aws

Mehrmandantenfähige SaaS-Autorisierung und API-Zugriffskontrolle: Implementierungsoptionen und Best Practices

AWS Präskriptive Leitlinien

AWS Präskriptive Leitlinien: Mehrmandantenfähige SaaS-Autorisierung und API-Zugriffskontrolle: Implementierungsoptionen und Best Practices

Table of Contents

Mehrinstanzenfähige SaaS-Autorisierung und API-Zugriffskontrolle: Implementierungsoptionen und Best Practices

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

Mai 2024 (Geschichte der Dokumente)

Autorisierung und API-Zugriffskontrolle stellen für viele Softwareanwendungen eine Herausforderung dar, insbesondere für Multi-Tenant-Software-as-a-Service (SaaS) -Anwendungen. Diese Komplexität wird deutlich, wenn man die Vielzahl von Microservice-APIs betrachtet, die gesichert werden müssen, und die große Anzahl von Zugriffsbedingungen, die sich aus unterschiedlichen Mandanten, Benutzermerkmalen und Anwendungsstatus ergeben. Um diese Probleme effektiv zu lösen, muss eine Lösung die Zugriffskontrolle für die vielen APIs durchsetzen, die von Microservices, Backend for Frontend-Schichten (BFF) und anderen Komponenten einer mehrinstanzenfähigen SaaS-Anwendung bereitgestellt werden. Dieser Ansatz muss mit einem Mechanismus einhergehen, der in der Lage ist, komplexe Zugriffsentscheidungen auf der Grundlage vieler Faktoren und Eigenschaften zu treffen.

Traditionell wurden API-Zugriffskontrolle und -autorisierung durch eine benutzerdefinierte Logik im Anwendungscode abgewickelt. Dieser Ansatz war fehleranfällig und nicht sicher, da Entwickler, die Zugriff auf diesen Code hatten, die Autorisierungslogik versehentlich oder bewusst ändern konnten, was zu unbefugtem Zugriff führen konnte. Es war schwierig, die Entscheidungen zu überprüfen, die durch die benutzerdefinierte Logik im Anwendungscode getroffen wurden, da die Prüfer sich mit der benutzerdefinierten Logik befassen mussten, um festzustellen, wie effektiv sie bei der Einhaltung eines bestimmten Standards war. Darüber hinaus war eine API-Zugriffskontrolle im Allgemeinen unnötig, da nicht so viele APIs gesichert werden mussten. Der Paradigmenwechsel beim Anwendungsdesign hin zu Microservices und serviceorientierten Architekturen hat die Anzahl der APIs erhöht, die eine Form der Autorisierung und Zugriffskontrolle verwenden müssen. Darüber hinaus stellt die Notwendigkeit, den mandantenbasierten Zugriff in einer SaaS-Anwendung mit mehreren Mandanten aufrechtzuerhalten, zusätzliche Autorisierungsherausforderungen zur Aufrechterhaltung des Mietverhältnisses dar. Die in diesem Leitfaden beschriebenen bewährten Verfahren bieten mehrere Vorteile:

- Die Autorisierungslogik kann zentralisiert und in einer deklarativen Sprache auf hoher Ebene geschrieben werden, die für keine Programmiersprache spezifisch ist.

- Die Autorisierungslogik wird vom Anwendungscode abstrahiert und kann als wiederholbares Muster auf alle APIs in einer Anwendung angewendet werden.
- Die Abstraktion verhindert, dass Entwickler versehentlich Änderungen an der Autorisierungslogik vornehmen.
- Die Integration in eine SaaS-Anwendung ist konsistent und einfach.
- Die Abstraktion verhindert, dass für jeden API-Endpunkt eine benutzerdefinierte Autorisierungslogik geschrieben werden muss.
- Audits werden vereinfacht, da ein Prüfer den Code nicht mehr überprüfen muss, um Berechtigungen zu ermitteln.
- Der in diesem Leitfaden skizzierte Ansatz unterstützt je nach den Anforderungen einer Organisation die Verwendung mehrerer Zugriffskontrollparadigmen.
- Dieser Ansatz zur Autorisierung und Zugriffskontrolle bietet eine einfache und unkomplizierte Möglichkeit, die Isolierung von Mandantendaten auf API-Ebene in einer SaaS-Anwendung aufrechtzuerhalten.
- Die Best Practices bieten einen konsistenten Ansatz für das Onboarding und Offboarding von Mandanten in Bezug auf die Autorisierung.
- Dieser Ansatz bietet verschiedene Modelle zur Bereitstellung von Autorisierungen (gebündelt oder isoliert), die sowohl Vor- als auch Nachteile haben, wie in diesem Leitfaden beschrieben.

Gezielte Geschäftsergebnisse

Diese präskriptiven Leitlinien beschreiben wiederholbare Entwurfsmuster für Autorisierungs- und API-Zugriffskontrollen, die für mehrinstanzenfähige SaaS-Anwendungen implementiert werden können. Diese Anleitung richtet sich an jedes Team, das Anwendungen mit komplexen Autorisierungsanforderungen oder strengen API-Zugriffskontrollen entwickelt. Die Architektur beschreibt detailliert die Erstellung eines Policy Decision Point (PDP) oder einer Policy Engine und die Integration von Policy Enforcement Points (PEP) in APIs. Zwei spezifische Optionen für die Erstellung eines PDP werden erörtert: die Verwendung von Amazon Verified Permissions mit dem Cedar SDK und die Verwendung des Open Policy Agent (OPA) mit der Rego-Richtliniensprache. In diesem Leitfaden wird auch das Treffen von Zugriffsentscheidungen auf der Grundlage eines ABAC-Modells (attribute-Based Access Control) oder eines rollenbasierten Zugriffskontrollmodells (RBAC) oder einer Kombination aus beiden Modellen erörtert. Wir empfehlen Ihnen, die in diesem Leitfaden beschriebenen Entwurfsmuster und Konzepte zu verwenden, um Ihre Implementierung von Autorisierung und API-Zugriffskontrolle in mehrinstanzenfähigen SaaS-Anwendungen zu unterstützen

und zu standardisieren. Diese Anleitung trägt dazu bei, die folgenden Geschäftsergebnisse zu erzielen:

