터 격리 요구 사항과 검증된 권한 정책의 구조를 독립적으로 평가해야 합니다.

전체 SaaS 솔루션에서 테넌트 식별자를 공유하는 일관된 방법을 적용하려면 앞서 설명한 것처럼 사용자 등록 중에 식별자를 사용자의 SaaS ID에 매핑하는 것이 좋습니다. 이 매핑을 IdP의 일부로 유지 관리하거나 DynamoDB와 같은 외부 데이터 소스에서 유지 관리하여 SaaS 애플리케이션에 제공할 수 있습니다. 또한 공유 정책 저장소 ID를 사용자에게 매핑하는 것이 좋습니다. ID는 테넌트 격리의 일부로 사용되지는 않지만 향후 변경을 용이하게 하기 때문에 좋은 방법입니다.

다음 예제는 API 엔드포인트가 다른 테넌트에 속하지만 권한 부여를 위해 정책 저장소 `Alice store-multi-tenant` ID와 Bob 정책 저장소를 공유하는 사용자와 사용자에게 JWT를 보내는 방법을 보여줍니다. 모든 테넌트가 단일 정책 저장소를 공유하므로 토큰이나 데이터베이스의 정책 저장소 ID를 유지할 필요가 없습니다. 모든 테넌트가 단일 정책 저장소 ID를 공유하므로 응용 프로그램이 정책 저장소를 호출하는 데 사용할 수 있는 환경 변수로 ID를 제공할 수 있습니다.



다음 샘플 정책은 단일 공유 멀티테넌트 정책 설계 패러다임을 보여줍니다. 이 정책에서는 상위 주체가 `MultiTenantApp::User` 있는 보안 주체가 모든 리소스의 데이터를 볼 수 있는 권한을 `MultiTenantApp::Role` `Admin` 갖습니다.

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

단일 정책 저장소가 사용 중이므로 Verified Permissions 정책 저장소는 보안 주체와 연결된 테넌시 속성이 리소스와 연결된 테넌시 속성과 일치하는지 확인해야 합니다. 이를 위해서는 정책 저장소에 다음 정책을 포함하여 리소스와 보안 주체의 테넌시 속성이 일치하지 않는 모든 권한 부여 요청이 거부되도록 할 수 있습니다.

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

단일 공유 멀티테넌트 정책 저장소 모델을 사용하는 권한 부여 요청의 경우 정책 저장소 ID는 공유 정책 저장소의 식별자입니다. 다음 요청에서는 가 /를 가지고 있기 때문에 `User` `Alice` 액세스가 허용되며`Admin`, 리소스 및 보안 주체와 관련된 `Tenant` 속성은 둘 `TenantA` 다입니다. `Role`

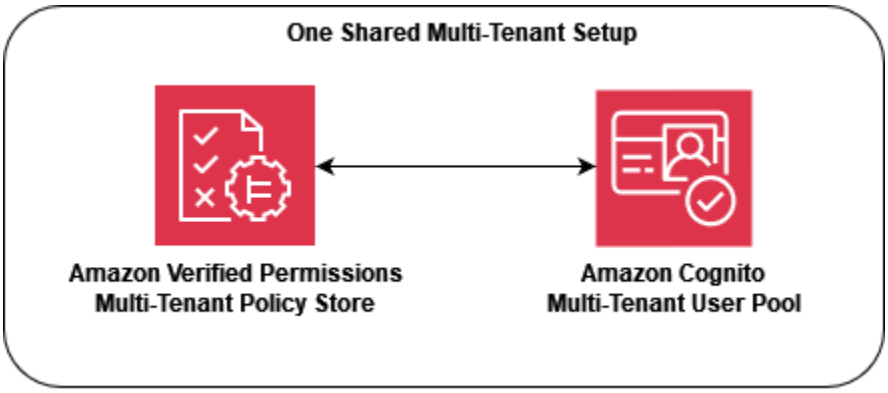
```
{
   "policyStoreId":"store-multi-tenant",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
```

```
    },
    "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":[]
          }
       ]
    }
```

```
}
```

검증된 권한을 사용하면 IdP를 정책 저장소와 통합할 수 있지만 필수는 아닙니다. 이 통합을 통해 정책에서는 ID 저장소의 보안 주체를 정책의 보안 주체로 명시적으로 참조할 수 있습니다. 검증된 권한을 위해 Amazon Cognito와 IdP로 통합하는 방법에 대한 자세한 내용은 검증된 권한 설명서 및 Amazon Cognito 설명서를 참조하십시오.

정책 저장소를 IdP와 통합하는 경우 정책 저장소당 하나의 ID 소스만 사용할 수 있습니다. 예를 들어 검증된 권한을 Amazon Cognito와 통합하기로 선택한 경우 검증된 권한 정책 스토어와 Amazon Cognito 사용자 풀의 테넌트 격리에 사용되는 전략을 미러링해야 합니다. 정책 스토어와 사용자 풀도 같은 위치에 있어야 합니다. AWS 계정



운영 및 감사 관점에서 볼 때 단일 공유 멀티 테넌트 정책 스토어 모델은 로그인한 각 CloudTrail 호출이 동일한 정책 저장소를 사용하기 때문에 테넌트의 개별 활동을 필터링하기 위해 더 복잡한 쿼리가 AWS CloudTrail필요하다는 단점이 있습니다. 이 시나리오에서는 적절한 수준의 관찰 가능성 및 감사 기능을 보장하기 CloudWatch 위해 테넌트별 차원에 대한 추가 사용자 지정 지표를 Amazon에 기록하는 것이 좋습니다.

단일 공유 멀티테넌트 정책 저장소 접근 방식을 사용하려면 검증된 권한 할당량이 SaaS 솔루션 운영에 지장을 주지 않도록 세심한 주의를 기울여야 합니다. 특히 계정 할당량별 지역별 초당 IsAuthorized 요청 수를 모니터링하여 한도를 초과하지 않는지 확인하는 것이 좋습니다. 이 할당량을 늘리도록 요청할 수 있습니다.

계층형 배포 모델

계층형 배포 모델을 생성하면 우선 순위가 높은 “엔터프라이즈 티어” 테넌트를 잠재적으로 더 많은 수의 “스탠다드 티어” 고객으로부터 격리할 수 있습니다. 이 모델에서는 정책 저장소의 정책에 배포된 모든 변경 내용을 계층별로 별도로 롤아웃하여 각 고객 계층을 해당 계층 외부에서 변경한 내용으로부터 격리할 수 있습니다. 계층형 배포 모델에서 정책 저장소는 일반적으로 테넌트가 온보딩될 때 배포되는 대신 각 계층에 대한 초기 인프라 프로비저닝의 일부로 생성됩니다.

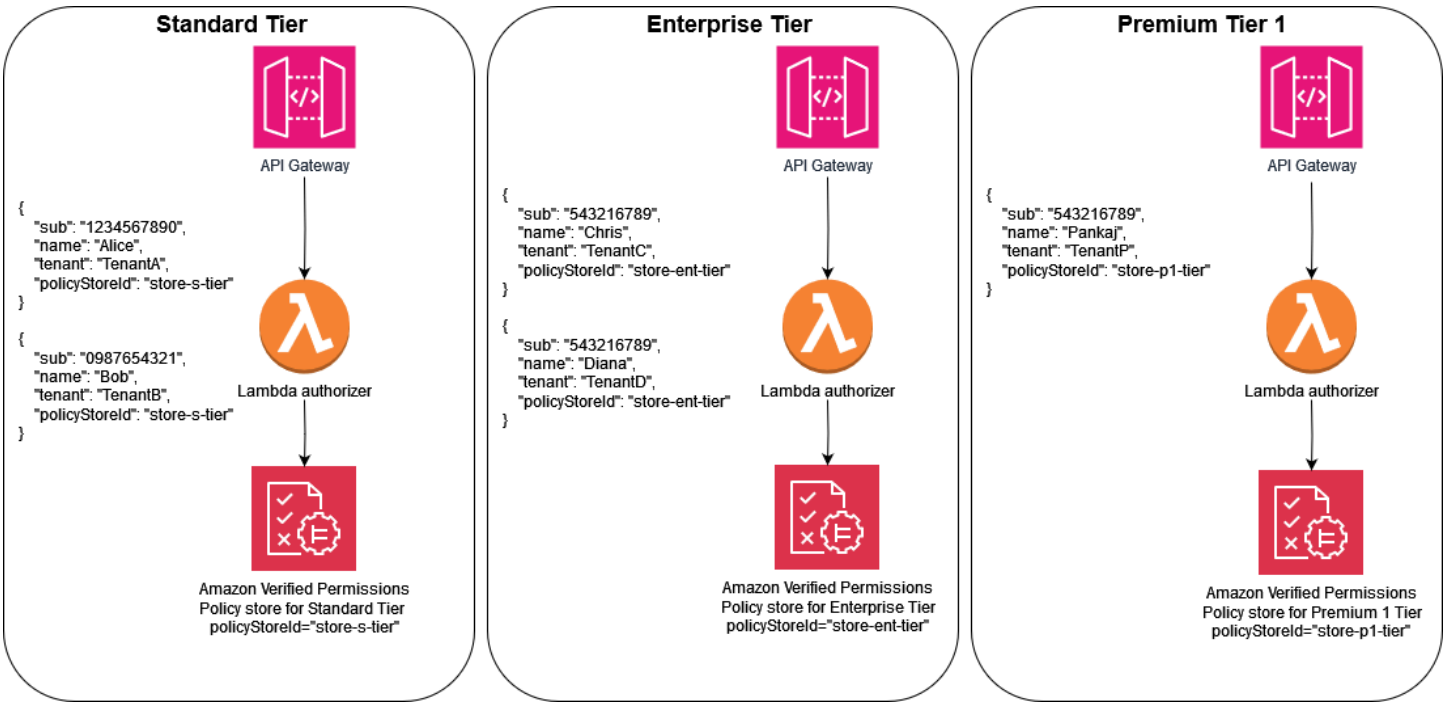
솔루션이 주로 풀링된 격리 모델을 사용하는 경우 추가 격리 또는 사용자 지정이 필요할 수 있습니다. 예를 들어, 각 테넌트가 자체 테넌트 계층 인프라를 갖게 되는 "프리미엄 등급"을 생성할 수 있습니다. 그러면 테넌트가 하나뿐인 풀링된 인스턴스를 배포하여 사일로화된 모델을 만들 수 있습니다. 이는 정책 저장소를 포함하여 완전히 분리된 "프리미엄 계층 테넌트 A"와 "프리미엄 계층 테넌트 B" 인프라의 형태를 취할 수 있습니다. 이 접근 방식을 통해 최고 수준의 고객을 위한 사일로화된 격리 모델이 만들어집니다.

계층형 배포 모델에서는 각 정책 저장소가 별도로 배포되기는 하지만 동일한 격리 모델을 따라야 합니다. 여러 정책 저장소가 사용되므로 전체 SaaS 솔루션에서 테넌트와 연결된 정책 저장소 식별자를 공유하는 일관된 방법을 적용해야 합니다. 테넌트별 정책 저장소 모델과 마찬가지로 사용자 등록 중에 테넌트 식별자를 사용자의 SaaS ID에 매핑하는 것이 좋습니다.

다음 다이어그램은 세 가지 계층 (`Standard Tier`, `Enterprise Tier` 및) 을 보여줍니다. `Premium Tier 1` 각 계층은 자체 인프라에 개별적으로 배포되며 계층 내에서 공유 정책 저장소 하나를 사용합니다. 표준 및 엔터프라이즈 등급에는 여러 테넌트가 포함됩니다. `TenantA`및 `TenantB` 은(는)`Standard Tier`, `TenantC` 그리고 엔터프라이즈 `TenantD` 등급에 속합니다.

`Premium Tier 1`만 `TenantP` 포함하므로 솔루션에 완전히 격리된 격리 모델이 있는 것처럼 프리미엄 테넌트에 서비스를 제공하고 사용자 지정 정책과 같은 기능을 제공할 수 있습니다. 새로운 프리미엄 등급 고객을 온보딩하면 인프라가 구축됩니다. `Premium Tier 2`

> **Note**
>
> 프리미엄 등급의 애플리케이션, 배포 및 테넌트 온보딩은 표준 및 엔터프라이즈 계층과 동일합니다. 유일한 차이점은 프리미엄 티어 온보딩 워크플로우는 새 티어 인프라의 프로비저닝으로 시작된다는 점입니다.


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 멀티테넌트 설계 고려 사항

개방형 정책 에이전트 (OPA) 는 애플리케이션이 정책 및 권한 부여 결정을 내려야 하는 다양한 사용 사례에 적용할 수 있는 유연한 서비스입니다. 멀티테넌트 SaaS 애플리케이션과 함께 OPA를 사용하려면 테넌트 격리와 같은 주요 SaaS 모범 사례가 OPA 구현의 일부로 유지되도록 고유한 기준을 고려해야 합니다. 이러한 기준에는 OPA 배포 패턴, 테넌트 격리 및 OPA 문서 모델, 테넌트 온보딩이 포함됩니다. 이들 각각은 멀티테넌트 애플리케이션과 관련된 OPA의 최적 설계에 영향을 미칩니다.

이 섹션의 설명에서는 OPA에 초점을 맞추고 있지만 일반적인 개념은 격리 사고방식과 OPA가 제공하는 지침에 기반을 두고 있습니다. SaaS 애플리케이션은 항상 테넌트 격리를 설계의 일부로 고려해야 하며, 이러한 일반 격리 원칙은 SaaS 애플리케이션에 OPA를 포함하는 것까지 확장됩니다. OPA를 적절하게 사용하면 SaaS 애플리케이션에서 격리를 적용하는 방식의 핵심 부분이 될 수 있습니다. 이 섹션에서는 사일로화된 SaaS 모델 및 풀링된 SaaS 모델과 같은 핵심 SaaS 격리 모델도 참조합니다. 자세한 내용은 AWS Well-Architected 프레임워크인 SaaS Lens의 핵심 격리 개념을 참조하십시오.

설계 고려 사항:

- 중앙 집중식 및 분산 배포 패턴 비교
- OPA 문서 모델을 사용한 테넌트 격리
- 테넌트 온보딩

중앙 집중식 및 분산 배포 패턴 비교

OPA는 중앙 집중식 또는 분산 배포 패턴으로 배포할 수 있으며 멀티테넌트 애플리케이션에 적합한 방법은 사용 사례에 따라 다릅니다. 이러한 패턴의 예는 이 가이드 앞부분의 API에서 PEP와 함께 중앙 집중식 PDP 사용 및 API에서 분산 PDP 및 PEP 사용 섹션을 참조하십시오. OPA는 운영 체제 또는 컨테이너에 데몬으로 배포할 수 있으므로 멀티테넌트 응용 프로그램을 지원하기 위해 여러 가지 방법으로 구현할 수 있습니다.

중앙 집중식 배포 패턴에서 OPA는 애플리케이션의 다른 서비스에서 사용할 수 있는 RESTful API와 함께 컨테이너 또는 데몬으로 배포됩니다. 서비스에 OPA의 결정이 필요한 경우 중앙 OPA RESTful API를 호출하여 이 결정을 내립니다. OPA는 한 번만 배포되므로 이 접근 방식은 배포 및 유지 관리가 간단합니다. 이 접근 방식의 단점은 테넌트 데이터 분리를 유지하기 위한 메커니즘을 제공하지 않는다는 것입니다. OPA는 한 번만 배포되므로 OPA에서 참조하는 외부 데이터를 포함하여 OPA 결정에 사용되는 모든 테넌트 데이터를 OPA에서 사용할 수 있어야 합니다. 이 접근 방식을 사용하여 테넌트 데이터 격리를 유지할 수 있지만 OPA의 정책 및 문서 구조 또는 외부 데이터에 대한 액세스를 통해 이를

적용해야 합니다. 또한 중앙 집중식 배포 패턴을 사용하려면 각 권한 부여 결정 시 다른 서비스에 대한 RESTful API 호출을 수행해야 하므로 지연 시간도 길어야 합니다.

분산 배포 패턴에서 OPA는 멀티테넌트 애플리케이션 서비스와 함께 컨테이너 또는 데몬으로 배포됩니다. 사이드카 컨테이너로 배포하거나 운영 체제에서 로컬로 실행되는 데몬으로 배포할 수 있습니다. OPA에서 결정을 가져오기 위해 서비스는 로컬 OPA 배포에 대해 RESTful API를 호출합니다. (OPA는 Go 패키지로 배포할 수 있으므로 RESTful API 호출을 사용하는 대신 Go를 기본적으로 사용하여 결정을 검색할 수 있습니다.) 중앙 집중식 배포 패턴과 달리 분산 패턴은 애플리케이션의 여러 영역에 걸쳐 존재하기 때문에 배포, 유지 관리 및 업데이트에 훨씬 더 강력한 노력이 필요합니다. 분산 배포 패턴의 이점은 특히 사일로화된 SaaS 모델을 사용하는 애플리케이션의 경우 테넌트 데이터의 격리를 유지할 수 있다는 것입니다. 분산 모델의 OPA는 테넌트와 함께 배포되므로 해당 테넌트별 OPA 배포에서 테넌트별 데이터를 분리할 수 있습니다. 또한 분산 배포 패턴은 각 권한 부여 결정을 로컬에서 내릴 수 있으므로 중앙 집중식 배포 패턴보다 지연 시간이 훨씬 짧습니다.

멀티테넌트 애플리케이션에서 OPA 배포 패턴을 선택할 때는 애플리케이션에 가장 중요한 기준을 평가해야 합니다. 멀티테넌트 애플리케이션이 지연 시간에 민감한 경우 분산 배포 패턴을 사용하면 배포 및 유지 관리가 더 복잡한 대신 더 나은 성능을 제공할 수 있습니다. 자동화를 통해 DevOps 이러한 복잡성 중 일부를 관리할 수 있지만 중앙 집중식 배포 패턴과 비교할 때 여전히 추가 노력이 필요합니다.