- Standardisierte API-Autorisierungsarchitektur für mehrinstanzenfähige SaaS-Anwendungen — Diese Architektur unterscheidet zwischen drei Komponenten: dem Policy Administration Point (PAP), an dem Richtlinien gespeichert und verwaltet werden, dem Policy Decision Point (PDP), an dem diese Richtlinien bewertet werden, um eine Autorisierungsentscheidung zu treffen, und dem Policy Enforcement Point (PEP), der diese Entscheidung durchsetzt. Der gehostete Autorisierungsdienst Verified Permissions dient sowohl als PAP als auch als PDP. Alternativ können Sie Ihr PDP selbst erstellen, indem Sie eine Open-Source-Engine wie Cedar oder OPA verwenden.
- Entkopplung der Autorisierungslogik von Anwendungen — Die Autorisierungslogik kann, wenn sie in den Anwendungscode eingebettet oder über einen Ad-hoc-Durchsetzungsmechanismus implementiert wird, versehentlichen oder böswilligen Änderungen unterliegen, die zu unbeabsichtigtem mandantenübergreifendem Datenzugriff oder anderen Sicherheitsverletzungen führen. Um diese Möglichkeiten einzudämmen, können Sie eine PAP wie Verified Permissions verwenden, um Autorisierungsrichtlinien unabhängig vom Anwendungscode zu speichern und die Verwaltung dieser Richtlinien streng zu steuern. Richtlinien können zentral in einer deklarativen Sprache auf hoher Ebene verwaltet werden, was die Verwaltung der Autorisierungslogik wesentlich einfacher macht, als wenn Sie Richtlinien in mehrere Abschnitte des Anwendungscodes einbetten. Dieser Ansatz stellt auch sicher, dass Updates konsistent angewendet werden.
- Flexibler Ansatz für Zugriffskontrollmodelle — Rollenbasierte Zugriffskontrolle (RBAC), attributebasierte Zugriffskontrolle (ABAC) oder eine Kombination aus beiden Modellen sind allesamt gültige Ansätze für die Zugriffskontrolle. Diese Modelle versuchen, die Autorisierungsanforderungen eines Unternehmens mithilfe verschiedener Ansätze zu erfüllen. In diesem Leitfaden werden diese Modelle verglichen und gegenübergestellt, um Ihnen bei der Auswahl eines Modells zu helfen, das für Ihr Unternehmen geeignet ist. In dem Leitfaden wird auch erläutert, wie diese Modelle auf verschiedene Sprachen mit Autorisierungsrichtlinien wie OPA/Rego und Cedar angewendet werden können. Die in diesem Leitfaden erörterten Architekturen ermöglichen die erfolgreiche Einführung eines oder beider Modelle.
- Strikte API-Zugriffskontrolle — Dieses Handbuch bietet eine Methode zur konsistenten und umfassenden Sicherung von APIs in einer Anwendung mit minimalem Aufwand. Dies ist besonders nützlich für serviceorientierte Anwendungsarchitekturen oder Microservice-Anwendungsarchitekturen, die im Allgemeinen eine große Anzahl von APIs verwenden, um die anwendungsinterne Kommunikation zu erleichtern. Eine strenge API-Zugriffskontrolle trägt dazu

bei, die Sicherheit einer Anwendung zu erhöhen und macht sie weniger anfällig für Angriffe oder Ausnutzung.

Isolierung von Mandanten und Autorisierung mehrerer Mandanten

Dieser Leitfaden bezieht sich auf die Konzepte der Mandantenisolierung und der Mehrmandantenautorisierung. Mandantenisolierung bezieht sich auf explizite Mechanismen, die Sie in einem SaaS-System verwenden, um sicherzustellen, dass die Ressourcen jedes Mandanten, auch wenn sie auf einer gemeinsam genutzten Infrastruktur betrieben werden, isoliert sind. Bei der Mehrmandantenautorisierung werden eingehende Aktionen autorisiert und verhindert, dass diese auf dem falschen Mandanten implementiert werden. Ein hypothetischer Benutzer könnte authentifiziert und autorisiert werden und trotzdem auf die Ressourcen eines anderen Mandanten zugreifen. Durch Authentifizierung und Autorisierung wird dieser Zugriff nicht blockiert. Um dieses Ziel zu erreichen, müssen Sie die Mandantenisolierung implementieren. Eine ausführlichere Erläuterung der Unterschiede zwischen diesen beiden Konzepten finden Sie im Abschnitt Mandantenisolierung des Whitepapers SaaS Architecture Fundamentals.

Arten der Zugriffskontrolle

Für die Implementierung der Zugriffskontrolle können Sie zwei allgemein definierte Modelle verwenden: die rollenbasierte Zugriffskontrolle (RBAC) und die attributebasierte Zugriffskontrolle (ABAC). Jedes Modell hat Vor- und Nachteile, auf die in diesem Abschnitt kurz eingegangen wird. Welches Modell Sie verwenden sollten, hängt von Ihrem spezifischen Anwendungsfall ab. Die in diesem Handbuch beschriebene Architektur unterstützt beide Modelle.

RBAC

Die rollenbasierte Zugriffskontrolle (RBAC) bestimmt den Zugriff auf Ressourcen auf der Grundlage einer Rolle, die normalerweise der Geschäftslogik entspricht. Der Rolle werden je nach Bedarf Berechtigungen zugeordnet. Beispielsweise würde eine Marketingrolle einen Benutzer autorisieren, Marketingaktivitäten innerhalb eines eingeschränkten Systems durchzuführen. Dies ist ein relativ einfach zu implementierendes Zugriffskontrollmodell, da es sich gut an der leicht erkennbaren Geschäftslogik orientiert.

Das RBAC-Modell ist weniger effektiv, wenn:

- Sie haben eindeutige Benutzer, deren Zuständigkeiten mehrere Rollen umfassen.
- Sie haben eine komplexe Geschäftslogik, die es schwierig macht, Rollen zu definieren.
- Die Skalierung auf eine große Größe erfordert eine ständige Verwaltung und Zuordnung von Berechtigungen zu neuen und bestehenden Rollen.
- Autorisierungen basieren auf dynamischen Parametern.

ABAC

Die attributebasierte Zugriffskontrolle (ABAC) bestimmt den Zugriff auf Ressourcen anhand von Attributen. Attribute können einem Benutzer, einer Ressource, einer Umgebung oder sogar einem Anwendungsstatus zugeordnet werden. Ihre Richtlinien oder Regeln verweisen auf Attribute und können mithilfe grundlegender boolescher Logik bestimmen, ob ein Benutzer eine Aktion ausführen darf. Hier ist ein einfaches Beispiel für Berechtigungen:

Im Zahlungssystem dürfen alle Benutzer der Finanzabteilung `/payments` während der Geschäftszeiten Zahlungen am API-Endpunkt abwickeln.

Die Mitgliedschaft in der Finanzabteilung ist ein Benutzerattribut, das den Zugriff auf bestimmt`/payments`. Dem `/payments` API-Endpunkt ist auch ein Ressourcenattribut zugeordnet, das den Zugriff nur während der Geschäftszeiten ermöglicht. In ABAC wird durch eine Richtlinie festgelegt, die die Mitgliedschaft in der Finanzabteilung als Benutzerattribut und die Zeit als Ressourcenattribut von beinhaltet, ob ein Benutzer eine Zahlung verarbeiten kann oder nicht. `/payments`

Das ABAC-Modell ist sehr flexibel und ermöglicht dynamische, kontextbezogene und detaillierte Autorisierungsentscheidungen. Das ABAC-Modell ist jedoch zunächst schwierig zu implementieren. Die Definition von Regeln und Richtlinien sowie die Aufzählung von Attributen für alle relevanten Zugriffsvektoren erfordern erhebliche Vorabinvestitionen für die Implementierung.

Hybrider RBAC-ABAC-Ansatz

Die Kombination von RBAC und ABAC kann einige der Vorteile beider Modelle bieten. Da RBAC so eng an der Geschäftslogik ausgerichtet ist, ist es einfacher zu implementieren als ABAC. Um eine zusätzliche Ebene der Granularität bei Autorisierungsentscheidungen zu erreichen, können Sie ABAC mit RBAC kombinieren. Dieser hybride Ansatz bestimmt den Zugriff, indem er die Rolle eines Benutzers (und die ihm zugewiesenen Berechtigungen) mit zusätzlichen Attributen kombiniert, um Zugriffsentscheidungen zu treffen. Die Verwendung beider Modelle ermöglicht eine einfache Verwaltung und Zuweisung von Berechtigungen und ermöglicht gleichzeitig mehr Flexibilität und Granularität bei Autorisierungsentscheidungen.