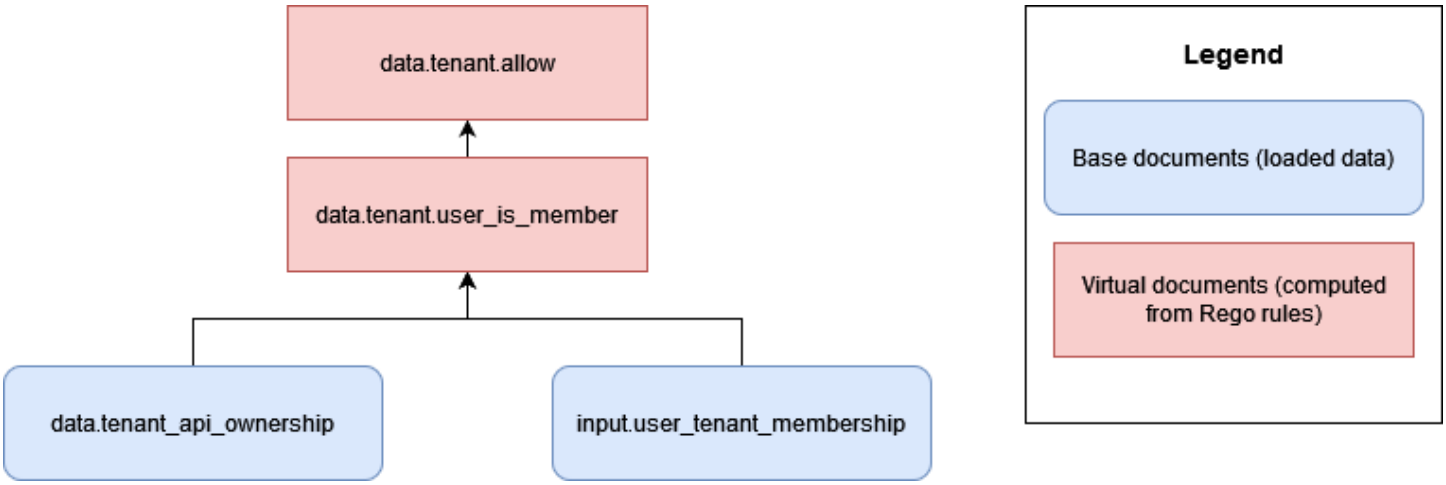
멀티테넌트 애플리케이션이 사일로화된 SaaS 모델을 사용하는 경우 분산 OPA 배포 패턴을 사용하여 테넌트 데이터 격리에 대한 사일로화된 접근 방식을 모방할 수 있습니다. 이는 OPA가 각 테넌트별 애플리케이션 서비스와 함께 실행되는 경우 해당 테넌트와 관련된 데이터만 포함하도록 각 OPA 배포를 사용자 지정할 수 있기 때문입니다. 중앙 집중식 OPA 배포 패턴에서 OPA 데이터를 사일로화하는 것은 불가능합니다. 중앙 집중식 배포 패턴 또는 분산 패턴을 풀링된 SaaS 모델과 함께 사용하는 경우 OPA 문서 모델에서 테넌트 데이터 격리를 유지해야 합니다.

OPA 문서 모델을 사용한 테넌트 격리

OPA는 문서를 사용하여 결정을 내립니다. 이러한 문서에는 테넌트별 데이터가 포함될 수 있으므로 테넌트 데이터 격리를 유지하는 방법을 고려해야 합니다. OPA 문서는 기본 문서와 가상 문서로 구성됩니다. 기본 문서에는 외부 세계의 데이터가 포함됩니다. 여기에는 OPA에 직접 제공된 데이터, OPA 요청에 대한 데이터, OPA에 입력으로 전달될 수 있는 데이터가 포함됩니다. 가상 문서는 정책에 따라 계산되며 OPA 정책 및 규칙을 포함합니다. 자세한 내용은 OPA 설명서를 참조하십시오.

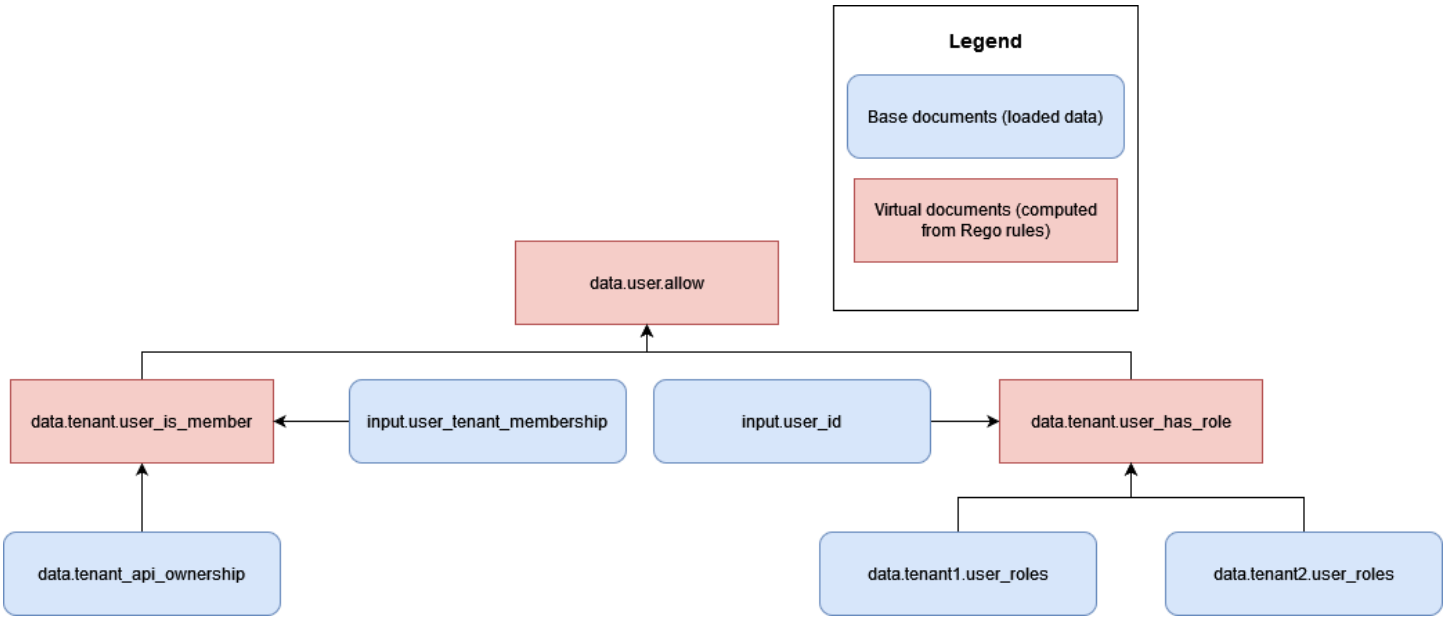
OPA에서 멀티테넌트 응용 프로그램을 위한 문서 모델을 설계하려면 먼저 OPA에서 결정을 내리는 데 필요한 기본 문서 유형을 고려해야 합니다. 이러한 기본 문서에 테넌트별 데이터가 포함된 경우 이 데이터가 실수로 테넌트 간 액세스에 노출되지 않도록 조치를 취해야 합니다. 다행히도 대부분의 경우 OPA에서 결정을 내리는 데 테넌트별 데이터가 필요하지 않습니다. 다음 예제는 입력 문서에 표시된 대

로 API를 소유한 테넌트와 사용자가 테넌트의 구성원인지 여부를 기반으로 API에 대한 액세스를 허용하는 가상의 OPA 문서 모델을 보여줍니다.



이 접근 방식에서 OPA는 API를 소유한 테넌트에 대한 정보를 제외하고는 테넌트별 데이터에 액세스할 수 없습니다. 이 경우 OPA가 액세스 결정을 내리는 데 사용하는 유일한 정보는 사용자와 테넌트의 연결 및 테넌트와 API의 연결뿐이므로 OPA가 테넌트 간 액세스를 용이하게 하는 것에 대해 걱정할 필요가 없습니다. 각 테넌트가 독립적인 리소스에 대한 소유권을 가지므로 이러한 유형의 OPA 문서 모델을 사일로화된 SaaS 모델에 적용할 수 있습니다.

그러나 많은 RBAC 권한 부여 접근 방식에서는 테넌트 간에 정보가 노출될 가능성이 있습니다. 다음 예제에서 가상 OPA 문서 모델은 사용자가 테넌트의 구성원인지 여부와 사용자에게 API에 액세스하는 데 필요한 올바른 역할이 있는지 여부에 따라 API에 대한 액세스를 허용합니다.



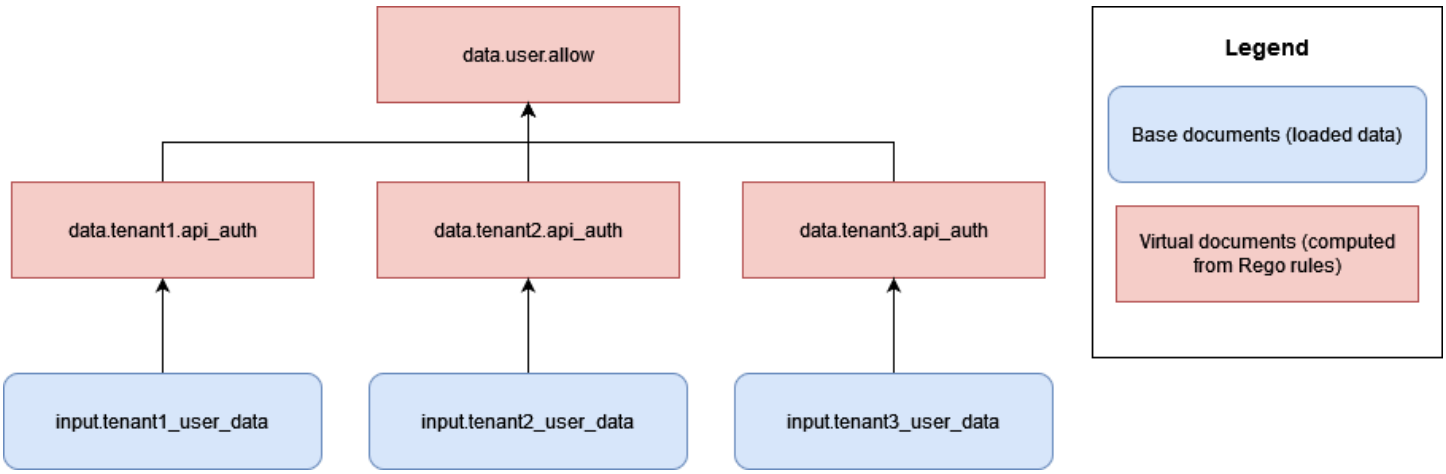
권한 부여 결정을 `data.tenant2.user_roles` 내리려면 이제 OPA에서 여러 테넌트의 역할과 권한에 `data.tenant1.user_roles` 액세스할 수 있어야 하기 때문에 이 모델은 테넌트 간 액세스의 위험을 초래합니다. 테넌트 격리를 유지하고 역할 매핑의 개인 정보를 보호하려면 이 데이터가 OPA 내에 있지 않아야 합니다. RBAC 데이터는 데이터베이스와 같은 외부 데이터 소스에 있어야 합니다. 또한 OPA는 사전 정의된 역할을 특정 권한에 매핑하는 데 사용해서는 안 됩니다. 이렇게 하면 테넌트가 자신의 역할과 권한을 정의하기가 어려워지기 때문입니다. 또한 권한 부여 로직이 엄격해지기 때문에 지속적인 업데이트가 필요합니다. RBAC 데이터를 OPA 의사 결정 프로세스에 안전하게 통합하는 방법에 대한 지침은 이 가이드 뒷부분의 테넌트 격리 및 데이터 개인 정보 보호를 위한 권장 사항 섹션을 참조하십시오.

테넌트별 데이터를 비동기 기본 문서로 저장하지 않으므로 OPA에서 테넌트 격리를 쉽게 유지할 수 있습니다. 비동기 기본 문서는 메모리에 저장되고 OPA에서 정기적으로 업데이트할 수 있는 데이터입니다. OPA 입력과 같은 기타 기본 문서는 동기적으로 전달되며 결정 시에만 사용할 수 있습니다. 예를 들어 쿼리에 대한 OPA 입력의 일부로 테넌트별 데이터를 제공해도 해당 데이터는 의사 결정 과정에서 동기식으로만 사용할 수 있으므로 테넌트 격리를 위반하는 것으로 간주되지 않습니다.

테넌트 온보딩

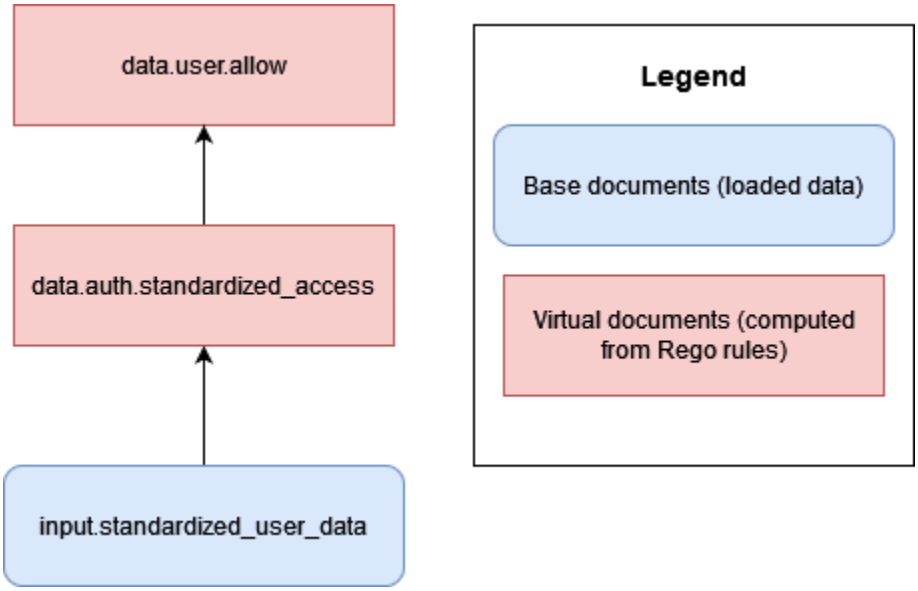
OPA 문서의 구조는 번거로운 요구 사항을 도입하지 않고도 간단한 테넌트 온보딩이 가능해야 합니다. OPA 문서 모델 계층 구조의 가상 문서를 패키지를 사용하여 구성할 수 있으며 이러한 패키지에는 많은 규칙이 포함될 수 있습니다. 멀티 테넌트 응용 프로그램을 위한 OPA 문서 모델을 계획할 때는 먼저 OPA가 결정을 내리는 데 필요한 데이터를 결정하십시오. 데이터를 입력으로 제공하거나, OPA에 미리 로드하거나, 의사 결정 시 또는 정기적으로 외부 데이터 소스에서 데이터를 제공할 수 있습니다. OPA에서 외부 데이터를 사용하는 방법에 대한 자세한 내용은 이 안내서의 뒷부분에 나오는 OPA에서 PDP에 대한 외부 데이터 검색 섹션을 참조하십시오.

OPA에서 결정을 내리는 데 필요한 데이터를 결정한 후에는 패키지로 구성된 OPA 규칙을 구현하여 해당 데이터로 결정을 내리는 방법을 고려해 보십시오. 예를 들어 각 테넌트에 권한 부여 결정 방식에 대한 고유한 요구 사항이 있을 수 있는 사일로화된 SaaS 모델에서는 테넌트별 OPA 규칙 패키지를 구현할 수 있습니다.



이 접근 방식의 단점은 SaaS 애플리케이션에 추가하는 각 테넌트에 대해 각 테넌트별로 고유한 새 OPA 규칙 세트를 추가해야 한다는 것입니다. 이는 번거롭고 확장하기 어렵지만 테넌트의 요구 사항에 따라 불가피한 경우도 있습니다.

또는 풀링된 SaaS 모델에서 모든 테넌트가 동일한 규칙을 기반으로 권한 부여 결정을 내리고 동일한 데이터 구조를 사용하는 경우 일반적으로 적용 가능한 규칙이 있는 표준 OPA 패키지를 사용하여 테넌트를 쉽게 등록하고 OPA 구현을 확장할 수 있습니다.



가능하면 일반화된 OPA 규칙 및 패키지 (또는 가상 문서) 를 사용하여 각 테넌트가 제공하는 표준화된 데이터를 기반으로 결정을 내리는 것이 좋습니다. 이 접근 방식을 사용하면 OPA가 규칙을 통해 결정을 내리는 방식이 아니라 각 테넌트에 대해 OPA에 제공된 데이터만 변경하므로 OPA를 쉽게 확장할 수 있습니다. 개별 테넌트가 고유한 결정을 요구하거나 OPA에 다른 테넌트와 다른 데이터를 제공해야 하는 경우에만 rules-per-tenant 모델을 도입하면 됩니다.

DevOps, PDP 데이터 모니터링, 로깅 및 검색

이 제안된 권한 부여 패러다임에서는 정책이 권한 부여 서비스에 중앙 집중화됩니다. 이 가이드에서 설명하는 설계 모델의 목표 중 하나는 정책 디커플링, 즉 응용 프로그램의 다른 구성 요소에서 권한 부여 논리를 제거하는 것이기 때문에 이러한 중앙 집중화는 의도적인 것입니다. Amazon 검증 권한과 공개 정책 에이전트 (OPA) 모두 권한 부여 로직을 변경해야 할 때 정책을 업데이트하기 위한 메커니즘을 제공합니다.