Vergleich der Modelle für die Zugriffskontrolle

In der folgenden Tabelle werden die drei zuvor erörterten Zugriffskontrollmodelle verglichen. Dieser Vergleich soll informativ und umfassend sein. Die Verwendung eines Zugriffsmodells in einer bestimmten Situation korreliert möglicherweise nicht unbedingt mit den Vergleichen in dieser Tabelle.

Faktor	RBAC	ABAC	Hybrid
Flexibilität	Medium	Hoch	Hoch
Schlichtheit	Hoch	Niedrig	Mittelschwer
Granularität	Niedrig	Hoch	Mittelschwer

Dynamische Entscheidungen und Regeln	Nein	Ja	Ja
Kontextsensitiv	Nein	Ja	Einigermaßen
Aufwand für die Umsetzung	Niedrig	Hoch	Mittelschwer

Implementierung eines PDP

Der Policy Decision Point (PDP) kann als Policy- oder Rules-Engine bezeichnet werden. Diese Komponente ist dafür verantwortlich, Richtlinien oder Regeln anzuwenden und eine Entscheidung darüber abzugeben, ob ein bestimmter Zugriff zulässig ist. Ein PDP kann mit Modellen der rollenbasierten Zugriffskontrolle (RBAC) und der attributebasierten Zugriffskontrolle (ABAC) funktionieren. Für ABAC ist jedoch ein PDP erforderlich. Ein PDP ermöglicht die Auslagerung der Autorisierungslogik im Anwendungscode auf ein separates System. Dies kann den Anwendungscode vereinfachen. Es bietet auch eine easy-to-use wiederholbare Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder jede andere Anwendungskomponente.

In den folgenden Abschnitten werden drei Methoden zur Implementierung eines PDP beschrieben. Dies ist jedoch keine vollständige Liste.

Methoden zur PDP-Implementierung:

- Implementierung eines PDP mithilfe von Amazon Verified Permissions
- Implementierung eines PDP mithilfe von OPA
- Verwenden einer benutzerdefinierten Policy-Engine

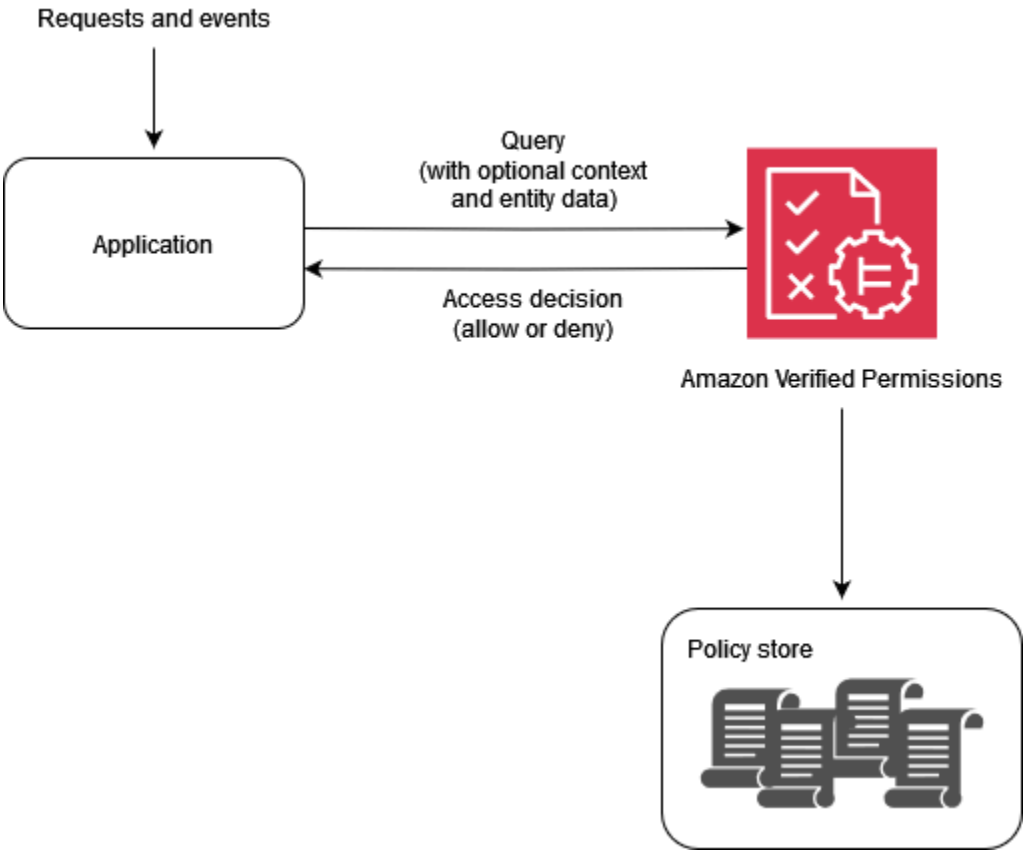
Implementierung eines PDP mithilfe von Amazon Verified Permissions

Amazon Verified Permissions ist ein skalierbarer, detaillierter Service zur Verwaltung und Autorisierung von Berechtigungen, mit dem Sie einen Policy Decision Point (PDP) implementieren können. Als Policy-Engine kann er Ihrer Anwendung dabei helfen, Benutzeraktionen in Echtzeit zu überprüfen und übermäßig privilegierte oder ungültige Berechtigungen hervorzuheben. Es hilft Ihren Entwicklern, sicherere Anwendungen schneller zu entwickeln, indem es die Autorisierung externalisiert und die Richtlinienverwaltung und -verwaltung zentralisiert. Durch die Trennung von Autorisierungslogik und Anwendungslogik unterstützt Verified Permissions die Entkopplung von Richtlinien.

Durch die Verwendung verifizierter Berechtigungen zur Implementierung eines PDP und die Implementierung der geringsten Rechte und der kontinuierlichen Überprüfung innerhalb von Anwendungen können Entwickler ihren Anwendungszugriff an den Zero-Trust-Prinzipien

ausrichten. Darüber hinaus können Sicherheits- und Auditteams besser analysieren und prüfen, wer Zugriff auf welche Ressourcen innerhalb einer Anwendung hat. Verified Permissions verwendet Cedar, eine speziell entwickelte und sicherheitsorientierte Open-Source-Richtliniensprache, um richtlinienbasierte Zugriffskontrollen auf der Grundlage von rollenbasierter Zugriffskontrolle (RBAC) und attributebasierter Zugriffskontrolle (ABAC) für eine detailliertere, kontextsensitive Zugriffskontrolle zu definieren.

Verified Permissions bietet einige nützliche Funktionen für SaaS-Anwendungen, z. B. die Möglichkeit, die Mehrmandantenautorisierung mithilfe mehrerer Identitätsanbieter wie Amazon Cognito, Google und Facebook zu aktivieren. Eine weitere Funktion für verifizierte Berechtigungen, die besonders für SaaS-Anwendungen hilfreich ist, ist die Unterstützung benutzerdefinierter Rollen pro Mandant. Wenn Sie ein CRM-System (Customer Relationship Management) entwerfen, kann ein Mandant die Granularität des Zugriffs nach Verkaufschancen auf der Grundlage eines bestimmten Kriterienkatalogs definieren. Ein anderer Mandant könnte eine andere Definition haben. Die zugrunde liegenden Berechtigungssysteme in Verified Permissions können diese Variationen unterstützen, was es zu einem ausgezeichneten Kandidaten für SaaS-Anwendungsfälle macht. Verified Permissions unterstützt auch die Möglichkeit, Richtlinien zu schreiben, die für alle Mandanten gelten, sodass es als SaaS-Anbieter einfach ist, Guardrail-Richtlinien anzuwenden, um unbefugten Zugriff zu verhindern.



Warum verifizierte Berechtigungen verwenden?

Verwenden Sie Verified Permissions mit einem Identitätsanbieter wie Amazon Cognito für eine dynamischere, richtlinienbasierte Zugriffsverwaltungslösung für Ihre Anwendungen. Sie können Anwendungen entwickeln, mit denen Benutzer Informationen austauschen und zusammenarbeiten können, während gleichzeitig die Sicherheit, Vertraulichkeit und Vertraulichkeit ihrer Daten gewahrt bleiben. Verified Permissions trägt zur Senkung der Betriebskosten bei, indem es Ihnen ein detailliertes Autorisierungssystem zur Verfügung stellt, mit dem Sie den Zugriff auf der Grundlage der Rollen und Attribute Ihrer Identitäten und Ressourcen durchsetzen können. Sie können Ihr Richtlinienmodell definieren, Richtlinien an einem zentralen Ort erstellen und speichern und Zugriffsanfragen innerhalb von Millisekunden auswerten.

In Verified Permissions können Sie Berechtigungen mithilfe einer einfachen, für Menschen lesbaren Deklarationssprache namens Cedar ausdrücken. In Cedar geschriebene Richtlinien können von Teams gemeinsam genutzt werden, unabhängig von der Programmiersprache, die von den jeweiligen Teamanwendungen verwendet wird.

Was ist zu beachten, wenn Sie verifizierte Berechtigungen verwenden

In Verified Permissions können Sie Richtlinien erstellen und diese im Rahmen der Bereitstellung automatisieren. Sie können Richtlinien auch zur Laufzeit als Teil der Anwendungslogik erstellen. Als bewährte Methode sollten Sie eine CI/CD-Pipeline (Continuous Integration and Continuous Deployment) verwenden, um Richtlinienversionen zu verwalten, zu ändern und nachzuverfolgen, wenn Sie Richtlinien im Rahmen des Onboardings und der Bereitstellung von Mandanten erstellen. Alternativ kann eine Anwendung Richtlinienversionen verwalten, ändern und nachverfolgen. Die Anwendungslogik bietet diese Funktionalität jedoch nicht von Natur aus. Um diese Funktionen in Ihrer Anwendung zu unterstützen, müssen Sie Ihre Anwendung explizit so entwerfen, dass sie diese Funktionalität implementiert.

Wenn für eine Autorisierungsentscheidung externe Daten aus anderen Quellen bereitgestellt werden müssen, müssen diese Daten abgerufen und im Rahmen der Autorisierungsanfrage an Verified Permissions weitergeleitet werden. Zusätzlicher Kontext, Entitäten und Attribute werden mit diesem Service standardmäßig nicht abgerufen.

Überblick über Cedar

Cedar ist eine flexible, erweiterbare und skalierbare richtlinienbasierte Sprache für die Zugriffskontrolle, mit der Entwickler Anwendungsberechtigungen in Form von Richtlinien ausdrücken können. Administratoren und Entwickler können Richtlinien definieren, die es Benutzern ermöglichen oder verbieten, auf Anwendungsressourcen zuzugreifen. Einer einzelnen Ressource können mehrere Richtlinien zugeordnet werden. Wenn ein Benutzer Ihrer Anwendung versucht, eine Aktion für eine Ressource auszuführen, fordert Ihre Anwendung eine Autorisierung von der Cedar Policy Engine an. Cedar bewertet die geltenden Richtlinien und gibt eine `ALLOW` `DENY` Oder-Entscheidung zurück. Cedar unterstützt Autorisierungsregeln für alle Arten von Prinzipalen und Ressourcen, ermöglicht eine rollenbasierte Zugriffskontrolle (RBAC) und eine attributebasierte Zugriffskontrolle (ABAC) und unterstützt Analysen mithilfe automatisierter Argumentationstools.

Mit Cedar können Sie Ihre Geschäftslogik von der Autorisierungslogik trennen. Wenn Sie Anfragen anhand des Codes Ihrer Anwendung stellen, rufen Sie die Autorisierungs-Engine von Cedar auf, um festzustellen, ob die Anfrage autorisiert ist. Wenn sie autorisiert ist (die Entscheidung lautet`ALLOW`), kann Ihre Anwendung den angeforderten Vorgang ausführen. Wenn sie nicht autorisiert ist (die Entscheidung ist es`DENY`), kann Ihre Anwendung eine Fehlermeldung zurückgeben. Zu den Hauptmerkmalen von Cedar gehören:

- Ausdruckskraft — Cedar wurde speziell für die Unterstützung von Anwendungsfällen im Bereich Autorisierung entwickelt und unter Berücksichtigung der Lesbarkeit durch den Menschen entwickelt.

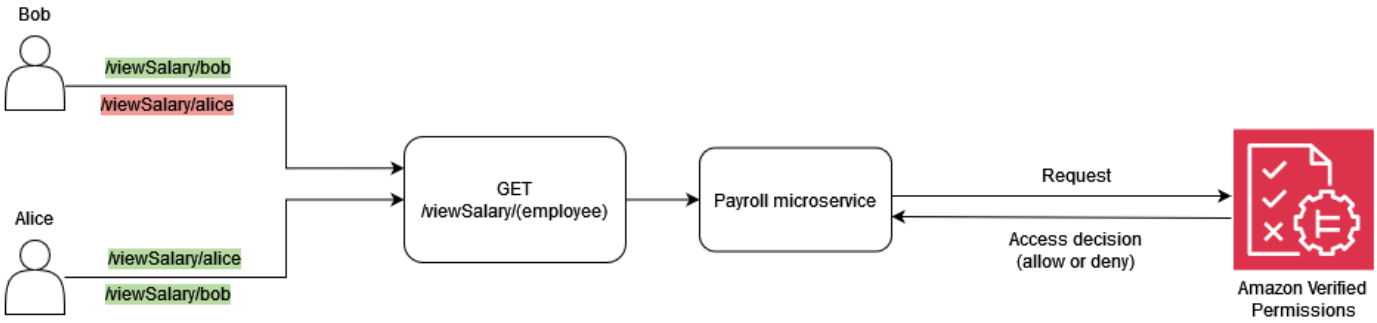
- Leistung — Cedar unterstützt Indexierungsrichtlinien für einen schnellen Abruf und bietet eine schnelle und skalierbare Echtzeitauswertung mit begrenzter Latenz.
- Analyse — Cedar unterstützt Analysetools, mit denen Sie Ihre Richtlinien optimieren und Ihr Sicherheitsmodell überprüfen können.

Weitere Informationen finden Sie auf der Cedar-Website.

Beispiel 1: Einfaches ABAC mit verifizierten Berechtigungen und Cedar

In diesem Beispielszenario wird Amazon Verified Permissions verwendet, um zu bestimmen, welche Benutzer auf Informationen in einem fiktiven Payroll-Microservice zugreifen dürfen. Dieser Abschnitt enthält Codefragmente von Cedar, um zu demonstrieren, wie Sie Cedar verwenden können, um Entscheidungen zur Zugriffskontrolle zu treffen. Diese Beispiele sind nicht dazu gedacht, einen vollständigen Überblick über die Funktionen von Cedar und Verified Permissions zu geben. Einen ausführlicheren Überblick über Cedar finden Sie in der Cedar-Dokumentation.

In der folgenden Abbildung möchten wir zwei allgemeine Geschäftsregeln durchsetzen, die mit der `viewSalary` `GET` Methode verknüpft sind: Mitarbeiter können ihr eigenes Gehalt einsehen und Mitarbeiter können das Gehalt aller Personen einsehen, die ihnen unterstellt sind. Sie können diese Geschäftsregeln mithilfe von Richtlinien für verifizierte Berechtigungen durchsetzen.