검증된 권한의 경우 AWS SDK에서 정책을 프로그래밍 방식으로 업데이트하는 메커니즘을 제공합니다 (Amazon 검증 권한 API 참조 가이드 참조). SDK를 사용하면 필요에 따라 새 정책을 푸시할 수 있습니다. 또한 Verified Permissions는 관리형 서비스이므로 업데이트를 수행하기 위해 컨트롤 플레인이나 에이전트를 관리, 구성 또는 유지 관리할 필요가 없습니다. 하지만 지속적 통합 및 지속적 배포 (CI/CD) 파이프라인을 사용하여 검증된 권한 정책 저장소 및 정책 업데이트의 배포를 SDK를 사용하여 관리하는 것이 좋습니다. AWS

검증된 권한을 통해 옵저버빌리티 기능에 쉽게 액세스할 수 있습니다. Amazon CloudWatch 로그 그룹 AWS CloudTrail, Amazon Simple Storage Service (Amazon S3) 버킷 또는 Amazon Data Firehose 전송 스트림에 대한 모든 액세스 시도를 기록하도록 구성하여 보안 사고 및 감사 요청에 신속하게 대응할 수 있습니다. 또한 를 통해 검증된 권한 서비스의 상태를 모니터링할 수 있습니다. AWS Health Dashboard검증된 권한은 관리형 서비스이므로 상태가 유지 관리되며 다른 AWS 관리형 서비스를 사용하여 관찰 기능을 구성할 수 있습니다. AWS

OPA의 경우 REST API는 정책을 프로그래밍 방식으로 업데이트할 수 있는 방법을 제공합니다. 설정된 위치에서 새 버전의 정책 번들을 가져오거나 요청 시 정책을 푸시하도록 API를 구성할 수 있습니다. 또한 OPA는 검색 번들을 배포하는 컨트롤 플레인을 통해 새 에이전트를 동적으로 구성하고 중앙에서 관리할 수 있는 기본 검색 서비스를 제공합니다. (OPA의 컨트롤 플레인은 OPA 운영자가 설정하고 구성해야 합니다.) 정책 엔진이 검증된 권한이든 OPA이든 다른 솔루션이든 상관없이 정책의 버전 관리, 확인 및 업데이트를 위한 강력한 CI/CD 파이프라인을 생성하는 것이 좋습니다.

OPA의 경우 컨트롤 플레인은 모니터링 및 감사 옵션도 제공합니다. OPA의 권한 부여 결정이 포함된 로그를 원격 HTTP 서버로 내보내 로그 집계를 수행할 수 있습니다. 이러한 의사 결정 로그는 감사 목적에 매우 유용합니다.

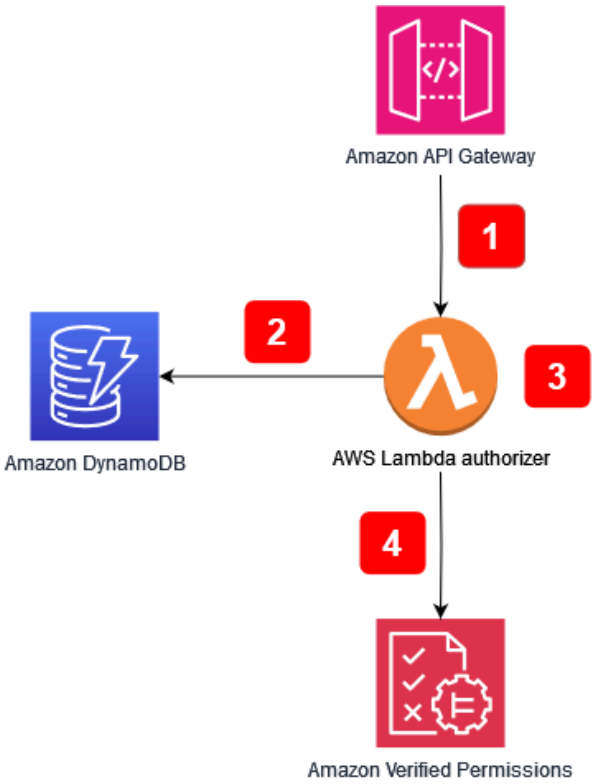
액세스 제어 결정을 애플리케이션과 분리하는 권한 부여 모델을 채택하려는 경우 권한 부여 서비스가 새 PDP를 온보딩하거나 정책을 업데이트하기 위한 효과적인 모니터링, 로깅 및 CI/CD 관리 기능을 갖추고 있는지 확인하십시오.

주제

Amazon 검증 권한에서 PDP에 대한 외부 데이터 검색

Amazon Verified Permissions는 PDP의 외부 데이터 검색을 지원하지 않지만 사용자가 제공한 데이터를 스키마의 일부로 저장할 수 있습니다. OPA에서처럼 승인 결정을 위한 모든 데이터를 승인 요청의 일부로 제공하거나 요청의 일부로 전달되는 JSON 웹 토큰 (JWT) 의 일부로 제공할 수 있는 경우 추가 구성이 필요하지 않습니다. 하지만 검증된 권한을 호출하는 애플리케이션의 권한 부여 서비스의 일부로 권한 부여 요청을 통해 외부 소스의 추가 데이터를 검증된 권한에 제공할 수 있습니다. 예를 들어 애플리케이션의 권한 부여 서비스는 DynamoDB 또는 Amazon RDS와 같은 외부 소스에 데이터를 쿼리할 수 있으며, 그러면 이러한 서비스가 외부에서 제공한 데이터를 권한 부여 요청의 일부로 포함할 수 있습니다.

다음 다이어그램은 추가 데이터를 검색하여 Verified Permissions 승인 요청에 통합하는 방법의 예를 보여줍니다. RBAC 역할 매핑과 같은 데이터를 검색하고, 리소스 또는 보안 주체와 관련된 추가 속성을 검색하거나, 데이터가 응용 프로그램의 다른 부분에 있고 ID 공급자 (IdP) 토큰을 통해 제공할 수 없는 경우 이 방법을 사용해야 할 수 있습니다.



신청 흐름:

1. 애플리케이션은 Amazon API Gateway에 대한 API 호출을 수신하고 AWS Lambda 권한을 부여자에게 전달합니다.
2. Lambda 권한 부여자는 Amazon DynamoDB를 호출하여 요청을 한 보안 주체에 대한 추가 데이터를 검색합니다.
3. Lambda 권한 부여자는 검증된 권한에 대한 권한 부여 요청에 추가 데이터를 통합합니다.
4. Lambda 권한 부여자는 검증된 권한에 권한 부여를 요청하고 승인 결정을 받습니다.

다이어그램에는 Lambda 권한 부여자라는 Amazon API Gateway의 기능이 포함되어 있습니다. 다른 서비스나 애플리케이션에서 제공하는 API에는 이 기능을 사용하지 못할 수도 있지만, 권한 부여자를 사용하여 추가 데이터를 가져와서 다양한 사용 사례에서 Verified Permissions 승인 요청에 통합하는 일반적인 모델을 복제할 수 있습니다.

OPA에서 PDP에 대한 외부 데이터 검색

OPA의 경우 권한 부여 결정에 필요한 모든 데이터를 입력으로 제공하거나 쿼리의 구성 요소로 전달되는 JWT (JSON 웹 토큰) 의 일부로 제공할 수 있는 경우 추가 구성이 필요하지 않습니다. (쿼리 입력의 일부로 JWT 및 SaaS 컨텍스트 데이터를 OPA에 전달하는 것은 비교적 간단합니다.) OPA는 오버로드 입력 방식이라고 하는 임의의 JSON 입력을 받아들일 수 있습니다. PDP에 입력 또는 JWT로 포함할 수 있는 것 이상의 데이터가 필요한 경우 OPA는 이 데이터를 검색할 수 있는 몇 가지 옵션을 제공합니다. 여기에는 번들링, 데이터 푸시 (복제) 및 동적 데이터 검색이 포함됩니다.

OPA 번들링

OPA 번들링 기능은 외부 데이터 검색을 위한 다음 프로세스를 지원합니다.

1. 정책 적용 지점 (PEP) 은 권한 부여 결정을 요청합니다.
2. OPA는 외부 데이터를 포함한 새 정책 번들을 다운로드합니다.
3. 번들링 서비스는 데이터 소스의 데이터를 복제합니다.

번들링 기능을 사용하면 OPA는 중앙 집중식 번들 서비스에서 정책 및 데이터 번들을 정기적으로 다운로드합니다. (OPA는 번들 서비스의 구현 및 설정을 제공하지 않습니다.) 번들 서비스에서 가져온 모든 정책과 외부 데이터는 메모리에 저장됩니다. 외부 데이터 크기가 너무 커서 메모리에 저장할 수 없거나 데이터가 너무 자주 변경되는 경우에는 이 옵션이 작동하지 않습니다.

번들링 기능에 대한 자세한 내용은 OPA 설명서를 참조하십시오.

OPA 복제 (데이터 푸시)

OPA 복제 접근 방식은 외부 데이터 검색을 위한 다음 프로세스를 지원합니다.

1. PEP는 권한 부여 결정을 요청합니다.
2. 데이터 복제기는 데이터를 OPA로 푸시합니다.
3. 데이터 복제기는 데이터 원본의 데이터를 복제합니다.

번들링 방식을 대체하는 이 방식에서는 데이터를 주기적으로 가져오는 대신 OPA로 푸시합니다. (OPA는 복제기의 구현 및 설정을 제공하지 않습니다.) OPA는 모든 데이터를 메모리에 저장하기 때문에 푸시 접근 방식에는 번들링 접근 방식과 동일한 데이터 크기 제한이 있습니다. 푸시 옵션의 주요 장점은 매번 외부 데이터를 모두 바꾸는 대신 델타를 사용하여 OPA의 데이터를 업데이트할 수 있다는 것입니다. 따라서 푸시 옵션은 자주 변경되는 데이터세트에 더 적합합니다.

복제 옵션에 대한 자세한 내용은 OPA 설명서를 참조하십시오.

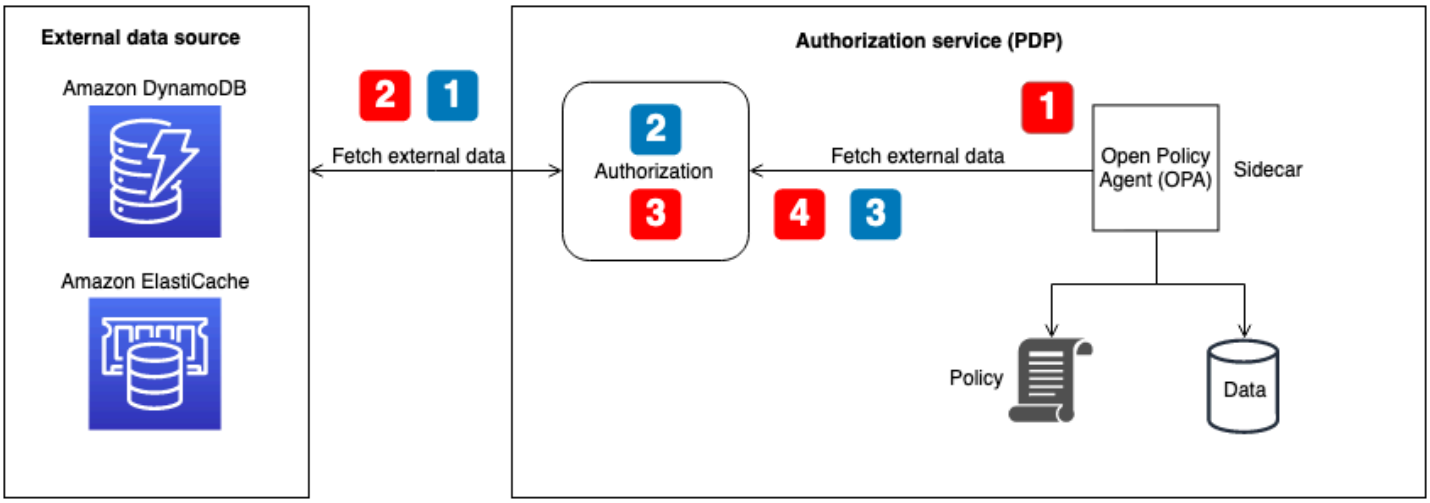
OPA 동적 데이터 검색

검색할 외부 데이터가 너무 커서 OPA의 메모리에 캐시할 수 없는 경우 권한 부여 결정을 평가하는 동안 외부 소스에서 데이터를 동적으로 가져올 수 있습니다. 이 접근 방식을 사용하면 데이터가 항상 최신 상태로 유지됩니다. 이 접근 방식에는 네트워크 지연 시간과 접근성이라는 두 가지 단점이 있습니다. 현재 OPA는 HTTP 요청을 통해서만 런타임에 데이터를 검색할 수 있습니다. 외부 데이터 소스로 이동하는 호출에서 HTTP 응답으로 데이터를 반환할 수 없는 경우 이 데이터를 OPA에 제공하기 위한 사용자 지정 API 또는 기타 메커니즘이 필요합니다. OPA는 HTTP 요청을 통해서만 데이터를 검색할 수 있고 데이터 검색 속도가 매우 중요하므로 가능하면 AWS 서비스 Amazon DynamoDB와 같은 외부 데이터를 보관하는 것이 좋습니다.

풀 접근 방식에 대한 자세한 내용은 OPA 설명서를 참조하십시오.

OPA를 통한 구현을 위한 권한 부여 서비스 사용

번들링, 복제 또는 동적 풀 접근 방식을 사용하여 외부 데이터를 가져올 때는 권한 부여 서비스가 이러한 상호 작용을 용이하게 하는 것이 좋습니다. 이는 권한 부여 서비스가 외부 데이터를 검색하고 이를 JSON으로 변환하여 OPA가 권한 부여 결정을 내릴 수 있기 때문입니다. 다음 다이어그램은 이러한 세 가지 외부 데이터 검색 접근 방식을 통해 권한 부여 서비스가 작동하는 방식을 보여줍니다.



OPA 플로우를 위한 외부 데이터 검색 — 의사 결정 시 번들 또는 동적 데이터 검색 (다이어그램에서 빨간색 번호가 매겨진 콜아웃으로 표시됨)

1. OPA는 권한 부여 서비스의 로컬 API 엔드포인트를 호출하며, 이 엔드포인트는 권한 부여 결정 시 동적 데이터 검색을 위한 번들 엔드포인트 또는 엔드포인트로 구성됩니다.
2. 권한 부여 서비스는 외부 데이터 원본을 쿼리하거나 호출하여 외부 데이터를 검색합니다. (번들 엔드포인트의 경우 이 데이터에는 OPA 정책 및 규칙도 포함되어야 합니다. 번들 업데이트는 OPA 캐시에 있는 모든 데이터 및 정책을 대체합니다.
3. 권한 부여 서비스는 반환된 데이터를 예상 JSON 입력으로 변환하기 위해 필요한 모든 변환을 수행합니다.
4. 데이터가 OPA로 반환됩니다. 번들 구성을 위해 메모리에 캐시되며 동적 권한 부여 결정에 즉시 사용됩니다.

OPA 플로우에 대한 외부 데이터 검색 — Replicator (다이어그램에서 파란색 번호 콜아웃으로 표시됨)

1. 복제기 (권한 부여 서비스의 일부) 는 외부 데이터 원본을 호출하고 OPA에서 업데이트할 모든 데이터를 검색합니다. 여기에는 정책, 규칙 및 외부 데이터가 포함될 수 있습니다. 이 호출은 정해진 빈도로 이루어질 수도 있고 외부 소스의 데이터 업데이트에 대한 응답으로 발생할 수도 있습니다.
2. 권한 부여 서비스는 반환된 데이터를 예상 JSON 입력으로 변환하기 위해 필요한 모든 변환을 수행합니다.
3. 권한 부여 서비스는 OPA를 호출하고 데이터를 메모리에 캐시합니다. 권한 부여 서비스는 데이터, 정책 및 규칙을 선택적으로 업데이트할 수 있습니다.

테넌트 격리 및 데이터 개인 정보 보호를 위한 권장 사항

이전 섹션에서는 권한 부여 결정을 지원하기 위해 OPA 및 Amazon 검증 권한과 함께 외부 데이터를 사용하는 몇 가지 접근 방식을 제공했습니다. 가능하면 OPA의 메모리에 데이터를 저장하는 대신 SaaS 컨텍스트 데이터를 OPA에 전달하여 권한 부여 결정을 내리는 데 오버로드 입력 접근 방식을 사용하는 것이 좋습니다. 서비스에 외부 데이터를 저장하는 것을 AWS Cloud Map지원하지 않기 때문에 이 사용 사례는 적용되지 않습니다.

역할 기반 액세스 제어 (RBAC) 또는 RBAC 및 속성 기반 액세스 제어 (ABAC) 하이브리드 모델에서는 권한 부여 결정을 내리려면 역할과 권한을 참조해야 하므로 권한 부여 요청 또는 쿼리로만 제공되는 데이터로는 충분하지 않을 수 있습니다. 테넌트 격리를 유지하고 역할 매핑의 개인 정보를 보호하려면 이 데이터가 OPA에 있지 않아야 합니다. RBAC 데이터는 데이터베이스와 같은 외부 데이터 소스에 있거나 IdP에서 JWT 클레임의 일부로 전달되어야 합니다. Verified Permissions에서는 각 테넌트에 논리적으로 분리된 자체 정책 저장소가 있기 때문에 RBAC 데이터를 테넌트별 정책 저장소 모델의 정책 및 스키마의 일부로 유지 관리할 수 있습니다. 그러나 단일 공유 다중 테넌트 정책 저장소 모델에서는 테넌트 격리를 유지하기 위해 역할 매핑 데이터가 Verified Permissions 내에 있어서는 안 됩니다.

또한 OPA와 검증된 권한을 사용하여 사전 정의된 역할을 특정 권한에 매핑하면 안 됩니다. 이렇게 하면 테넌트가 자신의 역할과 권한을 정의하기 어렵기 때문입니다. 또한 권한 부여 로직이 까다로워지고 지속적인 업데이트가 필요합니다. Verified Permissions의 테넌트별 정책 저장소 모델은 예외입니다. 이 모델을 사용하면 각 테넌트가 테넌트별로 독립적으로 평가할 수 있는 자체 정책을 가질 수 있기 때문입니다.

Amazon Verified Permissions

검증된 권한이 잠재적으로 비공개인 RBAC 데이터를 저장할 수 있는 유일한 장소는 스키마입니다. 각 테넌트에는 논리적으로 분리된 자체 정책 저장소가 있기 때문에 테넌트별 정책 저장소 모델에서는 이 방법을 사용할 수 있습니다. 그러나 단일 공유 멀티테넌트 정책 저장소 모델에서는 테넌트 격리가 손상될 수 있습니다. 권한 부여 결정을 내리는 데 이 데이터가 필요한 경우 DynamoDB 또는 Amazon RDS와 같은 외부 소스에서 데이터를 검색하여 검증된 권한 승인 요청에 통합해야 합니다.