Mitarbeiter können ihr eigenes Gehalt einsehen.

In Cedar ist das grundlegende Konstrukt eine Entität, die einen Prinzipal, eine Aktion oder eine Ressource darstellt. Um eine Autorisierungsanfrage zu stellen und eine Evaluierung mit einer Richtlinie für verifizierte Berechtigungen zu starten, müssen Sie einen Prinzipal, eine Aktion, eine Ressource und eine Liste von Entitäten angeben.

- Der Principal (`principal`) ist der angemeldete Benutzer oder die angemeldete Rolle.
- Die Aktion (`action`) ist die Operation, die durch die Anfrage ausgewertet wird.

- Die Ressource (`resource`) ist die Komponente, auf die die Aktion zugreift.
- Die Liste der Entitäten (`entityList`) enthält alle erforderlichen Entitäten, die zur Auswertung der Anfrage benötigt werden.

Um die Geschäftsregel „Mitarbeiter können ihr eigenes Gehalt einsehen“ zu erfüllen, können Sie eine Richtlinie für verifizierte Berechtigungen wie die folgende angeben.

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

Mit dieser Richtlinie `Action` wird geprüft, `ALLOW` ob das Objekt `viewSalary` und die Ressource in der Anfrage über ein Attribut verfügen, das dem Prinzipal entspricht. Wenn Bob beispielsweise der angemeldete Benutzer ist, der den Gehaltsbericht angefordert hat, und auch der Besitzer des Gehaltsberichts ist, wird die Richtlinie wie folgt ausgewertet. `ALLOW`

Die folgende Autorisierungsanfrage wird an Verified Permissions gesendet, um anhand der Beispielrichtlinie bewertet zu werden. In diesem Beispiel ist Bob der angemeldete Benutzer, der die `viewSalary` Anfrage stellt. Daher ist Bob der Principal des Entitätstyps`Employee`. Die Aktion, die Bob auszuführen versucht, ist `viewSalary,` und die Ressource, die angezeigt `viewSalary` wird, `Salary-Bob` entspricht dem Typ`Salary`. Um zu beurteilen, ob Bob die `Salary-Bob` Ressource anzeigen kann, müssen Sie eine Entitätsstruktur angeben, die den Typ `Employee` mit dem Wert `Bob` (Principal) mit dem Eigentümerattribut der Ressource verknüpft, die diesen Typ hat`Salary`. Sie geben diese Struktur in einer an`entityList`, wobei die zugehörigen Attribute einen Eigentümer `Salary` enthalten, der einen Eigentümer angibt, der den Typ `Employee` und den Wert enthält`Bob`. `entityIdentifier` Verified Permissions vergleicht die in der Autorisierungsanfrage `principal` angegebenen Werte mit dem `owner` Attribut, das der `Salary` Ressource zugeordnet ist, um eine Entscheidung zu treffen.

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

Die Autorisierungsanfrage an Verified Permissions gibt Folgendes als Ausgabe zurück, wobei das Attribut entweder `ALLOW` oder `decision` ist`DENY`.

```
{
    "determiningPolicies":
        [
```

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

Da Bob in diesem Fall versucht hat, sein eigenes Gehalt einzusehen, wird die an Verified Permissions gesendete Autorisierungsanfrage als ausgewertet. `ALLOW` Unser Ziel war es jedoch, Verified Permissions zu verwenden, um zwei Geschäftsregeln durchzusetzen. Die Geschäftsregel, die Folgendes besagt, sollte ebenfalls zutreffen:

Mitarbeiter können das Gehalt aller Mitarbeiter einsehen, die ihnen unterstellt sind.

Um diese Geschäftsregel zu erfüllen, können Sie eine weitere Richtlinie angeben. Mit der folgenden Richtlinie wird geprüft, `ALLOW` ob die Aktion `viewSalary` und die Ressource in der Anforderung ein Attribut hato`wner.manager`, das dem Prinzipal entspricht. Wenn Alice beispielsweise die angemeldete Benutzerin ist, die den Gehaltsbericht angefordert hat, und Alice die Managerin des Berichtsbesitzers ist, wird die Richtlinie wie folgt ausgewertet. `ALLOW`

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

Die folgende Autorisierungsanfrage wird an Verified Permissions gesendet, um anhand der Beispielrichtlinie bewertet zu werden. In diesem Beispiel ist Alice die angemeldete Benutzerin, die die `viewSalary` Anfrage stellt. Daher ist Alice die Principal und die Entität ist vom Typ`Employee`. Die Aktion, die Alice auszuführen versucht`viewSalary`, ist, und die Ressource, die angezeigt `viewSalary` wird, ist vom Typ `Salary` mit dem Wert von`Salary-Bob`. Um zu beurteilen, ob Alice die `Salary-Bob` Ressource anzeigen kann, müssen Sie eine Entitätsstruktur angeben, die den Typ `Employee` mit einem Wert von `Alice` mit dem `manager` Attribut verknüpft, das dann dem `owner` Attribut des Typs `Salary` mit dem Wert von zugeordnet werden muss`Salary-Bob`. Sie geben diese Struktur in einer an`entityList`, wobei die mit verknüpften Attributen einen Besitzer

`Salary` enthalten, der einen Eigentümer angibt`entityIdentifier`, der den Typ `Employee` und den Wert enthältBob. Verified Permissions überprüft zunächst das `owner` Attribut, woraufhin der Typ `Employee` und der Wert Bob ausgewertet werden. Anschließend bewertet Verified Permissions das zugeordnete `manager` Attribut `Employee` und vergleicht es mit dem angegebenen Prinzipal, um eine Autorisierungsentscheidung zu treffen. In diesem Fall liegt die Entscheidung `ALLOW` daran, dass die `resource.owner.manager` Attribute `principal` und identisch sind.

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

Bisher haben wir in diesem Beispiel die beiden mit der `viewSalary` Methode verknüpften Geschäftsregeln — Mitarbeiter können ihr eigenes Gehalt einsehen und Mitarbeiter können das Gehalt aller Mitarbeiter einsehen, die ihnen unterstellt sind — als Richtlinien für verifizierte Berechtigungen bereitgestellt, mit denen die Bedingungen jeder Geschäftsregel unabhängig voneinander erfüllt werden können. Sie können auch eine einzige Richtlinie für verifizierte Berechtigungen verwenden, um die Bedingungen beider Geschäftsregeln zu erfüllen:

Mitarbeiter können ihr eigenes Gehalt und das Gehalt aller Mitarbeiter einsehen, die ihnen unterstellt sind.

Wenn Sie die vorherige Autorisierungsanfrage verwenden, prüft die folgende Richtlinie, `ALLOW` ob die Aktion `viewSalary` und die Ressource in der Anforderung ein Attribut hat, `owner.manager` das gleich dem ist`principal`, oder ein Attribut, `owner` das dem `principal` entspricht.

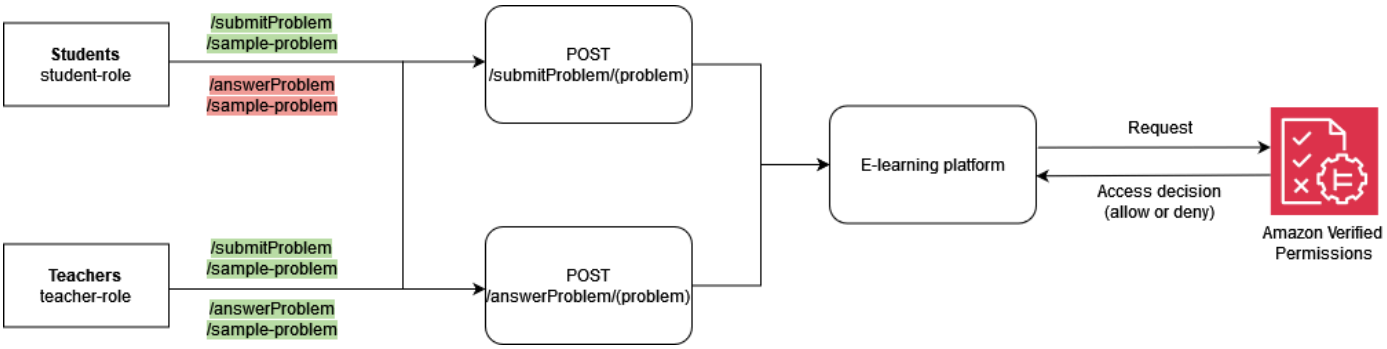
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

Wenn Alice beispielsweise die angemeldete Benutzerin ist, die den Gehaltsbericht anfordert, und wenn Alice entweder die Managerin des Eigentümers oder die Eigentümerin des Berichts ist, dann wird die Richtlinie wie folgt ausgewertet. `ALLOW`

Weitere Informationen zur Verwendung logischer Operatoren mit Cedar-Richtlinien finden Sie in der Cedar-Dokumentation.

Beispiel 2: Basic RBAC mit verifizierten Berechtigungen und Cedar

In diesem Beispiel werden Verified Permissions und Cedar verwendet, um grundlegende RBAC-Funktionen zu demonstrieren. Wie bereits erwähnt, ist das grundlegende Konstrukt von Cedar eine Entität. Entwickler definieren ihre eigenen Entitäten und können optional Beziehungen zwischen Entitäten erstellen. Das folgende Beispiel umfasst drei Arten von Entitäten: `UsersRoles`, und`Problems`. `Students`und `Teachers` können als Entitäten des Typs betrachtet werden, `Role`, und jede `User` kann mit Null oder einer der folgenden Entitäten verknüpft werden`Roles`.



In Cedar werden diese Beziehungen dadurch ausgedrückt, dass das Objekt mit dem `User` Bob als `Role` `Student` übergeordnetem Element verknüpft wird. Diese Zuordnung gruppiert logisch alle

studentischen Benutzer in einer Gruppe. Weitere Informationen zur Gruppierung in Cedar finden Sie in der Cedar-Dokumentation.

Die folgende Richtlinie gilt als Grundlage für die Entscheidung `ALLOW` für die Aktion `submitProblem`, für alle Prinzipale, die mit der logischen Gruppe `Students` des Typs verknüpft sind. `Role`

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

Die folgende Richtlinie bewertet die Entscheidung `ALLOW` für die Aktion `submitProblem` oder für alle Prinzipale`answerProblem`, die mit der logischen Gruppe `Teachers` des Typs verknüpft sind. `Role`

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

Um Anfragen mit diesen Richtlinien auswerten zu können, muss das Evaluierungsmodul wissen, ob der in der Autorisierungsanfrage angegebene Hauptbenutzer Mitglied der entsprechenden Gruppe ist. Daher muss die Anwendung im Rahmen der Autorisierungsanfrage die entsprechenden Informationen zur Gruppenmitgliedschaft an die Evaluierungs-Engine weitergeben. Dies erfolgt über die `entities` Eigenschaft, die es Ihnen ermöglicht, der Cedar Evaluation Engine Attribut- und Gruppenmitgliedschaftsdaten für den Prinzipal und die Ressource zur Verfügung zu stellen, die am Autorisierungsaufruf beteiligt waren. Im folgenden Code wird die Gruppenzugehörigkeit dadurch gekennzeichnet, dass ein Elternteil angerufen hat`Role::"Students"`. `User::"Bob"`

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

In diesem Beispiel ist Bob der angemeldete Benutzer, der die `answerProblem` Anfrage stellt. Daher ist Bob der Principal und die Entität ist vom Typ`User`. Die Aktion, die Bob auszuführen versucht, ist`answerProblem`. Um zu beurteilen, ob Bob die `answerProblem` Aktion ausführen kann, müssen Sie eine Entitätsstruktur angeben, die die Entität `User` mit einem Wert von verknüpft `Bob` und ihr Gruppenmitgliedschaft zuweist, indem sie eine übergeordnete Entität als `Role::"Students"`

auflistet. Da Entitäten in der Benutzergruppe nur die Aktion ausführen `Role::"Students"` dürfen`submitProblem`, wird diese Autorisierungsanfrage wie folgt ausgewertet. `DENY`

Wenn andererseits der Typ`User`, der den Wert hat `Alice` und Teil der Gruppe ist, `Role::"Teachers"` versucht, die `answerProblem` Aktion auszuführen, wird die Autorisierungsanfrage als ausgewertet`ALLOW`, da die Richtlinie vorschreibt, dass Prinzipale in der Gruppe die Aktion `answerProblem` für alle Ressourcen ausführen `Role::"Teachers"` dürfen. Der folgende Code zeigt diese Art von Autorisierungsanfrage, die zu ausgewertet wird. `ALLOW`