OPA

RBAC 데이터의 개인 정보 보호 및 테넌트 격리를 유지하기 위한 OPA의 안전한 접근 방식에는 동적 데이터 검색 또는 복제를 사용하여 외부 데이터를 가져오는 것이 포함됩니다. 이는 이전 다이어그램에 나와 있는 권한 부여 서비스를 사용하여 권한 부여 결정을 내리는 데 필요한 테넌트별 또는 사용자별 외부 데이터만 제공할 수 있기 때문입니다. 예를 들어, 사용자가 로그인할 때 복제기를 사용하여 OPA 캐시에 RBAC 데이터 또는 권한 매트릭스를 제공하고 입력 데이터에 제공된 사용자를 기반으로 데이

터를 참조하도록 할 수 있습니다. 동적으로 가져온 데이터에도 유사한 접근 방식을 사용하여 권한 부여 결정을 내리는 데 필요한 관련 데이터만 검색할 수 있습니다. 또한 동적 데이터 검색 접근 방식에서는 이 데이터를 OPA에 캐시할 필요가 없습니다. 번들링 접근 방식은 OPA 캐시의 모든 내용을 업데이트하고 정확한 업데이트를 처리할 수 없기 때문에 테넌트 격리를 유지하는 데 동적 검색 접근 방식만큼 효과적이지 않습니다. 번들링 모델은 여전히 OPA 정책 및 비 RBAC 데이터를 업데이트하는 데 적합합니다.

모범 사례

이 섹션에는 이 가이드의 몇 가지 개괄적인 내용이 나열되어 있습니다. 각 요점에 대한 자세한 설명을 보려면 해당 섹션으로 연결되는 링크를 클릭하십시오.

애플리케이션에 적합한 액세스 제어 모델을 선택하십시오.

이 가이드에서는 여러 액세스 제어 모델에 대해 설명합니다. 애플리케이션 및 비즈니스 요구 사항에 따라 적합한 모델을 선택해야 합니다. 이러한 모델을 사용하여 액세스 제어 요구 사항을 충족하는 방법과 선택한 접근 방식을 변경해야 하는 액세스 제어 요구 사항이 어떻게 발전할지 생각해 보십시오.

PDP 구현

정책 결정 지점 (PDP) 은 정책 또는 규칙 엔진으로 특징지을 수 있습니다. 이 구성 요소는 정책 또는 규칙을 적용하고 특정 액세스의 허용 여부에 대한 결정을 반환하는 역할을 합니다. PDP를 사용하면 애플리케이션 코드의 권한 부여 로직을 별도의 시스템으로 오프로드할 수 있습니다. 이렇게 하면 애플리케이션 코드를 단순화할 수 있습니다. 또한 API, 마이크로서비스, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내릴 수 있는 동등한 인터페이스를 제공합니다. easy-to-use PDP를 사용하여 애플리케이션 전체에 테넌시 요구 사항을 일관되게 적용할 수 있습니다.

애플리케이션의 모든 API에 PEP를 구현하십시오.

정책 적용 지점 (PEP) 을 구현하려면 애플리케이션에서 액세스 제어를 적용해야 하는 위치를 결정해야 합니다. 첫 번째 단계로, 애플리케이션에서 PEP를 통합할 수 있는 지점을 찾으십시오. PEP 추가 위치를 결정할 때는 다음 원칙을 고려하십시오.

애플리케이션이 API를 노출하는 경우 해당 API에 대한 권한 부여 및 액세스 제어가 있어야 합니다.

Amazon 검증 권한 또는 OPA를 PDP의 정책 엔진으로 사용하는 것을 고려해 보십시오.

Amazon 검증 권한은 사용자 지정 정책 엔진에 비해 장점이 있습니다. 구축하는 애플리케이션을 위한 확장 가능하고 세분화된 권한 관리 및 권한 부여 서비스입니다. 고급 선언적 오픈 소스 언어인 Cedar로 정책을 작성할 수 있도록 지원합니다. 따라서 Verified Permissions를 사용하여 정책 엔진을 구현하는

경우 자체 솔루션을 구현하는 것보다 개발 작업이 덜 필요합니다. 또한 검증된 권한은 완전히 관리되므로 기본 인프라를 관리할 필요가 없습니다.

개방형 정책 에이전트 (OPA) 는 사용자 지정 정책 엔진에 비해 장점이 있습니다. OPA와 Rego를 통한 정책 평가는 고급 선언적 언어로 정책을 작성할 수 있도록 지원하는 유연한 사전 구축된 정책 엔진을 제공합니다. 따라서 자체 솔루션을 구축하는 것보다 정책 엔진을 구현하는 데 필요한 노력이 훨씬 적습니다. 또한 OPA는 빠르게 잘 지원되는 권한 부여 표준으로 자리잡고 있습니다.

모니터링 DevOps, 로깅을 위한 OPA용 컨트롤 플레인을 구현하십시오.

OPA는 소스 제어를 통해 권한 부여 로직의 변경 내용을 업데이트하고 추적하는 방법을 제공하지 않으므로 이러한 기능을 수행하도록 컨트롤 플레인을 구현하는 것이 좋습니다. 이렇게 하면 특히 OPA가 분산 시스템에서 운영되는 경우 업데이트를 OPA 에이전트에 더 쉽게 배포할 수 있으므로 OPA 사용에 따른 관리 부담이 줄어듭니다. 또한 컨트롤 플레인을 사용하여 집계를 위한 로그를 수집하고 OPA 에이전트의 상태를 모니터링할 수 있습니다.

검증된 권한에서 로깅 및 관찰 기능을 구성하십시오.

검증된 권한을 통해 옵저버빌리티 기능에 쉽게 액세스할 수 있습니다. Amazon CloudWatch 로그 그룹 AWS CloudTrail, S3 버킷 또는 Amazon Data Firehose 전송 스트림에 대한 모든 액세스 시도를 기록하도록 서비스를 구성하여 보안 사고 및 감사 요청에 신속하게 대응할 수 있습니다. 또한 를 통해 서비스 상태를 모니터링할 수 있습니다. AWS Health Dashboard Verified Permissions는 관리형 서비스이므로 상태가 유지 관리되며 다른 AWS 관리형 서비스를 사용하여 옵저버빌리티 기능을 구성할 수 있습니다. AWS

CI/CD 파이프라인을 사용하여 검증된 권한의 정책 저장소와 정책을 프로비저닝하고 업데이트하십시오.

검증된 권한은 관리형 서비스이므로 업데이트를 수행하기 위해 컨트롤 플레인이나 에이전트를 관리, 구성 또는 유지 관리할 필요가 없습니다. 하지만 SDK를 사용하여 검증된 권한 정책 저장소 및 정책 업데이트의 배포를 관리하려면 CI/CD (지속적 통합 및 지속적 배포) 파이프라인을 사용하는 것이 좋습니다. AWS이렇게 하면 수동 작업을 없애고 Verified Permissions 리소스를 변경할 때 운영자 오류가 발생할 가능성을 줄일 수 있습니다.

권한 부여 결정에 외부 데이터가 필요한지 여부를 결정하고 이를 수용할 모델을 선택하십시오.

PDP가 JSON 웹 토큰 (JWT) 에 포함된 데이터만을 기반으로 권한 부여 결정을 내릴 수 있는 경우 일반적으로 이러한 결정을 내리는 데 도움이 되는 외부 데이터를 가져올 필요가 없습니다. 검증된 권한 또는 OPA를 PDP로 사용하는 경우 해당 데이터가 JWT에 포함되어 있지 않더라도 요청의 일부로 전달되는 추가 입력도 수락할 수 있습니다. 검증된 권한의 경우 추가 데이터에 컨텍스트 매개변수를 사용할 수 있습니다. OPA의 경우 JSON 데이터를 오버로드 입력으로 사용할 수 있습니다. JWT를 사용하는 경우 일반적으로 컨텍스트 또는 오버로드 입력 방법이 외부 데이터를 다른 소스에 유지하는 것보다 훨씬 쉽습니다. 권한 부여 결정을 내리는 데 더 복잡한 외부 데이터가 필요한 경우 OPA는 외부 데이터를 검색하기 위한 여러 모델을 제공하며 Verified Permissions는 권한 부여 서비스를 통해 외부 소스를 참조하여 권한 부여 요청의 데이터를 보완할 수 있습니다.

FAQ

이 섹션에서는 멀티테넌트 SaaS 애플리케이션에서 API 액세스 제어 및 권한 부여를 구현하는 것과 관련하여 자주 제기되는 질문에 대한 답변을 제공합니다.

Q: 권한 부여와 인증의 차이는 무엇입니까?

A. 인증은 사용자가 누구인지 확인하는 프로세스입니다. 권한 부여는 사용자에게 특정 리소스에 액세스할 수 있는 권한을 부여합니다.

Q: SaaS 애플리케이션에서 권한 부여와 테넌트 격리의 차이는 무엇입니까?

A. 테넌트 격리란 공유 인프라에서 운영되는 경우에도 각 테넌트의 리소스가 격리되도록 SaaS 시스템에서 사용되는 명시적 메커니즘을 말합니다. 멀티 테넌트 인증이란 인바운드 작업을 승인하여 잘못된 테넌트에서 구현되지 않도록 하는 것을 말합니다. 가상의 사용자는 인증 및 권한 부여를 받을 수 있지만 다른 테넌트의 리소스에는 여전히 액세스할 수 있습니다. 인증 및 권한 부여에 관한 어떤 것도 이 액세스를 반드시 차단하지는 않지만, 이 목표를 달성하려면 테넌트 격리가 필요합니다. 이 두 개념에 대한 자세한 내용은 AWS SaaS 아키텍처 기초 백서의 테넌트 격리 설명을 참조하십시오.

Q: SaaS 애플리케이션에 대해 테넌트 격리를 고려해야 하는 이유는 무엇입니까?

A. SaaS 애플리케이션에는 여러 테넌트가 있습니다. 테넌트는 고객 조직 또는 해당 SaaS 애플리케이션을 사용하는 외부 엔티티일 수 있습니다. 이는 애플리케이션 설계 방식에 따라 테넌트가 공유 API, 데이터베이스 또는 기타 리소스에 액세스할 수 있음을 의미합니다. 한 테넌트의 개인 정보에 사용자가 액세스하지 못하도록 테넌트 격리, 즉 리소스에 대한 액세스를 엄격하게 제어하고 다른 테넌트의 리소스에 액세스하려는 시도를 차단하는 구조를 유지하는 것이 중요합니다. SaaS 애플리케이션은 애플리케이션 전체에서 테넌트 격리가 유지되고 테넌트가 자체 리소스에만 액세스할 수 있도록 설계되는 경우가 많습니다.

Q: 액세스 제어 모델이 필요한 이유는 무엇입니까?

A. 액세스 제어 모델은 애플리케이션의 리소스에 대한 액세스 권한을 부여하는 방법을 결정하는 일관된 방법을 만드는 데 사용됩니다. 이는 사용자에게 비즈니스 로직과 밀접하게 연계된 역할을 할당하여 수행할 수도 있고, 시간대나 사용자가 사전 정의된 조건을 충족하는지 여부와 같은 다른 상황별 속성을 기반으로 할 수도 있습니다. 액세스 제어 모델은 애플리케이션에서 사용자 권한을 결정하기 위한 권한 부여 결정을 내릴 때 사용하는 기본 로직을 구성합니다.

Q: 내 애플리케이션에 API 액세스 제어가 필요한가요?

A. 네. API는 항상 호출자에게 적절한 액세스 권한이 있는지 확인해야 합니다. 또한 퍼베이시브 API 액세스 제어를 통해 테넌트를 기반으로 액세스 권한을 부여하여 적절한 격리를 유지할 수 있습니다.

Q: 권한 부여에 정책 엔진 또는 PDP를 권장하는 이유는 무엇입니까?

A. PDP (정책 결정 지점) 를 사용하면 애플리케이션 코드의 권한 부여 로직을 별도의 시스템으로 오프로드할 수 있습니다. 이를 통해 애플리케이션 코드를 단순화할 수 있습니다. 또한 API, 마이크로서비스, BFF (Backend for Fronend) 계층 또는 기타 애플리케이션 구성 요소에 대한 권한 부여 결정을 내릴 수 있는 동등한 인터페이스를 제공합니다. easy-to-use

Q. PEP란 무엇입니까?

A. 정책 적용 지점 (PEP) 은 평가를 위해 PDP에 보내는 권한 부여 요청을 받는 역할을 합니다. PEP는 데이터와 리소스를 보호해야 하거나 권한 부여 논리가 적용되는 응용 프로그램 어디에나 있을 수 있습니다. PEP는 PDP에 비해 비교적 간단합니다. PEP는 권한 부여 결정을 요청하고 평가하는 역할만 담당하며 권한 부여 논리를 PEP에 통합할 필요는 없습니다.

Q. Amazon 검증 권한과 OPA 중에서 어떻게 선택해야 합니까?

A. 검증된 권한과 공개 정책 에이전트 (OPA) 중에서 선택하려면 사용 사례와 고유한 요구 사항을 항상 염두에 두십시오. Verified Permissions는 밀리초 단위 처리로 애플리케이션 지연 요구 사항을 충족하면서 세분화된 권한을 정의하고, 애플리케이션 전반의 권한을 감사하고, 애플리케이션의 정책 관리 시스템을 중앙 집중화하는 완전 관리형 방법을 제공합니다. OPA는 애플리케이션 스택 전반에서 정책을 통합하는 데도 도움이 되는 오픈 소스 범용 정책 엔진입니다. OPA를 실행하려면 일반적으로 컨테이너 또는 함수를 사용하여 사용자 AWS 환경에서 OPA를 호스팅해야 합니다. AWS Lambda

검증된 권한은 오픈 소스 Cedar 정책 언어를 사용하는 반면 OPA는 Rego를 사용합니다. 따라서 이러한 언어 중 하나를 잘 알고 있으면 해당 솔루션을 선택해야 할 수도 있습니다. 하지만 두 언어에 대해 모두 읽은 다음 해결하려는 문제를 되돌아보고 사용 사례에 가장 적합한 솔루션을 찾는 것이 좋습니다.

Q: 검증된 권한과 OPA에 대한 오픈 소스 대안이 있습니까?

A. 검증된 권한 및 공개 정책 에이전트 (OPA) 와 유사한 몇 가지 오픈 소스 시스템 (예: 공통 표현 언어 (CEL) 이 있습니다. 이 가이드에서는 확장 가능한 권한 관리 및 세분화된 권한 부여 서비스인 Verified Permissions와 널리 채택되고 문서화되어 있으며 다양한 유형의 애플리케이션 및 인증 요구 사항에 맞게 조정할 수 있는 OPA에 중점을 둡니다.

Q: OPA를 사용하려면 권한 부여 서비스를 작성해야 합니까, 아니면 OPA와 직접 상호 작용할 수 있습니까?

A. OPA와 직접 상호 작용할 수 있습니다. 이 지침의 맥락에서 권한 부여 서비스는 권한 부여 결정 요청을 OPA 쿼리로 또는 그 반대로 변환하는 서비스를 말합니다. 애플리케이션이 OPA 응답을 직접 쿼리하고 수락할 수 있는 경우 이러한 복잡성을 가중시킬 필요가 없습니다.

Q: 가동 시간 및 감사 목적으로 OPA 에이전트를 모니터링하려면 어떻게 해야 합니까?

A. OPA는 로깅 및 기본 가동 시간 모니터링을 제공하지만 기본 구성으로는 엔터프라이즈 배포에 충분하지 않을 수 있습니다. 자세한 내용은 이 가이드 앞부분의 DevOps, 모니터링 및 로깅 섹션을 참조하십시오.

Q: 업타임 및 감사 목적으로 검증된 권한을 모니터링하려면 어떻게 해야 합니까?

A. 검증된 권한은 AWS 관리형 서비스이며 를 통해 가용성을 모니터링할 수 있습니다. AWS Health Dashboard또한 검증된 권한은 Amazon CloudWatch Logs AWS CloudTrail, Amazon S3 및 Amazon Data Firehose에 로그인할 수 있습니다.

Q: OPA를 실행하는 데 사용할 수 있는 운영 체제와 AWS 서비스는 무엇입니까?

A. macOS, 윈도우, 리눅스에서 OPA를 실행할 수 있습니다. OPA 에이전트는 아마존 Elastic Compute Cloud (Amazon EC2) 에이전트뿐만 아니라 아마존 엘라스틱 컨테이너 서비스 (Amazon ECS) 및 아마존 Elastic Kubernetes 서비스 (Amazon EKS) 와 같은 컨테이너화 서비스에서도 구성할 수 있습니다.

Q: 검증된 권한을 실행하는 데 사용할 수 있는 운영 체제와 서비스는 무엇입니까? AWS

A. 검증된 권한은 AWS 관리형 서비스이며 에서 운영됩니다 AWS. 서비스에 권한 부여 요청을 할 수 있는 기능 외에는 검증된 권한을 사용하기 위해 추가 구성, 설치 또는 호스팅이 필요하지 않습니다.

Q: 에서 OPA를 실행할 수 있습니까? AWS Lambda

A. Lambda에서 OPA를 Go 라이브러리로 실행할 수 있습니다. API Gateway Lambda 권한 부여자에 대해 이 작업을 수행하는 방법에 대한 자세한 내용은 블로그 게시물 Open Policy 에이전트를 사용하여 사용자 지정 Lambda 권한 부여자 만들기를 참조하십시오 AWS .

Q: 분산 PDP와 중앙 집중식 PDP 접근 방식 중 하나를 어떻게 결정해야 합니까?

A. 이는 애플리케이션에 따라 다릅니다. 분산 PDP 모델과 중앙 집중식 PDP 모델 간의 지연 시간 차이를 기반으로 결정될 가능성이 큽니다. 개념 증명을 작성하고 애플리케이션 성능을 테스트하여 솔루션을 검증하는 것이 좋습니다.

Q: API 이외의 사용 사례에 OPA를 사용할 수 있습니까?

A. 네. OPA 설명서에는 쿠버네티스, 엔보이, 도커, 카프카, SSH 및 sudo, 테라폼에 대한 예제가 나와 있습니다. 또한 OPA는 Rego 부분 규칙을 사용하여 쿼리에 임의의 JSON 응답을 반환할 수 있습니다. 쿼리에 따라 OPA를 사용하여 JSON 응답으로 많은 질문에 답할 수 있습니다.

Q: API 이외의 사용 사례에 검증된 권한을 사용할 수 있습니까?

A. 네. 검증된 권한은 수신한 모든 승인 요청에 `DENY` 대해 `ALLOW` 또는 응답을 제공할 수 있습니다. 검증된 권한은 권한 부여 결정이 필요한 모든 애플리케이션 또는 서비스에 대한 권한 부여 응답을 제공할 수 있습니다.

Q: IAM 정책 언어를 사용하여 검증된 권한에서 정책을 생성할 수 있습니까?

A. 아니요. 정책을 작성하려면 Cedar 정책 언어를 사용해야 합니다. Cedar는 고객 애플리케이션 리소스에 대한 권한 관리를 지원하도록 설계된 반면, AWS Identity and Access Management (IAM) 정책 언어는 리소스에 대한 액세스 제어를 지원하도록 발전했습니다. AWS

다음 단계

권한 부여를 결정하는 데 언어에 구애받지 않는 표준화된 접근 방식을 채택하여 다중 테넌트 SaaS 애플리케이션에 대한 권한 부여 및 API 액세스 제어의 복잡성을 극복할 수 있습니다. 이러한 접근 방식에는 유연하고 광범위한 방식으로 액세스를 강화하는 정책 결정 지점 (PDP) 과 정책 집행 지점 (PDP) 이 포함됩니다. RBAC(역할 기반 액세스 제어), ABAC(속성 기반 액세스 제어) 또는 이 둘의 조합과 같은 다양한 액세스 제어 접근 방식을 일관적인 액세스 제어 전략에 통합할 수 있습니다. 애플리케이션에서 권한 부여 논리를 제거하면 액세스 제어 문제를 해결하기 위해 애플리케이션 코드에 임시 솔루션을 포함하는 데 따른 부담이 사라집니다. 이 설명서에서 설명하는 구현 및 모범 사례는 다중 테넌트 SaaS 애플리케이션에서 권한 부여 및 API 액세스 제어를 구현하는 방법에 대한 정보를 제공하고 표준화하는 것을 목적으로 합니다. 이 지침은 정보를 수집하고 애플리케이션에 대한 강력한 액세스 제어 및 권한 부여 시스템을 설계하는 데 첫 번째 단계로 사용할 수 있습니다. 다음 단계:

- 권한 부여 및 테넌트 격리 요구 사항을 검토하고, 애플리케이션에 맞는 액세스 제어 모델을 선택하세요.
- Amazon 검증 권한 또는 공개 정책 에이전트 (OPA) 를 사용하거나 사용자 지정 정책 엔진을 직접 작성하여 테스트용 개념 증명을 구축하십시오.
- 애플리케이션에서 PEP를 구현해야 하는 API와 위치를 식별하세요.

리소스

참조

- Amazon 검증 권한 설명서 (AWS 설명서)
- 승인을 위해 Amazon 검증 권한을 사용하는 방법 (AWS 블로그 게시물)
- Amazon 검증 권한을 사용하여 ASP.NET Core 앱을 위한 사용자 지정 권한 부여 정책 공급자 구현 (AWS 블로그 게시물)
- Amazon 검증 권한을 사용하여 PBAC를 통한 역할 및 권한AWS 관리 (블로그 게시물)
- 테넌트별 정책 저장소와 함께 Amazon 검증 권한을 사용한 SaaS 액세스 제어 (AWS 블로그 게시물)
- OPA 공식 설명서
- 기업이 가장 최근에 졸업한 CNCF 프로젝트를 수용해야 하는 이유 — 공개 정책 담당자 (포브스 기사, 자나키람 MSV, 2021년 2월 8일)
- 개방형 정책 에이전트를 사용하여 사용자 지정 Lambda 권한 부여자 생성 (블로그 게시물)AWS
- 개방형 정책 에이전트를 통해 AWS Cloud Development Kit를 사용하여 정책을 코드로 구현 (AWS 블로그 게시물)
- 코드형 정책을 AWS 통한 클라우드 거버넌스 및 규정 준수 (AWS 블로그 게시물)
- Amazon EKS에서 오픈 폴리시 에이전트 사용 (AWS 블로그 게시물)
- 오픈 폴리시 에이전트, Amazon 및 AWS Lambda(AWS 블로그 게시물) 를 사용하는 Amazon EventBridge ECS 코드로서의 규정 준수
- 쿠버네티스에 대한 정책 기반 대책 — 1부 (블로그 게시물)AWS
- API Gateway Lambda 권한 부여자 사용 (설명서)AWS

도구

- 시더 플레이그라운드 (브라우저에서 Cedar를 테스트하기 위한)
- 시더 깃허브 리포지토리
- 시더 언어 레퍼런스
- 레고 플레이그라운드 (브라우저에서 Rego를 테스트하기 위한)
- OPA 리포지토리 GitHub

파트너

- ID 및 액세스 관리 파트너
- 애플리케이션 보안 파트너
- 클라우드 거버넌스 파트너
- 보안 및 규정 준수 파트너
- 보안 운영 및 자동화 파트너
- 보안 엔지니어링 파트너

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
Amazon 검증 권한에 대한 세부 정보 및 예제 추가	Amazon 검증 권한을 사용하여 PDP를 구현하기 위한 자세한 설명, 예제 및 코드를 추가했습니다. 새 섹션에는 다음이 포함됩니다. • 아마존 검증 권한을 사용하여 PDP 구현 • Amazon 검증 권한을 위한 디자인 모델 • Amazon 검증 권한 멀티 테넌트 설계 고려 사항 • Amazon 검증 권한에서 PDP에 대한 외부 데이터 검색	2024년 5월 28일
명확한 정보	API 설계 모델을 기반으로 PEP를 사용한 분산 PDP를 명확히 했습니다.	2024년 1월 10일
새 서비스에 대한 간략한 정보가 추가되었습니다. AWS	OPA와 동일한 기능 및 이점을 제공하는 Amazon 검증 권한에 대한 정보가 추가되었습니다.	2023년 5월 22일
—	최초 게시	2021년 8월 17일

AWS 규범적 지침 용어집

다음은 규범적 지침에서 제공하는 AWS 전략, 가이드 및 패턴에서 일반적으로 사용되는 용어입니다. 용어집 항목을 제안하려면 용어집 끝에 있는 피드백 제공 링크를 사용하십시오.

숫자

7가지 전략

애플리케이션을 클라우드로 이전하기 위한 7가지 일반적인 마이그레이션 전략 이러한 전략은 Gartner가 2011년에 파악한 5가지 전략을 기반으로 하며 다음으로 구성됩니다.

- 리팩터링/리아키텍트 - 클라우드 네이티브 기능을 최대한 활용하여 애플리케이션을 이동하고 해당 아키텍처를 수정함으로써 민첩성, 성능 및 확장성을 개선합니다. 여기에는 일반적으로 운영 체제와 데이터베이스 이식이 포함됩니다. 예: 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL 호환 에디션으로 마이그레이션하십시오.
- 리플랫포밍(리프트 앤드 리셰이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 오라클용 Amazon RDS (Amazon RDS) 로 마이그레이션합니다. AWS 클라우드
- 재구매(드롭 앤드 숍) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리 (CRM) 시스템을 Salesforce.com으로 마이그레이션하십시오.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 의 EC2 인스턴스에서 Oracle로 마이그레이션합니다. AWS 클라우드
- 재배치(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 온프레미스 플랫폼에서 동일한 플랫폼의 클라우드 서비스로 서버를 마이그레이션합니다. 예: Microsoft Hyper-V 애플리케이션을 다음으로 마이그레이션하십시오. AWS
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중으로 연기하려는 애플리케이션과 비즈니스 차원에서 마이그레이션할 이유가 없어 유지하려는 레거시 애플리케이션이 여기에 포함될 수 있습니다.
- 사용 중지 - 소스 환경에서 더 이상 필요하지 않은 애플리케이션을 폐기하거나 제거합니다.

A

ABAC

속성 기반 액세스 제어를 참조하십시오.

추상화된 서비스

관리형 서비스를 참조하십시오.

산

원자성, 일관성, 격리성, 내구성을 참조하십시오.

능동-능동 마이그레이션

양방향 복제 도구 또는 이중 쓰기 작업을 사용하여 소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되고, 두 데이터베이스 모두 마이그레이션 중 연결 애플리케이션의 트랜잭션을 처리하는 데이터베이스 마이그레이션 방법입니다. 이 방법은 일회성 전환이 필요한 대신 소규모의 제어된 배치로 마이그레이션을 지원합니다. 더 유연하지만 액티브-패시브 마이그레이션보다 더 많은 작업이 필요합니다.

능동-수동 마이그레이션

소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되지만 소스 데이터베이스만 연결 애플리케이션의 트랜잭션을 처리하고 데이터는 대상 데이터베이스로 복제되는 데이터베이스 마이그레이션 방법입니다. 대상 데이터베이스는 마이그레이션 중 어떤 트랜잭션도 허용하지 않습니다.

집계 함수

행 그룹에서 연산을 수행하고 그룹에 대한 단일 반환값을 계산하는 SQL 함수입니다. 집계 함수의 예로는 및 등이 `SUM` 있습니다. `MAX`

AI

인공 지능을 참조하십시오.

AIOps

인공 지능 운영을 참조하십시오.

익명화

데이터세트에서 개인 정보를 영구적으로 삭제하는 프로세스입니다. 익명화는 개인 정보 보호에 도움이 될 수 있습니다. 익명화된 데이터는 더 이상 개인 데이터로 간주되지 않습니다.

안티 패턴

솔루션이 다른 솔루션보다 비생산적이거나 비효율적이거나 덜 효과적이어서 반복되는 문제에 자주 사용되는 솔루션입니다.

애플리케이션 제어

시스템을 멀웨어로부터 보호하기 위해 승인된 애플리케이션만 사용할 수 있는 보안 접근 방식입니다.

애플리케이션 포트폴리오

애플리케이션 구축 및 유지 관리 비용과 애플리케이션의 비즈니스 가치를 비롯하여 조직에서 사용하는 각 애플리케이션에 대한 세부 정보 모음입니다. 이 정보는 포트폴리오 검색 및 분석 프로세스의 핵심이며 마이그레이션, 현대화 및 최적화할 애플리케이션을 식별하고 우선순위를 정하는 데 도움이 됩니다.

인공 지능

컴퓨터 기술을 사용하여 학습, 문제 해결, 패턴 인식 등 일반적으로 인간과 관련된 인지 기능을 수행하는 것을 전문으로 하는 컴퓨터 과학 분야입니다. 자세한 내용은 What is Artificial Intelligence?를 참조하십시오.

인공 지능 운영(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AWS 마이그레이션 전략에서 AIOps가 사용되는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

비대칭 암호화

한 쌍의 키, 즉 암호화를 위한 퍼블릭 키와 복호화를 위한 프라이빗 키를 사용하는 암호화 알고리즘입니다. 퍼블릭 키는 복호화에 사용되지 않으므로 공유할 수 있지만 프라이빗 키에 대한 액세스는 엄격히 제한되어야 합니다.

원자성, 일관성, 격리성, 내구성(ACID)

오류, 정전 또는 기타 문제가 발생한 경우에도 데이터베이스의 데이터 유효성과 운영 신뢰성을 보장하는 소프트웨어 속성 세트입니다.

ABAC(속성 기반 액세스 제어)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서의 AWS ABAC for를 참조하십시오.

신뢰할 수 있는 데이터 소스

가장 신뢰할 수 있는 정보 소스로 간주되는 기본 버전의 데이터를 저장하는 위치입니다. 익명화, 편집 또는 가명화와 같은 데이터 처리 또는 수정의 목적으로 신뢰할 수 있는 데이터 소스의 데이터를 다른 위치로 복사할 수 있습니다.

가용 영역

다른 가용 영역의 장애로부터 격리되고 동일한 지역 내 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 별도의 위치. AWS 리전

AWS 클라우드 채택 프레임워크 (AWS CAF)

조직이 클라우드로 성공적으로 AWS 전환하기 위한 효율적이고 효과적인 계획을 개발하는 데 도움이 되는 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 관점이라고 하는 6가지 중점 영역, 즉 비즈니스, 사람, 거버넌스, 플랫폼, 보안, 운영으로 분류합니다. 비즈니스, 사람 및 거버넌스 관점은 비즈니스 기술과 프로세스에 초점을 맞추고, 플랫폼, 보안 및 운영 관점은 전문 기술과 프로세스에 중점을 둡니다. 예를 들어, 사람 관점은 인사(HR), 직원 배치 기능 및 인력 관리를 담당하는 이해관계자를 대상으로 합니다. 이러한 관점에서 AWS CAF는 조직이 성공적인 클라우드 채택을 준비할 수 있도록 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 AWS CAF 웹 사이트와 AWS CAF 백서를 참조하십시오.

AWS 워크로드 검증 프레임워크 (AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고 마이그레이션 전략을 권장하며 작업 예상치를 제공하는 도구입니다. AWS WQF는 () 에 포함됩니다. AWS Schema Conversion Tool AWS SCT데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

B

배드 봇

개인이나 조직을 방해하거나 피해를 입히려는 의도를 가진 봇입니다.

BCP

비즈니스 연속성 계획을 참조하십시오.

동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그온 시도, 의심스러운 API 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 Data in a behavior graph를 참조하십시오.

빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안도 참조하십시오.

바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, '이 제품은 책인가요, 자동차인가요?' 등의 문제를 예측해야 할 수 있습니다.

블룸 필터

요소가 세트의 멤버인지 여부를 테스트하는 데 사용되는 메모리 효율성이 높은 확률론적 데이터 구조입니다.

블루/그린(Blue/Green) 배포

서로 다르지만 동일한 환경을 두 개 만드는 배포 전략입니다. 현재 애플리케이션 버전을 한 환경 (파란색) 에서 실행하고 다른 환경 (녹색) 에서 새 애플리케이션 버전을 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 신속하게 롤백할 수 있습니다.

bot

인터넷을 통해 자동화된 작업을 실행하고 사람의 활동이나 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같은 일부 봇은 유용하거나 유용합니다. 배드 봇으로 알려진 일부 다른 봇은 개인이나 조직을 방해하거나 피해를 입히기 위한 것입니다.

봇넷

멀웨어에감염되어 봇 허더 또는 봇 운영자로 알려진 단일 당사자의 통제 하에 있는 봇 네트워크. 봇넷은 봇과 그 영향을 확장하는 가장 잘 알려진 메커니즘입니다.

브랜치

코드 리포지토리의 포함된 영역입니다. 리포지토리에 생성되는 첫 번째 브랜치가 기본 브랜치입니다. 기존 브랜치에서 새 브랜치를 생성한 다음 새 브랜치에서 기능을 개발하거나 버그를 수정할 수 있습니다. 기능을 구축하기 위해 생성하는 브랜치를 일반적으로 기능 브랜치라고 합니다. 기능을 출시할 준비가 되면 기능 브랜치를 기본 브랜치에 다시 병합합니다. 자세한 내용은 브랜치 정보 (문서) 를 참조하십시오. GitHub

브레이크 글래스 액세스

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스 권한이 없는 데이터에 빠르게 액세스할 수 AWS 계정 있는 수단입니다. 자세한 내용은 Well-Architected AWS 지침의 브레이크 글래스 절차 구현 표시기를 참조하십시오.

브라운필드 전략

사용자 환경의 기존 인프라 시스템 아키텍처에 브라운필드 전략을 채택할 때는 현재 시스템 및 인프라의 제약 조건을 중심으로 아키텍처를 설계합니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

버퍼 캐시

가장 자주 액세스하는 데이터가 저장되는 메모리 영역입니다.

사업 역량

기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 마이크로서비스 아키텍처 및 개발 결정은 비즈니스 역량에 따라 이루어질 수 있습니다. 자세한 내용은 백서의 AWS에서 컨테이너화된 마이크로서비스 실행의 비즈니스 역량 중심의 구성화 섹션을 참조하십시오.

비즈니스 연속성 계획(BCP)

대규모 마이그레이션과 같은 중단 이벤트가 운영에 미치는 잠재적 영향을 해결하고 비즈니스가 신속하게 운영을 재개할 수 있도록 지원하는 계획입니다.

C

CAF

클라우드 채택 프레임워크를 참조하십시오AWS .

카나리아 배포

최종 사용자에게 버전을 느리고 점진적으로 릴리스하는 것입니다. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

CCoE

클라우드 센터 오브 엑설런스를 참조하십시오.

CDC

변경 데이터 캡처를 참조하십시오.

변경 데이터 캡처(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 내용을 감사하거나 복제하여 동기화를 유지하는 등의 다양한 용도로 CDC를 사용할 수 있습니다.

카오스 엔지니어링

시스템의 복원력을 테스트하기 위해 의도적으로 장애나 장애를 일으키는 이벤트를 발생시키는 행위 AWS Fault Injection Service (AWS FIS) 를 사용하여 AWS 워크로드에 스트레스를 주는 실험을 수행하고 응답을 평가할 수 있습니다.

CI/CD

지속적 통합 및 지속적 전달을 참조하십시오.

분류

예측을 생성하는 데 도움이 되는 분류 프로세스입니다. 분류 문제에 대한 ML 모델은 이산 값을 예측합니다. 이산 값은 항상 서로 다릅니다. 예를 들어, 모델이 이미지에 자동차가 있는지 여부를 평가해야 할 수 있습니다.

클라이언트측 암호화

대상이 데이터를 AWS 서비스 수신하기 전에 데이터를 로컬로 암호화합니다.

클라우드 혁신 센터(CCoE)

클라우드 모범 사례 개발, 리소스 동원, 마이그레이션 타임라인 설정, 대규모 혁신을 통한 조직 선도 등 조직 전체에서 클라우드 채택 노력을 추진하는 다분야 팀입니다. 자세한 내용은 AWS 클라우드 기업 전략 블로그의 CCoE 게시물을 참조하십시오.

클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 엣지 컴퓨팅 기술과 연결됩니다.

클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 클라우드 운영 모델 구축을 참조하십시오.

클라우드 채택 단계

조직이 마이그레이션할 때 일반적으로 거치는 4단계는 다음과 같습니다. AWS 클라우드

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 기반 – 클라우드 채택 확장을 위한 기초 투자(예: 랜딩 존 생성, CCoE 정의, 운영 모델 구축)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

Stephen Orban은 기업 전략 블로그의 클라우드 우선주의를 향한 여정 및 채택 단계에 대한 블로그 게시물에서 이러한 단계를 정의했습니다. AWS 클라우드 이들이 AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오.