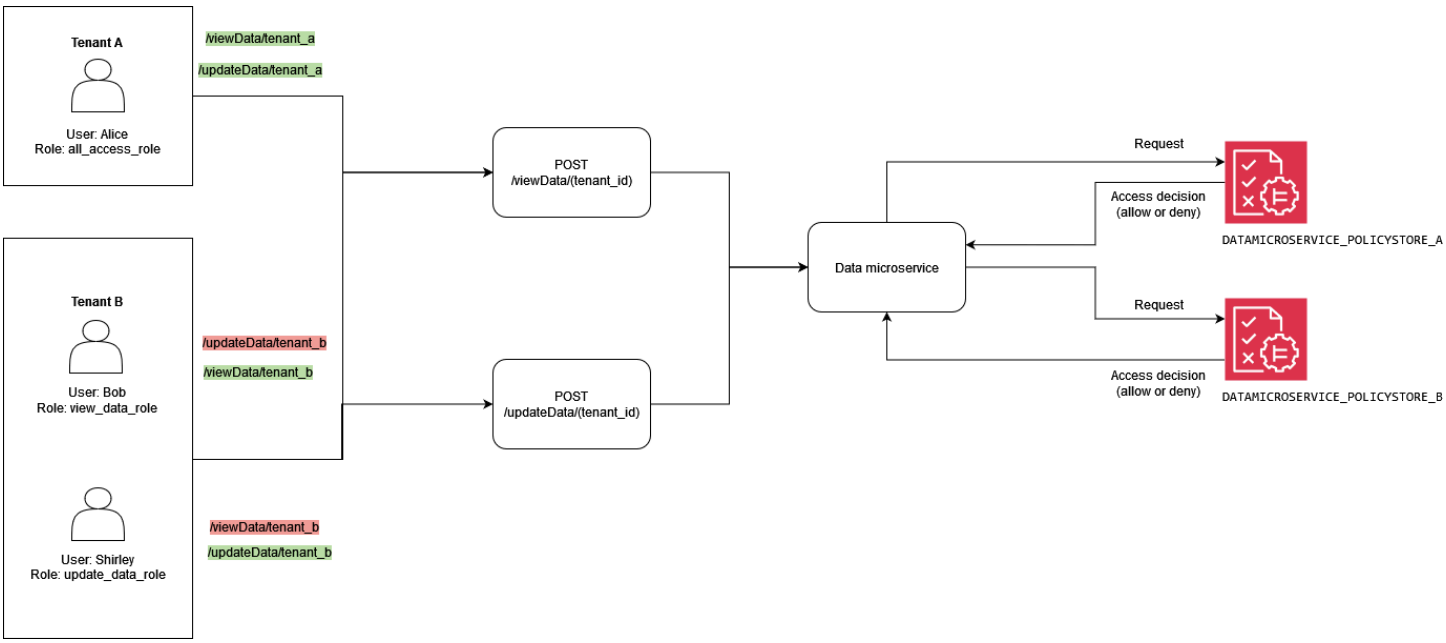
```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId":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

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

Beispiel 3: Zugriffskontrolle für mehrere Mandanten mit RBAC

Um das vorherige RBAC-Beispiel näher zu erläutern, können Sie Ihre Anforderungen um SaaS-Mehrmandantenfähigkeit erweitern, was eine häufige Anforderung für SaaS-Anbieter ist. Bei Lösungen mit mehreren Mandanten wird der Ressourcenzugriff immer im Namen eines bestimmten Mandanten bereitgestellt. Das heißt, Benutzer von Mandant A können die Daten von Mandant B nicht einsehen, selbst wenn diese Daten logisch oder physisch in einem System zusammengefasst sind. Das folgende Beispiel zeigt, wie Sie die Mandantenisolierung mithilfe mehrerer Richtlinienspeicher für verifizierte Berechtigungen implementieren können und wie Sie Benutzerrollen verwenden können, um Berechtigungen innerhalb des Mandanten zu definieren.

Die Verwendung des Entwurfsmusters „Pro Tenant Policy Store“ ist eine bewährte Methode, um die Mandantenisolierung aufrechtzuerhalten und gleichzeitig die Zugriffskontrolle mit verifizierten Berechtigungen zu implementieren. In diesem Szenario werden Benutzeranfragen von Mandant A und Mandant B anhand separater Richtlinienspeicher bzw. `DATAMICROSERVICE_POLICYSTORE_A` `DATAMICROSERVICE_POLICYSTORE_B` Weitere Informationen zu Designüberlegungen für verifizierte Berechtigungen für mehrinstanzenfähige SaaS-Anwendungen finden Sie im Abschnitt Überlegungen zum Entwurf verifizierter Berechtigungen für mehrere Mandanten.



Die folgende Richtlinie befindet sich im Richtlinienspeicher. `DATAMICROSERVICE_POLICYSTORE_A` Es wird überprüft, ob der Prinzipal Teil der Typgruppe `allAccessRole` ist. `Role` In diesem Fall darf der Principal die `updateData` Aktionen `viewData` und für alle Ressourcen ausführen, die Mandant A zugeordnet sind.

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

Die folgenden Richtlinien befinden sich im `DATAMICROSERVICE_POLICYSTORE_B` Richtlinienspeicher. Die erste Richtlinie überprüft, ob der Prinzipal Teil der `updateDataRole` Typgruppe ist. `Role` Unter der Annahme, dass dies der Fall ist, erteilt sie den Prinzipalen die Erlaubnis, die `updateData` Aktion für Ressourcen auszuführen, die Mandant B zugeordnet sind.

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

Diese zweite Richtlinie schreibt vor, dass Prinzipale, die Teil der `viewDataRole` Typgruppe sind, die `viewData` Aktion für Ressourcen ausführen `Role` dürfen, die Mandant B zugeordnet sind.

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

Die Autorisierungsanfrage von Mandant A muss an den `DATAMICROSERVICE_POLICYSTORE_A` Richtlinienspeicher gesendet und anhand der Richtlinien überprüft werden, die zu diesem Speicher gehören. In diesem Fall wird sie anhand der ersten Richtlinie verifiziert, die zuvor im Rahmen dieses Beispiels erörtert wurde. In dieser Autorisierungsanfrage fordert der Prinzipal des Typs `User` mit dem `Alice` Wert von die Ausführung der `viewData` Aktion an. Der Principal gehört zur Typgruppe `allAccessRoleRole`. Alice versucht, die `viewData` Aktion auf der `SampleData` Ressource auszuführen. Da Alice die `allAccessRole` Rolle innehat, führt diese Bewertung zu einer `ALLOW` Entscheidung.

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

Wenn Sie sich stattdessen eine Anfrage von Mandant B von ansehen`User Bob`, werden Sie etwa die folgende Autorisierungsanfrage sehen. Die Anfrage wird an den `DATAMICROSERVICE_POLICYSTORE_B` Richtlinienspeicher gesendet, da sie von Mandant B stammt. In dieser Anfrage `Bob` möchte der Principal die Aktion `updateData` für die Ressource `SampleData` ausführen. `Bob`Ist jedoch nicht Teil einer Gruppe, die Zugriff auf die Aktion `updateData` für diese Ressource hat. Daher führt die Anfrage zu einer `DENY` Entscheidung.

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

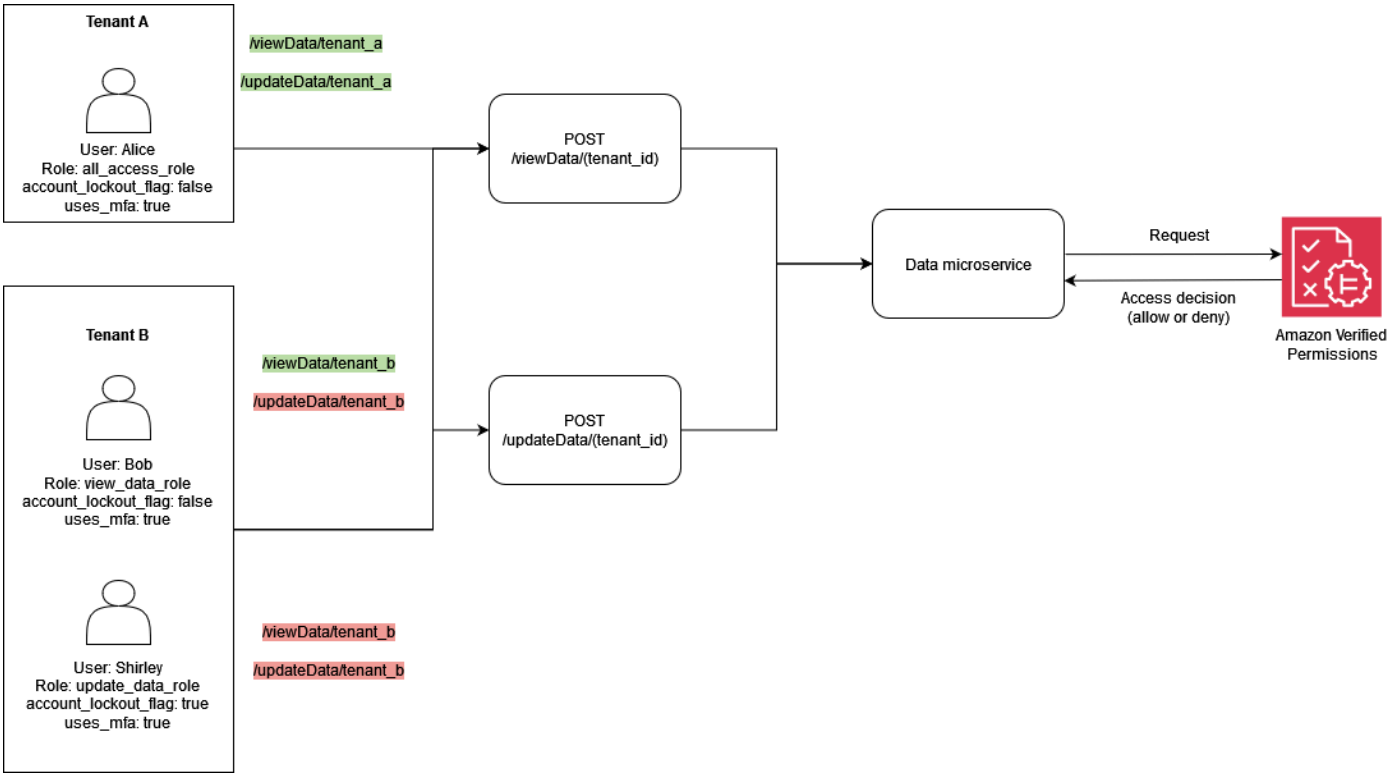
In diesem dritten Beispiel wird `User Alice` versucht, die `viewData` Aktion für die Ressource auszuführen`SampleData`. Diese Anforderung wird an den `DATAMICROSERVICE_POLICYSTORE_A` Richtlinienspeicher weitergeleitet, da der Principal zu Mandant A `Alice` gehört. Er `Alice` ist Teil der Gruppe `allAccessRole` des Typs`Role`, der es ihr ermöglicht, die `viewData` Aktion für Ressourcen auszuführen. Daher führt die Anfrage zu einer `ALLOW` Entscheidung.

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

Beispiel 4: Zugriffskontrolle für mehrere Mandanten mit RBAC und ABAC

Um das RBAC-Beispiel aus dem vorherigen Abschnitt zu erweitern, können Sie Benutzern Attribute hinzufügen, um einen hybriden RBAC-ABAC-Ansatz für die mehrinstanzenfähige Zugriffskontrolle zu erstellen. Dieses Beispiel enthält dieselben Rollen wie das vorherige Beispiel, fügt jedoch das Benutzerattribut und den Kontextparameter hinzu. `account_lockout_flag` `uses_mfa` Das Beispiel verfolgt auch einen anderen Ansatz zur Implementierung der mehrinstanzenfähigen Zugriffskontrolle mithilfe von RBAC und ABAC und verwendet einen gemeinsamen Richtlinienspeicher anstelle eines anderen Richtlinienspeichers für jeden Mandanten.



Dieses Beispiel stellt eine mehrinstanzenfähige SaaS-Lösung dar, bei der Sie Autorisierungsentscheidungen für Mandant A und Mandant B treffen müssen, ähnlich wie im vorherigen Beispiel.

Um die Benutzersperrfunktion zu implementieren, fügt das Beispiel das Attribut `account_lockout_flag` dem `User` Entitätsprinzipal in der Autorisierungsanfrage hinzu. Dieses Flag sperrt den Benutzerzugriff auf das System und gewährt dem gesperrten Benutzer `DENY` alle Rechte. Das `account_lockout_flag` Attribut ist der `User` Entität zugeordnet und gilt so lange, `User` bis das Kennzeichen in mehreren Sitzungen aktiv aufgehoben wird. In dem Beispiel wird die `when` Bedingung zur Auswertung `account_lockout_flag` verwendet.

Das Beispiel fügt auch Details zur Anfrage und Sitzung hinzu. Die Kontextinformationen geben an, dass die Sitzung mithilfe der Multi-Faktor-Authentifizierung authentifiziert wurde. Um diese Überprüfung zu implementieren, verwendet das Beispiel die `when` Bedingung, um das `uses_mfa` Flag als Teil des Kontextfeldes auszuwerten. Weitere Informationen zu bewährten Methoden für das Hinzufügen von Kontext finden Sie in der Cedar-Dokumentation.