CMDB

구성 관리 데이터베이스를 참조하십시오.

코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반 클라우드 리포지토리에는 또는 이 포함됩니다 GitHub . AWS CodeCommit코드의 각 버전을 브랜치라고 합니다. 마이크로서비스 구조에서 각 리포지토리는 단일 기능 전용입니다. 단일 CI/CD 파이프라인은 여러 리포지토리를 사용할 수 있습니다.

콜드 캐시

비어 있거나, 제대로 채워지지 않았거나, 오래되었거나 관련 없는 데이터를 포함하는 버퍼 캐시입니다. 주 메모리나 디스크에서 데이터베이스 인스턴스를 읽어야 하기 때문에 성능에 영향을 미치며, 이는 버퍼 캐시에서 읽는 것보다 느립니다.

콜드 데이터

거의 액세스되지 않고 일반적으로 과거 데이터인 데이터. 이런 종류의 데이터를 쿼리할 때는 일반적으로 느린 쿼리가 허용됩니다. 이 데이터를 성능이 낮고 비용이 저렴한 스토리지 계층 또는 클래스로 옮기면 비용을 절감할 수 있습니다.

컴퓨터 비전 (CV)

기계 학습을 사용하여 디지털 이미지 및 비디오와 같은 시각적 형식에서 정보를 분석하고 추출하는 AI 분야. 예를 들어 AWS Panorama 는 온프레미스 카메라 네트워크에 CV를 추가하는 디바이스를 제공하고, SageMaker Amazon은 CV용 이미지 처리 알고리즘을 제공합니다.

구성 드리프트

워크로드의 경우 구성이 예상 상태에서 변경됩니다. 이로 인해 워크로드가 규정을 준수하지 않게 될 수 있으며, 일반적으로 점진적이고 의도하지 않은 방식으로 진행됩니다.

구성 관리 데이터베이스(CMDB)

하드웨어 및 소프트웨어 구성 요소와 해당 구성을 포함하여 데이터베이스와 해당 IT 환경에 대한 정보를 저장하고 관리하는 리포지토리입니다. 일반적으로 마이그레이션의 포트폴리오 검색 및 분석 단계에서 CMDB의 데이터를 사용합니다.

규정 준수 팩

AWS Config 규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 규칙 및 수정 조치 모음입니다. YAML 템플릿을 사용하여 한 AWS 계정 및 지역 또는 조직 전체에 단일 엔티티로 적합성 팩을 배포할 수 있습니다. 자세한 내용은 설명서의 적합성 팩을 참조하십시오. AWS Config

지속적 통합 및 지속적 전달(CI/CD)

소프트웨어 릴리스 프로세스의 소스, 빌드, 테스트, 스테이징 및 프로덕션 단계를 자동화하는 프로세스입니다. CI/CD는 일반적으로 파이프라인으로 설명됩니다. CI/CD를 통해 프로세스를 자동화하고, 생산성을 높이고, 코드 품질을 개선하고, 더 빠르게 제공할 수 있습니다. 자세한 내용은 지속적 전달의 이점을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 지속적 전달(Continuous Delivery)과 지속적인 개발을 참조하십시오.

CV

컴퓨터 비전을 참조하십시오.

D

저장 데이터

스토리지에 있는 데이터와 같이 네트워크에 고정되어 있는 데이터입니다.

데이터 분류

중요도와 민감도를 기준으로 네트워크의 데이터를 식별하고 분류하는 프로세스입니다. 이 프로세스는 데이터에 대한 적절한 보호 및 보존 제어를 결정하는 데 도움이 되므로 사이버 보안 위험 관리 전략의 중요한 구성 요소입니다. 데이터 분류는 AWS Well-Architected 프레임워크의 보안 핵심 요소입니다. 자세한 내용은 데이터 분류를 참조하십시오.

데이터 드리프트

프로덕션 데이터와 ML 모델 학습에 사용된 데이터 간의 상당한 차이 또는 시간 경과에 따른 입력 데이터의 의미 있는 변화. 데이터 드리프트는 ML 모델 예측의 전반적인 품질, 정확성 및 공정성을 저하시킬 수 있습니다.

전송 중 데이터

네트워크를 통과하고 있는 데이터입니다. 네트워크 리소스 사이를 이동 중인 데이터를 예로 들 수 있습니다.

데이터 메시

중앙 집중식 관리 및 거버넌스와 함께 분산되고 분산된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터 최소화를 실천하면 개인 정보 보호 위험, 비용 및 분석에 따른 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

데이터 경계

신뢰할 수 있는 ID만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스하도록 하는 데 도움이 되는 AWS 환경 내 일련의 예방 가드레일입니다. 자세한 내용은 데이터 경계 구축을 참조하십시오. AWS

데이터 사전 처리

원시 데이터를 ML 모델이 쉽게 구문 분석할 수 있는 형식으로 변환하는 것입니다. 데이터를 사전 처리한다는 것은 특정 열이나 행을 제거하고 누락된 값, 일관성이 없는 값 또는 중복 값을 처리함을 의미할 수 있습니다.

데이터 출처

라이프사이클 전반에 걸쳐 데이터의 출처와 기록을 추적하는 프로세스(예: 데이터 생성, 전송, 저장 방법).

데이터 주체

데이터를 수집 및 처리하는 개인입니다.

데이터 웨어하우스

분석과 같은 비즈니스 인텔리전스를 지원하는 데이터 관리 시스템. 데이터 웨어하우스에는 일반적으로 대량의 과거 데이터가 포함되며 일반적으로 쿼리 및 분석에 사용됩니다.

데이터 정의 언어(DDL)

데이터베이스에서 테이블 및 객체의 구조를 만들거나 수정하기 위한 명령문 또는 명령입니다.

데이터베이스 조작 언어(DML)

데이터베이스에서 정보를 수정(삽입, 업데이트 및 삭제)하기 위한 명령문 또는 명령입니다.

DDL

데이터베이스 정의 언어를 참조하십시오.

딥 앙상블

예측을 위해 여러 딥 러닝 모델을 결합하는 것입니다. 딥 앙상블을 사용하여 더 정확한 예측을 얻거나 예측의 불확실성을 추정할 수 있습니다.

딥 러닝

여러 계층의 인공 신경망을 사용하여 입력 데이터와 관심 대상 변수 간의 매핑을 식별하는 ML 하위 분야입니다.

defense-in-depth

네트워크와 그 안의 데이터 기밀성, 무결성 및 가용성을 보호하기 위해 컴퓨터 네트워크 전체에 일련의 보안 메커니즘과 제어를 신중하게 계층화하는 정보 보안 접근 방식입니다. 이 전략을 채택하면 AWS Organizations 구조의 여러 계층에 AWS여러 컨트롤을 추가하여 리소스를 보호하는 데 도움이 됩니다. 예를 들어 다단계 인증, 네트워크 세분화, 암호화를 결합한 defense-in-depth 접근 방식을 사용할 수 있습니다.

위임된 관리자

에서 AWS Organizations호환 가능한 서비스는 AWS 구성원 계정을 등록하여 조직의 계정을 관리하고 해당 서비스에 대한 권한을 관리할 수 있습니다. 이러한 계정을 해당 서비스의 위임된 관리자라고 합니다. 자세한 내용과 호환되는 서비스 목록은 AWS Organizations 설명서의 AWS Organizations와 함께 사용할 수 있는 AWS 서비스를 참조하십시오.

배포

대상 환경에서 애플리케이션, 새 기능 또는 코드 수정 사항을 사용할 수 있도록 하는 프로세스입니다. 배포에는 코드 베이스의 변경 사항을 구현한 다음 애플리케이션 환경에서 해당 코드베이스를 구축하고 실행하는 작업이 포함됩니다.

개발 환경

환경을 참조하십시오.

탐지 제어

이벤트 발생 후 탐지, 기록 및 알림을 수행하도록 설계된 보안 제어입니다. 이러한 제어는 기존의 예방적 제어를 우회한 보안 이벤트를 알리는 2차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 Detective controls를 참조하십시오.

개발 가치 흐름 매핑 (DVSM)

소프트웨어 개발 라이프사이클에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다. DVSM은 원래 린 제조 방식을 위해 설계된 가치 흐름 매핑 프로세스를 확장합니다. 소프트웨어 개발 프로세스를 통해 가치를 창출하고 이동하는 데 필요한 단계와 팀에 중점을 둡니다.

디지털 트윈

건물, 공장, 산업 장비 또는 생산 라인과 같은 실제 시스템을 가상으로 표현한 것입니다. 디지털 트윈은 예측 유지 보수, 원격 모니터링, 생산 최적화를 지원합니다.

치수 표

스타 스키마에서 팩트 테이블의 양적 데이터에 대한 데이터 속성을 포함하는 작은 테이블입니다. 차원 테이블 속성은 일반적으로 텍스트처럼 동작하는 텍스트 필드 또는 불연속형 숫자입니다. 이러한 속성은 일반적으로 쿼리 제한, 필터링 및 결과 집합 레이블 지정에 사용됩니다.

재해

워크로드 또는 시스템이 기본 배포 위치에서 비즈니스 목표를 달성하지 못하게 방해하는 이벤트입니다. 이러한 이벤트는 자연재해, 기술적 오류, 의도하지 않은 구성 오류 또는 멀웨어 공격과 같은 사람의 행동으로 인한 결과일 수 있습니다.

재해 복구(DR)

재해로 인한 다운타임과 데이터 손실을 최소화하기 위해 사용하는 전략과 프로세스입니다. 자세한 내용은 워크로드의 재해 복구 AWS: AWS Well-Architected 프레임워크에서의 클라우드 복구를 참조하십시오.

DML

데이터베이스 조작 언어를 참조하십시오.

도메인 기반 설계

구성 요소를 각 구성 요소가 제공하는 진화하는 도메인 또는 핵심 비즈니스 목표에 연결하여 복잡한 소프트웨어 시스템을 개발하는 접근 방식입니다. 이 개념은 에릭 에반스에 의해 그의 저서인 도메인 기반 디자인: 소프트웨어 중심의 복잡성 해결(Boston: Addison-Wesley Professional, 2003)에서 소개되었습니다. Strangler Fig 패턴과 함께 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

DR

재해 복구를 참조하십시오.

드리프트 감지

기준 구성으로부터의 편차 추적. 예를 들어 시스템 리소스의 편차를 감지하는 AWS CloudFormation 데 사용하거나 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 착륙 지대의 변경 사항을 탐지하는 AWS Control Tower 데 사용할 수 있습니다.

DVSM

개발 가치 흐름 매핑을 참조하십시오.

E

EDA

탐색적 데이터 분석을 참조하십시오.

엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 클라우드 컴퓨팅과 비교할 때 엣지 컴퓨팅은 통신 대기 시간을 줄이고 응답 시간을 개선할 수 있습니다.

암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 암호문으로 변환하는 컴퓨팅 프로세스입니다.

암호화 키

암호화 알고리즘에 의해 생성되는 무작위 비트의 암호화 문자열입니다. 키의 길이는 다양할 수 있으며 각 키는 예측할 수 없고 고유하게 설계되었습니다.

엔디안

컴퓨터 메모리에 바이트가 저장되는 순서입니다. 빅 엔디안 시스템은 가장 중요한 바이트를 먼저 저장합니다. 리틀 엔디안 시스템은 가장 덜 중요한 바이트를 먼저 저장합니다.

엔드포인트

서비스 엔드포인트를 참조하십시오.

엔드포인트 서비스

Virtual Private Cloud(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 다른 주체 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 부여하여 엔드포인트 서비스를 생성하고 권한을 부여할 수 있습니다. AWS PrivateLink 이러한 계정 또는 보안 주체는 인터페이스 VPC 엔드포인트를 생성하여 엔드포인트 서비스에 비공개로 연결할 수 있습니

다. 자세한 내용은 Amazon Virtual Private Cloud(VPC) 설명서의 엔드포인트 서비스 생성을 참조하십시오.

ERP (전사적 자원 관리)

기업의 주요 비즈니스 프로세스 (예: 회계, MES, 프로젝트 관리) 를 자동화하고 관리하는 시스템입니다.

봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 봉투 암호화를 참조하십시오.

환경

실행 중인 애플리케이션의 인스턴스입니다. 다음은 클라우드 컴퓨팅의 일반적인 환경 유형입니다.

- 개발 환경 - 애플리케이션 유지 관리를 담당하는 핵심 팀만 사용할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. 개발 환경은 변경 사항을 상위 환경으로 승격하기 전에 테스트하는 데 사용됩니다. 이러한 유형의 환경을 테스트 환경이라고도 합니다.
- 하위 환경 - 초기 빌드 및 테스트에 사용되는 환경을 비롯한 애플리케이션의 모든 개발 환경입니다.
- 프로덕션 환경 - 최종 사용자가 액세스할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. CI/CD 파이프라인에서 프로덕션 환경이 마지막 배포 환경입니다.
- 상위 환경 - 핵심 개발 팀 이외의 사용자가 액세스할 수 있는 모든 환경입니다. 프로덕션 환경, 프로덕션 이전 환경 및 사용자 수용 테스트를 위한 환경이 여기에 포함될 수 있습니다.

에픽

애자일 방법론에서 작업을 구성하고 우선순위를 정하는 데 도움이 되는 기능적 범주입니다. 에픽은 요구 사항 및 구현 작업에 대한 개괄적인 설명을 제공합니다. 예를 들어 AWS CAF 보안 에픽에는 ID 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호, 사고 대응 등이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 프로그램 구현 가이드를 참조하십시오.

ERP

엔터프라이즈 리소스 계획을 참조하십시오.

탐색 데이터 분석(EDA)

데이터 세트를 분석하여 주요 특성을 파악하는 프로세스입니다. 데이터를 수집 또는 집계한 다음 초기 조사를 수행하여 패턴을 찾고, 이상을 탐지하고, 가정을 확인합니다. EDA는 요약 통계를 계산하고 데이터 시각화를 생성하여 수행됩니다.

F

팩트 테이블

[스타 스키마의](#) 중앙 테이블. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블에는 측정값이 포함된 열과 차원 테이블의 외부 키가 포함된 열 등 두 가지 유형의 열이 포함됩니다.

빨리 실패하세요

빈번하고 점진적인 테스트를 통해 개발 라이프사이클을 단축하는 철학. 이는 애자일 접근 방식의 중요한 부분입니다.

장애 격리 경계

장애 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역 AWS 리전, 컨트롤 플레인 또는 데이터 플레인과 같은 경계 AWS 클라우드자세한 내용은 [AWS 장애 격리](#) 경계를 참조하십시오.

기능 브랜치

[브랜치를](#) 참조하십시오.

기능

예측에 사용하는 입력 데이터입니다. 예를 들어, 제조 환경에서 기능은 제조 라인에서 주기적으로 캡처되는 이미지일 수 있습니다.

기능 중요도

모델의 예측에 특성이 얼마나 중요한지를 나타냅니다. 이는 일반적으로 SHAP(Shapley Additive Descriptions) 및 통합 그레디언트와 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 [다음을AWS사용한 기계 학습 모델 해석 가능성을](#) 참조하십시오.

기능 변환

추가 소스로 데이터를 보강하거나, 값을 조정하거나, 단일 데이터 필드에서 여러 정보 세트를 추출하는 등 ML 프로세스를 위해 데이터를 최적화하는 것입니다. 이를 통해 ML 모델이 데이터를 활용할 수 있습니다. 예를 들어, 날짜 '2021-05-27 00:15:37'을 '2021년', '5월', '목', '15일'로 분류하면 학습 알고리즘이 다양한 데이터 구성 요소와 관련된 미묘한 패턴을 학습하는 데 도움이 됩니다.

FGAC

[세분화된 액세스 제어를](#) 참조하십시오.

세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

플래시컷 마이그레이션

단계별 접근 방식 대신 변경 데이터 캡처를 통한 지속적인 데이터 복제를 통해 최단 시간에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

G

지리적 차단

지리적 제한을 참조하십시오.

지리적 제한(지리적 차단)

CloudFrontAmazon에서는 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 옵션을 제공합니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 설명서의 콘텐츠의 지리적 배포 제한을 참조하십시오. CloudFront

Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 트렁크 기반 워크플로는 현대적이고 선호되는 접근 방식입니다.

브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 브라운필드라고도 하는 기존 인프라와의 호환성 제한 없이 모든 새로운 기술을 선택할 수 있습니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

가드레일

조직 단위(OU) 전체에서 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 중요 규칙입니다. 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책과 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, 이들은, Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

H

하

고가용성을 확인하세요.

이기종 데이터베이스 마이그레이션

다른 데이터베이스 엔진을 사용하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Oracle에서 Amazon Aurora로) 이기종 마이그레이션은 일반적으로 리아키텍트 작업의 일부이며 스키마를 변환하는 것은 복잡한 작업일 수 있습니다.AWS 는 스키마 변환에 도움이 되는 AWS SCT를 제공합니다.

높은 가용성(HA)

문제나 재해 발생 시 개입 없이 지속적으로 운영할 수 있는 워크로드의 능력. HA 시스템은 자동으로 장애 조치되고, 지속적으로 고품질 성능을 제공하고, 성능에 미치는 영향을 최소화하면서 다양한 부하와 장애를 처리하도록 설계되었습니다.

히스토리언 현대화

제조 산업의 요구 사항을 더 잘 충족하도록 운영 기술(OT) 시스템을 현대화하고 업그레이드하는 데 사용되는 접근 방식입니다. 히스토리언은 공장의 다양한 출처에서 데이터를 수집하고 저장하는 데 사용되는 일종의 데이터베이스입니다.

동종 데이터베이스 마이그레이션

동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Microsoft SQL Server에서 Amazon RDS for SQL Server로) 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫포밍 작업의 일부입니다. 네이티브 데이터베이스 유틸리티를 사용하여 스키마를 마이그레이션할 수 있습니다.

핫 데이터

자주 액세스하는 데이터(예: 실시간 데이터 또는 최근 번역 데이터). 일반적으로 이 데이터에는 빠른 쿼리 응답을 제공하기 위한 고성능 스토리지 계층 또는 클래스가 필요합니다.

핫픽스

프로덕션 환경의 중요한 문제를 해결하기 위한 긴급 수정입니다. 긴급성 때문에 핫픽스는 일반적으로 일반적인 DevOps 릴리스 워크플로 외부에서 만들어집니다.

하이퍼케어 기간

전환 직후 마이그레이션 팀이 문제를 해결하기 위해 클라우드에서 마이그레이션된 애플리케이션을 관리하고 모니터링하는 기간입니다. 일반적으로 이 기간은 1~4일입니다. 하이퍼케어 기간이 끝나면 마이그레이션 팀은 일반적으로 애플리케이션에 대한 책임을 클라우드 운영 팀에 넘깁니다.

I

IaC

인프라를 코드로 보세요.

자격 증명 기반 정책

환경 내에서 권한을 정의하는 하나 이상의 IAM 보안 주체에 연결된 정책입니다. AWS 클라우드

유휴 애플리케이션

90일 동안 평균 CPU 및 메모리 사용량이 5~20%인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하거나 온프레미스에 유지하는 것이 일반적입니다.

IIoT

산업용 사물 인터넷을 참조하십시오.

불변의 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드용 새 인프라를 배포하는 모델입니다. 변경 불가능한 인프라는 기본적으로 변경 가능한 인프라보다 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 Well-Architected AWS 프레임워크의 변경 불가능한 인프라를 사용한 배포 모범 사례를 참조하십시오.

인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 VPC는 애플리케이션 외부에서 네트워크 연결을 허용, 검사 및 라우팅합니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

증분 마이그레이션

한 번에 전체 전환을 수행하는 대신 애플리케이션을 조금씩 마이그레이션하는 전환 전략입니다. 예를 들어, 처음에는 소수의 마이크로서비스나 사용자만 새 시스템으로 이동할 수 있습니다. 모든 것

이 제대로 작동하는지 확인한 후에는 레거시 시스템을 폐기할 수 있을 때까지 추가 마이크로서비스 또는 사용자를 점진적으로 이동할 수 있습니다. 이 전략을 사용하면 대규모 마이그레이션과 관련된 위험을 줄일 수 있습니다.

Industry 4.0

Klaus Schwab이 연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통한 제조 프로세스의 현대화를 지칭하기 위해 2016년 도입한 용어입니다.

인프라

애플리케이션의 환경 내에 포함된 모든 리소스와 자산입니다.

코드형 인프라(IaC)

구성 파일 세트를 통해 애플리케이션의 인프라를 프로비저닝하고 관리하는 프로세스입니다. IaC는 새로운 환경의 반복 가능성, 신뢰성 및 일관성을 위해 인프라 관리를 중앙 집중화하고, 리소스를 표준화하고, 빠르게 확장할 수 있도록 설계되었습니다.

산업용 사물 인터넷(IIoT)

제조, 에너지, 자동차, 의료, 생명과학, 농업 등의 산업 부문에서 인터넷에 연결된 센서 및 디바이스의 사용 자세한 내용은 산업용 사물 인터넷(IoT) 디지털 트랜스포메이션 전략 구축을 참조하십시오.

검사 VPC

AWS 다중 계정 아키텍처에서 VPC (동일하거나 AWS 리전다른), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 관리하는 중앙 집중식 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

사물 인터넷(IoT)

인터넷이나 로컬 통신 네트워크를 통해 다른 디바이스 및 시스템과 통신하는 센서 또는 프로세서가 내장된 연결된 물리적 객체의 네트워크 자세한 내용은 IoT란?을 참조하십시오.

해석력

모델의 예측이 입력에 따라 어떻게 달라지는지를 사람이 이해할 수 있는 정도를 설명하는 기계 학습 모델의 특성입니다. 자세한 내용은 Machine learning model interpretability with AWS를 참조하십시오.

IoT

사물 인터넷을 참조하십시오.

IT 정보 라이브러리(TIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞게 조정하기 위한 일련의 모범 사례 ITIL은 ITSM의 기반을 제공합니다.

IT 서비스 관리(TSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 운영을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

ITIL

IT 정보 라이브러리를 참조하십시오.

ITSM

IT 서비스 관리를 참조하십시오.

L

레이블 기반 액세스 제어(LBAC)

사용자 및 데이터 자체에 각각 보안 레이블 값을 명시적으로 할당하는 필수 액세스 제어(MAC)를 구현한 것입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

랜딩 존

Landing Zone은 확장 가능하고 안전한 잘 설계된 다중 계정 AWS 환경입니다. 조직은 여기에서부터 보안 및 인프라 환경에 대한 확신을 가지고 워크로드와 애플리케이션을 신속하게 시작하고 배포할 수 있습니다. 랜딩 존에 대한 자세한 내용은 안전하고 확장 가능한 다중 계정 AWS 환경 설정을 참조하십시오.

대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

LBAC

레이블 기반 액세스 제어를 참조하십시오.

최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 최소 권한 적용을 참조하십시오.

리프트 앤드 시프트

7 R을 참조하십시오.

리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안 참조.

하위 환경

환경 참조.

M

기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공 지능의 한 유형입니다. ML은 사물 인터넷(IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 기계 학습을 참조하십시오.

기본 브랜치

브랜치 참조.

악성 코드

컴퓨터 보안 또는 개인 정보를 침해하도록 설계된 소프트웨어 멀웨어는 컴퓨터 시스템을 방해하거나, 민감한 정보를 유출하거나, 무단 액세스를 얻을 수 있습니다. 멀웨어의 예로는 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

매니지드 서비스

AWS 서비스 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하며 사용자는 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. 관리형 서비스의 예로는 아마존 심플 스토리지 서비스 (Amazon S3) 와 아마존 DynamoDB가 있습니다. 이러한 서비스를 추상화된 서비스라고도 합니다.

제조 실행 시스템 (MES)

제조 현장에서 원자재를 완제품으로 전환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

MAP

Migration Acceleration 프로그램을 참조하십시오.

기구

도구를 만들고 도구 채택을 유도한 다음 결과를 검토하여 조정하는 전체 프로세스입니다. 메커니즘은 작동하면서 자체적으로 강화되고 개선되는 사이클입니다. 자세한 내용은 AWS Well-Architected 프레임워크에서의 메커니즘 구축을 참조하십시오.

멤버 계정

조직의 일부인 관리 계정을 AWS 계정 제외한 모든 계정 AWS Organizations하나의 계정은 한 번에 하나의 조직 멤버만 될 수 있습니다.

MES

제조 실행 시스템을 참조하십시오.

메시지 큐 텔레메트리 전송 (MQTT)

퍼블리시/구독 패턴을 기반으로 하는 리소스가 제한된 IoT 디바이스를 위한 경량 machine-to-machine (M2M) 통신 프로토콜입니다.

마이크로서비스

잘 정의된 API를 통해 통신하고 일반적으로 소규모 자체 팀이 소유하는 소규모 독립 서비스입니다. 예를 들어, 보험 시스템에는 영업, 마케팅 등의 비즈니스 역량이나 구매, 청구, 분석 등의 하위 영역에 매핑되는 마이크로 서비스가 포함될 수 있습니다. 마이크로서비스의 이점으로 민첩성, 유연한 확장, 손쉬운 배포, 재사용 가능한 코드, 복원력 등이 있습니다. 자세한 내용은 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하십시오. AWS

마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량 API를 사용하여 잘 정의된 인터페이스를 통해 통신합니다. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 마이크로서비스 구현을 참조하십시오. AWS

Migration Acceleration Program(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄할 수 있도록 컨설팅 지원, 교육 및 서비스를 제공하는 AWS 프로그램입니다. MAP에는 레거시 마이그레이션을 체계적인 방식으로 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하는 도구 세트가 포함되어 있습니다.

대규모 마이그레이션

애플리케이션 포트폴리오의 대다수를 웨이브를 통해 클라우드로 이동하는 프로세스로, 각 웨이브에서 더 많은 애플리케이션이 더 빠른 속도로 이동합니다. 이 단계에서는 이전 단계에서 배운 모범 사례와 교훈을 사용하여 팀, 도구 및 프로세스의 마이그레이션 팩토리를 구현하여 자동화 및 민첩한 제공을 통해 워크로드 마이그레이션을 간소화합니다. 이것은 AWS 마이그레이션 전략의 세 번째 단계입니다.

마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자 및 스프린트에서 일하는 DevOps 전문가가 포함됩니다. 엔터프라이즈 애플리케이션 포트폴리오의 20~50%는 공장 접근 방식으로 최적화할 수 있는 반복되는 패턴으로 구성되어 있습니다. 자세한 내용은 이 콘텐츠 세트의 클라우드 마이그레이션 팩토리 가이드와 마이그레이션 팩토리에 대한 설명을 참조하십시오.

마이그레이션 메타데이터

마이그레이션을 완료하는 데 필요한 애플리케이션 및 서버에 대한 정보 각 마이그레이션 패턴에는 서로 다른 마이그레이션 메타데이터 세트가 필요합니다. 마이그레이션 메타데이터의 예로는 대상 서브넷, 보안 그룹, 계정 등이 있습니다. AWS

마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: 애플리케이션 마이그레이션 서비스를 사용하여 Amazon EC2로 AWS 마이그레이션을 재호스팅합니다.

Migration Portfolio Assessment(MPA)

로 마이그레이션하기 위한 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다. AWS 클라우드 MPA는 상세한 포트폴리오 평가(서버 적정 규모 조정, 가격 책정, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선순위 지정, 웨이브 계획)을 제공합니다. MPA 도구 (로그인 필요) 는 모든 컨설턴트와 APN 파트너 AWS 컨설턴트에게 무료로 제공됩니다.

마이그레이션 준비 상태 평가(MRA)

CAF를 사용하여 조직의 클라우드 준비 상태에 대한 통찰력을 얻고, 강점과 약점을 파악하고, 식별된 격차를 해소하기 위한 실행 계획을 수립하는 프로세스입니다. AWS 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오. MRA는 AWS 마이그레이션 전략의 첫 번째 단계입니다.

마이그레이션 전략

워크로드를 로 마이그레이션하는 데 사용된 접근 방식. AWS 클라우드자세한 내용은 이 용어집의 7R 항목 및 대규모 마이그레이션 가속화를 위한 조직 동원을 참조하십시오.

ML

기계 학습을 참조하십시오.

현대화

비용을 절감하고 효율성을 높이고 혁신을 활용하기 위해 구식(레거시 또는 모놀리식) 애플리케이션과 해당 인프라를 클라우드의 민첩하고 탄력적이고 가용성이 높은 시스템으로 전환하는 것입니다. 자세한 내용은 의 AWS 클라우드애플리케이션 현대화 전략을 참조하십시오.

현대화 준비 상태 평가

조직 애플리케이션의 현대화 준비 상태를 파악하고, 이점, 위험 및 종속성을 식별하고, 조직이 해당 애플리케이션의 향후 상태를 얼마나 잘 지원할 수 있는지를 확인하는 데 도움이 되는 평가입니다. 평가 결과는 대상 아키텍처의 청사진, 현대화 프로세스의 개발 단계와 마일스톤을 자세히 설명하는 로드맵 및 파악된 격차를 해소하기 위한 실행 계획입니다. 자세한 내용은 에서 애플리케이션의 현대화 준비 상태 평가를 참조하십시오. AWS 클라우드

모놀리식 애플리케이션(모놀리식 유형)

긴밀하게 연결된 프로세스를 사용하여 단일 서비스로 실행되는 애플리케이션입니다. 모놀리식 애플리케이션에는 몇 가지 단점이 있습니다. 한 애플리케이션 기능에 대한 수요가 급증하면 전체 아키텍처 규모를 조정해야 합니다. 코드 베이스가 커지면 모놀리식 애플리케이션의 기능을 추가하거나 개선하는 것도 더 복잡해집니다. 이러한 문제를 해결하기 위해 마이크로서비스 아키텍처를 사용할 수 있습니다. 자세한 내용은 마이크로서비스로 모놀리식 유형 분해를 참조하십시오.

MPA

마이그레이션 포트폴리오 평가를 참조하십시오.

MQTT

메시지 큐 원격 분석 전송을 참조하십시오.

멀티클래스 분류

여러 클래스에 대한 예측(2개 이상의 결과 중 하나 예측)을 생성하는 데 도움이 되는 프로세스입니다. 예를 들어, ML 모델이 '이 제품은 책인가요, 자동차인가요, 휴대폰인가요?' 또는 '이 고객이 가장 관심을 갖는 제품 범주는 무엇인가요?'라고 물을 수 있습니다.

변경 가능한 인프라

프로덕션 워크로드를 위해 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 안정성 및 예측 가능성을 개선하기 위해 AWS Well-Architected Framework는 변경 불가능한 인프라를 모범 사례로 사용할 것을 권장합니다.

O

OAC

원본 액세스 제어를 참조하십시오.

좋아요

원본 액세스 ID를 참조하십시오.

OCM

조직 변경 관리를 참조하십시오.

오프라인 마이그레이션

마이그레이션 프로세스 중 소스 워크로드가 중단되는 마이그레이션 방법입니다. 이 방법은 가동 중지 증가를 수반하며 일반적으로 작고 중요하지 않은 워크로드에 사용됩니다.

OI

운영 통합을 참조하십시오.

안녕하세요.

운영 수준 계약을 참조하십시오.

온라인 마이그레이션

소스 워크로드를 오프라인 상태로 전환하지 않고 대상 시스템에 복사하는 마이그레이션 방법입니다. 워크로드에 연결된 애플리케이션은 마이그레이션 중에도 계속 작동할 수 있습니다. 이 방법은 가동 중지 차단 또는 최소화를 수반하며 일반적으로 중요한 프로덕션 워크로드에 사용됩니다.

OPC-UA

오픈 프로세스 커뮤니케이션 - 통합 아키텍처를 참조하십시오.

오픈 프로세스 커뮤니케이션 - 통합 아키텍처 (OPC-UA)

산업 machine-to-machine 자동화를 위한 (M2M) 통신 프로토콜. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계와 함께 상호 운용성 표준을 제공합니다.

운영 수준 협약(OLA)

서비스 수준에 관한 계약(SLA)을 지원하기 위해 직무 IT 그룹이 서로에게 제공하기로 약속한 내용을 명확히 하는 계약입니다.

운영 준비 검토 (ORR)

인시던트 및 발생 가능한 실패의 범위를 이해, 평가, 예방 또는 줄이는 데 도움이 되는 질문 및 관련 모범 사례로 구성된 체크리스트입니다. 자세한 내용은 Well-Architected AWS 프레임워크의 운영 준비 상태 검토 (ORR) 를 참조하십시오.

운영 기술 (OT)

물리적 환경과 함께 작동하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템. 제조 분야에서는 OT와 정보 기술 (IT) 시스템의 통합이 인더스트리 4.0 혁신의 핵심 초점입니다.

운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 운영 통합 가이드를 참조하십시오.

조직 트레일

이를 통해 AWS CloudTrail 생성되는 트레일은 조직 AWS 계정 내 모든 사용자의 모든 이벤트를 기록합니다. AWS Organizations이 트레일은 조직에 속한 각 AWS 계정 에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 CloudTrail 설명서에서 조직을 위한 트레일 만들기를 참조하십시오.

조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 중대하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM은 변화 채택을 가속화하고, 과도기적 문제를 해결하고, 문화 및 조직적 변화를 주도함으로써 조직이 새로운 시스템 및 전략을 준비하고 전환할 수 있도록 지원합니다. 클라우드 채택 프로젝트에 필요한 변화 속도 때문에 AWS 마이그레이션 전략에서는 이 프레임워크를 사용자 가속화라고 합니다. 자세한 내용은 사용 가이드를 참조하십시오.

오리진 액세스 제어(OAC)

CloudFront에서는 Amazon Simple Storage Service (Amazon S3) 콘텐츠의 보안을 위해 액세스를 제한하는 향상된 옵션을 제공합니다. OAC는 모든 S3 버킷 AWS 리전, AWS KMS (SSE-KMS) 를 사용한 서버 측 암호화, S3 버킷에 대한 동적 및 요청을 모두 지원합니다. PUT DELETE

오리진 액세스 ID(OAI)

CloudFront에서는 Amazon S3 콘텐츠 보안을 위해 액세스를 제한하는 옵션입니다. OAI를 사용하면 Amazon S3가 인증할 수 있는 보안 주체를 CloudFront 생성합니다. 인증된 보안 주체는 특정 배

포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. CloudFront 더 세분화되고 향상된 액세스 제어를 제공하는 OAC도 참조하십시오.

또는

운영 준비 상태 검토를 참조하십시오.

아니요

운영 기술을 참조하십시오.

아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작되는 네트워크 연결을 처리하는 VPC입니다. AWS Security Reference Architecture에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

P

권한 경계

사용자나 역할이 가질 수 있는 최대 권한을 설정하기 위해 IAM 보안 주체에 연결되는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 권한 경계를 참조하십시오.

개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. PII의 예로는 이름, 주소, 연락처 정보 등이 있습니다.

PII

개인 식별 정보를 참조하십시오.

플레이북

클라우드에서 핵심 운영 기능을 제공하는 등 마이그레이션과 관련된 작업을 캡처하는 일련의 사전 정의된 단계입니다. 플레이북은 스크립트, 자동화된 런북 또는 현대화된 환경을 운영하는 데 필요한 프로세스나 단계 요약의 형태를 취할 수 있습니다.

PLC

프로그래머블 로직 컨트롤러를 참조하십시오.

PLM

제품 라이프사이클 관리를 참조하십시오.

정책

권한을 정의 (ID 기반 정책 참조) 하거나, 액세스 조건을 지정 (리소스 기반 정책 참조) 하거나, 조직 내 모든 계정에 대한 최대 권한을 정의 AWS Organizations (서비스 제어 정책 참조) 할 수 있는 개체입니다.

다국어 지속성

데이터 액세스 패턴 및 기타 요구 사항을 기반으로 독립적으로 마이크로서비스의 데이터 스토리지 기술 선택. 마이크로서비스가 동일한 데이터 스토리지 기술을 사용하는 경우 구현 문제가 발생하거나 성능이 저하될 수 있습니다. 요구 사항에 가장 적합한 데이터 스토어를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다. 자세한 내용은 마이크로서비스에서 데이터 지속성 활성화를 참조하십시오.

포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 마이그레이션 준비 상태 평가를 참조하십시오.

조건자

일반적으로 조항에 있는 `true` `false` OR를 반환하는 쿼리 조건입니다. `WHERE`

조건부 푸시다운

전송하기 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄어들고 쿼리 성능이 향상됩니다.

예방적 제어

이벤트 발생을 방지하도록 설계된 보안 제어입니다. 이 제어는 네트워크에 대한 무단 액세스나 원치 않는 변경을 방지하는 데 도움이 되는 1차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 Preventative controls를 참조하십시오.

보안 주체

작업을 수행하고 리소스에 액세스할 수 AWS 있는 엔티티 이 엔티티는 일반적으로 IAM 역할의 루트 사용자 또는 사용자입니다. AWS 계정자세한 내용은 IAM 설명서의 역할 용어 및 개념의 보안 주체를 참조하십시오.

개인 정보 보호 중심 설계

전체 엔지니어링 프로세스에서 개인 정보를 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

프라이빗 호스팅 영역

Amazon Route 53에서 하나 이상의 VPC 내 도메인과 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보가 담긴 컨테이너입니다. 자세한 내용은 Route 53 설명서의 프라이빗 호스팅 영역 작업을 참조하십시오.

사전 예방 제어

규정을 준수하지 않는 리소스의 배포를 방지하도록 설계된 보안 제어입니다. 이러한 컨트롤은 리소스를 프로비저닝하기 전에 리소스를 스캔합니다. 리소스가 컨트롤과 호환되지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 컨트롤 참조 안내서를 참조하고 보안 제어 구현의 사전 제어를 참조하십시오. AWS

제품 라이프사이클 관리 (PLM)

설계, 개발, 출시부터 성장 및 성숙도, 폐기 및 제거에 이르는 전체 라이프사이클에 걸쳐 제품에 대한 데이터 및 프로세스를 관리하는 것입니다.

프로덕션 환경

환경을 참조하십시오.

프로그래머블 로직 컨트롤러 (PLC)

제조 분야에서 기계를 모니터링하고 제조 프로세스를 자동화하는 매우 안정적이고 적응력이 뛰어난 컴퓨터입니다.

가명화

데이터세트의 개인 식별자를 자리 표시자 값으로 바꾸는 프로세스입니다. 가명화는 개인 정보를 보호하는 데 도움이 될 수 있습니다. 가명화된 데이터는 여전히 개인 데이터로 간주됩니다.

게시/구독 (게시/구독)

마이크로서비스 간의 비동기 통신을 통해 확장성과 응답성을 개선할 수 있는 패턴입니다. 예를 들어 마이크로서비스 기반 MES에서 마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로서비스를 추가할 수 있습니다.

Q

쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 일련의 단계 (예: 지침).

쿼리 계획 회귀

데이터베이스 서비스 최적화 프로그램이 데이터베이스 환경을 변경하기 전보다 덜 최적의 계획을 선택하는 경우입니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩 및 데이터베이스 엔진 업데이트의 변경으로 인해 발생할 수 있습니다.

R

RACI 매트릭스

RACI (책임, 책임, 상담, 정보 제공) 를 참조하십시오.

랜섬웨어

결제가 완료될 때까지 컴퓨터 시스템이나 데이터에 대한 액세스를 차단하도록 설계된 악성 소프트웨어입니다.

RASCI 매트릭스

책임, 책임, 상담, 정보 제공 (RACI) 을 참조하십시오.

RCAC

행 및 열 액세스 제어를 참조하십시오.

읽기 전용 복제본

읽기 전용 용도로 사용되는 데이터베이스의 사본입니다. 쿼리를 읽기 전용 복제본으로 라우팅하여 기본 데이터베이스의 로드를 줄일 수 있습니다.

재설계

7 R을 참조하십시오.

Recovery Point Objective(RPO)

마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

Recovery Time Objective(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

리팩터링

7 R을 참조하십시오.

리전

지리적 AWS 영역별 리소스 모음. AWS 리전 각각은 격리되어 있고 서로 독립적이므로 내결함성, 안정성 및 복원력을 제공합니다. 자세한 내용은 사용할 수 있는 AWS 리전 계정 지정을 참조하십시오.

회귀

숫자 값을 예측하는 ML 기법입니다. 예를 들어, '이 집은 얼마에 팔릴까?'라는 문제를 풀기 위해 ML 모델은 선형 회귀 모델을 사용하여 주택에 대해 알려진 사실(예: 면적)을 기반으로 주택의 매매 가격을 예측할 수 있습니다.

리호스팅

7 R을 참조하십시오.

release

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

고쳐 놓다

7 R을 참조하십시오.

리플랫포밍

7 R을 참조하십시오.

환매

7 R을 참조하십시오.

복원력

장애를 견디거나 장애를 복구할 수 있는 애플리케이션의 능력 고가용성 및 재해 복구는 복원력을 계획할 때 일반적으로 고려해야 할 사항입니다. AWS 클라우드자세한 내용은 AWS 클라우드 복원력을 참조하십시오.

리소스 기반 정책

Amazon S3 버킷, 엔드포인트, 암호화 키 등의 리소스에 연결된 정책입니다. 이 유형의 정책은 액세스가 허용된 보안 주체, 지원되는 작업 및 충족해야 하는 기타 조건을 지정합니다.

RACI(Responsible, Accountable, Consulted, Informed) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결

정권자 (A), 업무 수행 조언자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원자를 포함하면 매트릭스를 RASCI 매트릭스라고 하고, 지원자를 제외하면 RACI 매트릭스라고 합니다.

대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 Implementing security controls on AWS의 Responsive controls를 참조하십시오.

retain

7 R을 참조하십시오.

은퇴

7 R을 참조하십시오.

회전

공격자가 자격 증명에 액세스하는 것을 더 어렵게 만들기 위해 암호를 주기적으로 업데이트하는 프로세스입니다.

행 및 열 액세스 제어(RCAC)

액세스 규칙이 정의된 기본적이고 유연한 SQL 표현식을 사용합니다. RCAC는 행 권한과 열 마스크로 구성됩니다.

RPO

복구 지점 목표를 참조하십시오.

RTO

복구 시간 목표를 참조하십시오.

런북

특정 작업을 수행하는 데 필요한 일련의 수동 또는 자동 절차입니다. 일반적으로 오류율이 높은 반복 작업이나 절차를 간소화하기 위해 런북을 만듭니다.

S

SAML 2.0

많은 ID 제공업체 (IdPs) 가 사용하는 개방형 표준입니다. 이 기능을 사용하면 페더레이션 싱글 사인온 (SSO) 이 가능하므로 조직의 모든 사용자를 위해 IAM에서 사용자를 생성하지 않고도 사용자가 AWS API 작업에 AWS Management Console 로그인하거나 API 작업을 호출할 수 있습니다.

SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 SAML 2.0 기반 페더레이션 정보를 참조하십시오.

SCADA

감독 제어 및 데이터 수집을 참조하십시오.

SCP

서비스 제어 정책을 참조하십시오.

secret

에는 AWS Secrets Manager암호화된 형태로 저장하는 비밀번호나 사용자 자격 증명과 같은 기밀 또는 제한된 정보. 비밀 값과 해당 메타데이터로 구성됩니다. 비밀 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 Secrets Manager 시크릿에는 무엇이 들어 있나요? 를 참조하십시오. Secrets Manager 설명서에서 확인할 수 있습니다.

보안 제어

위협 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어에는 예방적, 탐정적, 대응적, 사전 예방적 제어의 네 가지 기본 유형이 있습니다.

보안 강화

공격 표면을 줄여 공격에 대한 저항력을 높이는 프로세스입니다. 더 이상 필요하지 않은 리소스 제거, 최소 권한 부여의 보안 모범 사례 구현, 구성 파일의 불필요한 기능 비활성화 등의 작업이 여기에 포함될 수 있습니다.

보안 정보 및 이벤트 관리(SIEM) 시스템

보안 정보 관리(SIM)와 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협과 보안 침해를 탐지하고 알림을 생성합니다.

보안 대응 자동화

보안 이벤트에 자동으로 대응하거나 보안 이벤트를 해결하도록 설계된 사전 정의되고 프로그래밍된 조치입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 탐지 또는 대응형 보안 제어 역할을 합니다. AWS 자동 응답 조치의 예로는 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치, 자격 증명 교체 등이 있습니다.

서버 측 암호화

수신자에 의한 목적지의 데이터 암호화 AWS 서비스

서비스 제어 정책(SCP)

AWS Organizations에 속한 조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다. SCP는 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대해 제한을 설정하거나 가드레일을 정의합니다. SCP를 허용 목록 또는 거부 목록으로 사용하여 허용하거나 금지할 서비스 또는 작업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 서비스 제어 정책을 참조하십시오.

서비스 엔드포인트

의 진입점 URL입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 AWS 서비스 엔드포인트를 참조하십시오.

서비스 수준에 관한 계약(SLA)

IT 팀이 고객에게 제공하기로 약속한 내용(예: 서비스 가동 시간 및 성능)을 명시한 계약입니다.

서비스 수준 표시기 (SLI)

오류율, 가용성 또는 처리량과 같은 서비스의 성능 측면을 측정하는 것입니다.

서비스 수준 목표 (SLO)

서비스 수준 지표로 측정되는 서비스 상태를 나타내는 대상 지표입니다.

공동 책임 모델

클라우드 보안 및 규정 준수에 AWS 대한 책임을 공유하는 것을 설명하는 모델입니다. AWS 클라우드의 보안을 책임지는 반면, 사용자는 클라우드에서의 보안을 담당합니다. 자세한 내용은 공동 책임 모델을 참조하십시오.

시엠

보안 정보 및 이벤트 관리 시스템을 참조하십시오.

단일 장애 지점 (SPOF)

응용 프로그램의 중요한 단일 구성 요소에서 발생한 오류로 인해 시스템이 중단될 수 있습니다.

SLA

SLA (서비스 수준 계약) 를 참조하십시오.

SLI

서비스 수준 표시기 참조.

SLO

서비스 수준 목표를 참조하십시오.

split-and-seed 모델

현대화 프로젝트를 확장하고 가속화하기 위한 패턴입니다. 새로운 기능과 제품 릴리스가 정의되면 핵심 팀이 분할되어 새로운 제품 팀이 만들어집니다. 이를 통해 조직의 역량과 서비스 규모를 조정하고, 개발자 생산성을 개선하고, 신속한 혁신을 지원할 수 있습니다. 자세한 내용은 의 애플리케이션 현대화를 위한 단계별 접근 방식을 참조하십시오. AWS 클라우드

SPOF

단일 장애 지점 보기.

스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정 데이터를 저장하고 하나 이상의 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 구성 구조입니다. 이 구조는 데이터 웨어하우스에서 사용하거나 비즈니스 인텔리전스 용도로 설계되었습니다.

Strangler Fig 패턴

레거시 시스템을 폐기할 수 있을 때까지 시스템 기능을 점진적으로 다시 작성하고 교체하여 모놀리식 시스템을 현대화하기 위한 접근 방식. 이 패턴은 무화과 덩굴이 나무로 자라 결국 숙주를 압도하고 대체하는 것과 비슷합니다. Martin Fowler가 모놀리식 시스템을 다시 작성할 때 위험을 관리하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET(ASMX) 웹 서비스를 점진적으로 현대화하는 방법을 참조하십시오.

서브넷

VPC의 IP 주소 범위입니다. 서브넷은 단일 가용 영역에 상주해야 합니다.

감독 통제 및 데이터 수집 (SCADA)

제조 시 하드웨어와 소프트웨어를 사용하여 물리적 자산과 생산 작업을 모니터링하는 시스템입니다.

대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

합성 테스트

잠재적 문제를 감지하거나 성능을 모니터링하기 위해 사용자 상호 작용을 시뮬레이션하는 방식으로 시스템을 테스트합니다. Amazon CloudWatch Synthetics를 사용하여 이러한 테스트를 생성할 수 있습니다.

T

tags

리소스 구성을 위한 메타데이터 역할을 하는 키-값 쌍. AWS 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다. 자세한 내용은 AWS 리소스에 태그 지정을 참조하십시오.

대상 변수

지도 ML에서 예측하려는 값으로, 결과 변수라고도 합니다. 예를 들어, 제조 설정에서 대상 변수는 제품 결함일 수 있습니다.

작업 목록

런북을 통해 진행 상황을 추적하는 데 사용되는 도구입니다. 작업 목록에는 런북의 개요와 완료해야 할 일반 작업 목록이 포함되어 있습니다. 각 일반 작업에 대한 예상 소요 시간, 소유자 및 진행 상황이 작업 목록에 포함됩니다.

테스트 환경

환경을 참조하십시오.

훈련

ML 모델이 학습할 수 있는 데이터를 제공하는 것입니다. 훈련 데이터에는 정답이 포함되어야 합니다. 학습 알고리즘은 훈련 데이터에서 대상(예측하려는 답)에 입력 데이터 속성을 매핑하는 패턴을 찾고, 이러한 패턴을 캡처하는 ML 모델을 출력합니다. 그런 다음 ML 모델을 사용하여 대상을 모르는 새 데이터에 대한 예측을 할 수 있습니다.

전송 게이트웨이

VPC와 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 트랜짓 게이트웨이란 무엇입니까? 를 참조하십시오.

트렁크 기반 워크플로

개발자가 기능 브랜치에서 로컬로 기능을 구축하고 테스트한 다음 해당 변경 사항을 기본 브랜치에 병합하는 접근 방식입니다. 이후 기본 브랜치는 개발, 프로덕션 이전 및 프로덕션 환경에 순차적으로 구축됩니다.

신뢰할 수 있는 액세스

조직 내 AWS Organizations 및 해당 계정에서 사용자를 대신하여 작업을 수행하도록 지정한 서비스에 권한 부여 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관

리 작업을 수행합니다. 자세한 내용은 AWS Organizations 설명서의 다른 AWS 서비스와 AWS Organizations 함께 사용을 참조하십시오.

튜닝

ML 모델의 정확도를 높이기 위해 훈련 프로세스의 측면을 여러 변경하는 것입니다. 예를 들어, 레이블링 세트를 생성하고 레이블을 추가한 다음 다양한 설정에서 이러한 단계를 여러 번 반복하여 모델을 최적화하는 방식으로 ML 모델을 훈련할 수 있습니다.

피자 두 판 팀

피자 두 판만 들고 배불리 먹을 수 있는 소규모 DevOps 팀. 피자 두 판 팀 규모는 소프트웨어 개발에 있어 가능한 최상의 공동 작업 기회를 보장합니다.

U

불확실성

예측 ML 모델의 신뢰성을 저해할 수 있는 부정확하거나 불완전하거나 알려지지 않은 정보를 나타내는 개념입니다. 불확실성에는 두 가지 유형이 있습니다. 인식론적 불확실성은 제한적이고 불완전한 데이터에 의해 발생하는 반면, 우연한 불확실성은 데이터에 내재된 노이즈와 무작위성에 의해 발생합니다. 자세한 내용은 Quantifying uncertainty in deep learning systems 가이드를 참조하십시오.

차별화되지 않은 작업

애플리케이션을 만들고 운영하는 데 필요하지만 최종 사용자에게 직접적인 가치를 제공하거나 경쟁 우위를 제공하지 못하는 작업을 헤비 리프팅이라고도 합니다. 차별화되지 않은 작업의 예로는 조달, 유지보수, 용량 계획 등이 있습니다.

상위 환경

환경을 보세요.

V

정리

스토리지를 회수하고 성능을 향상시키기 위해 증분 업데이트 후 정리 작업을 수반하는 데이터베이스 유지 관리 작업입니다.

버전 제어

리포지토리의 소스 코드 변경과 같은 변경 사항을 추적하는 프로세스 및 도구입니다.

VPC 피어링

프라이빗 IP 주소를 사용하여 트래픽을 라우팅할 수 있게 하는 두 VPC 간의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 VPC 피어링이란?을 참조하십시오.

취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

W

웜 캐시

자주 액세스하는 최신 관련 데이터를 포함하는 버퍼 캐시입니다. 버퍼 캐시에서 데이터베이스 인스턴스를 읽을 수 있기 때문에 주 메모리나 디스크에서 읽는 것보다 빠릅니다.

웜 데이터

자주 액세스하지 않는 데이터입니다. 이런 종류의 데이터를 쿼리할 때는 일반적으로 적절히 느린 쿼리가 허용됩니다.

윈도우 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에 대해 계산을 수행하는 SQL 함수입니다. 윈도우 함수는 이동 평균을 계산하거나 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 작업을 처리하는 데 유용합니다.

워크로드

고객 대면 애플리케이션이나 백엔드 프로세스 같이 비즈니스 가치를 창출하는 리소스 및 코드 모음입니다.

워크스트림

마이그레이션 프로젝트에서 특정 작업 세트를 담당하는 직무 그룹입니다. 각 워크스트림은 독립적이지만 프로젝트의 다른 워크스트림을 지원합니다. 예를 들어, 포트폴리오 워크스트림은 애플리케이션 우선순위 지정, 웨이브 계획, 마이그레이션 메타데이터 수집을 담당합니다. 포트폴리오 워크스트림은 이러한 자산을 마이그레이션 워크스트림에 전달하고, 마이그레이션 워크스트림은 서버와 애플리케이션을 마이그레이션합니다.

웜

한 번 쓰고, 많이 읽으세요.

WQF

AWS 워크로드 검증 프레임워크를 참조하십시오.

한 번 작성하고 여러 번 읽기 (WORM)

데이터를 한 번 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 인증된 사용자는 필요한 만큼 데이터를 여러 번 읽을 수 있지만 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 변경할 수 없는 것으로 간주됩니다.

Z

제로데이 익스플로잇

제로데이 취약점을 악용하는 공격 (일반적으로 멀웨어) 입니다.

제로데이 취약성

프로덕션 시스템의 명백한 결함 또는 취약성입니다. 위협 행위자는 이러한 유형의 취약성을 사용하여 시스템을 공격할 수 있습니다. 개발자는 공격의 결과로 취약성을 인지하는 경우가 많습니다.

좀비 애플리케이션

평균 CPU 및 메모리 사용량이 5% 미만인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하는 것이 일반적입니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.