```
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
};
```

Diese Richtlinie verhindert den Zugriff auf Ressourcen, es sei denn, die Ressource befindet sich in derselben Gruppe wie das `Tenant` Attribut des anfordernden Prinzipals. Dieser Ansatz zur Aufrechterhaltung der Mandantenisolierung wird als One Shared Multi-Tenant Policy Store-Ansatz bezeichnet. Weitere Informationen zu Designüberlegungen für verifizierte Berechtigungen für mehrinstanzenfähige SaaS-Anwendungen finden Sie im Abschnitt Überlegungen zum Entwurf verifizierter Berechtigungen für mehrere Mandanten.

Die Richtlinie stellt außerdem sicher, dass der Principal Mitglied von `allAccessRole` und ist und seine Aktionen auf und beschränkt. `viewData` `updateData` Darüber hinaus überprüft diese Richtlinie, ob dies der Fall `account_lockout_flag` ist `false` und ob der Kontextwert für als `uses_mfa` bewertet wird. `true`

In ähnlicher Weise stellt die folgende Richtlinie sicher, dass sowohl der Prinzipal als auch die Ressource demselben Mandanten zugeordnet sind, wodurch ein mandantenübergreifender Zugriff verhindert wird. Diese Richtlinie stellt außerdem sicher, dass der Principal Mitglied von ist, `viewDataRole` und beschränkt seine Aktionen auf. `viewData` Darüber hinaus wird überprüft, ob der Wert `account_lockout_flag` ist `false` und ob der Kontextwert für als `uses_mfa` bewertet wird. `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
};
```

Die dritte Richtlinie ähnelt der vorherigen. Die Richtlinie erfordert, dass die Ressource Mitglied der Gruppe ist, die der Entität entspricht, die vertreten wird durch`principal.Tenant`. Dadurch wird sichergestellt, dass sowohl der Prinzipal als auch die Ressource Mandant B zugeordnet sind, wodurch ein mandantenübergreifender Zugriff verhindert wird. Diese Richtlinie stellt sicher, dass der Principal Mitglied von ist, `updateDataRole` und beschränkt Aktionen auf. `updateData` Darüber hinaus überprüft diese Richtlinie, ob der Wert `account_lockout_flag` ist `false` und ob der Kontextwert für als `uses_mfa` bewertet wird. `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
};
```

Die folgende Autorisierungsanfrage wird anhand der drei zuvor in diesem Abschnitt erörterten Richtlinien bewertet. In dieser Autorisierungsanfrage `Alice` stellt der Principal vom Typ `User` und mit dem Wert von eine `updateData` Anfrage mit der Rolle`allAccessRole`. `Alice`hat das Attribut`Tenant`, dessen Wert ist`Tenant::"TenantA"`. Die Aktion`Alice`, die ausgeführt werden soll, ist `updateData,` und die Ressource, auf die sie angewendet werden soll, ist `SampleData` vom Typ`Data`. `SampleData`hat `TenantA` als übergeordnete Entität.

`Alice`Kann gemäß der ersten Richtlinie im `<DATAMICROSERVICE_POLICYSTOREID>` Richtlinienspeicher die `updateData` Aktion für die Ressource ausführen, vorausgesetzt, dass die Bedingungen in der `when` Klausel der Richtlinie erfüllt sind. Die erste Bedingung erfordert, dass das `principal.Tenant` Attribut ausgewertet wird`TenantA`. Die zweite Bedingung erfordert, dass das Attribut `account_lockout_flag` des Prinzipals `false` Die letzte Bedingung erfordert`uses_mfa`, dass `true` der Kontext Da alle drei Bedingungen erfüllt sind, gibt die Anfrage eine `ALLOW` Entscheidung zurück.

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

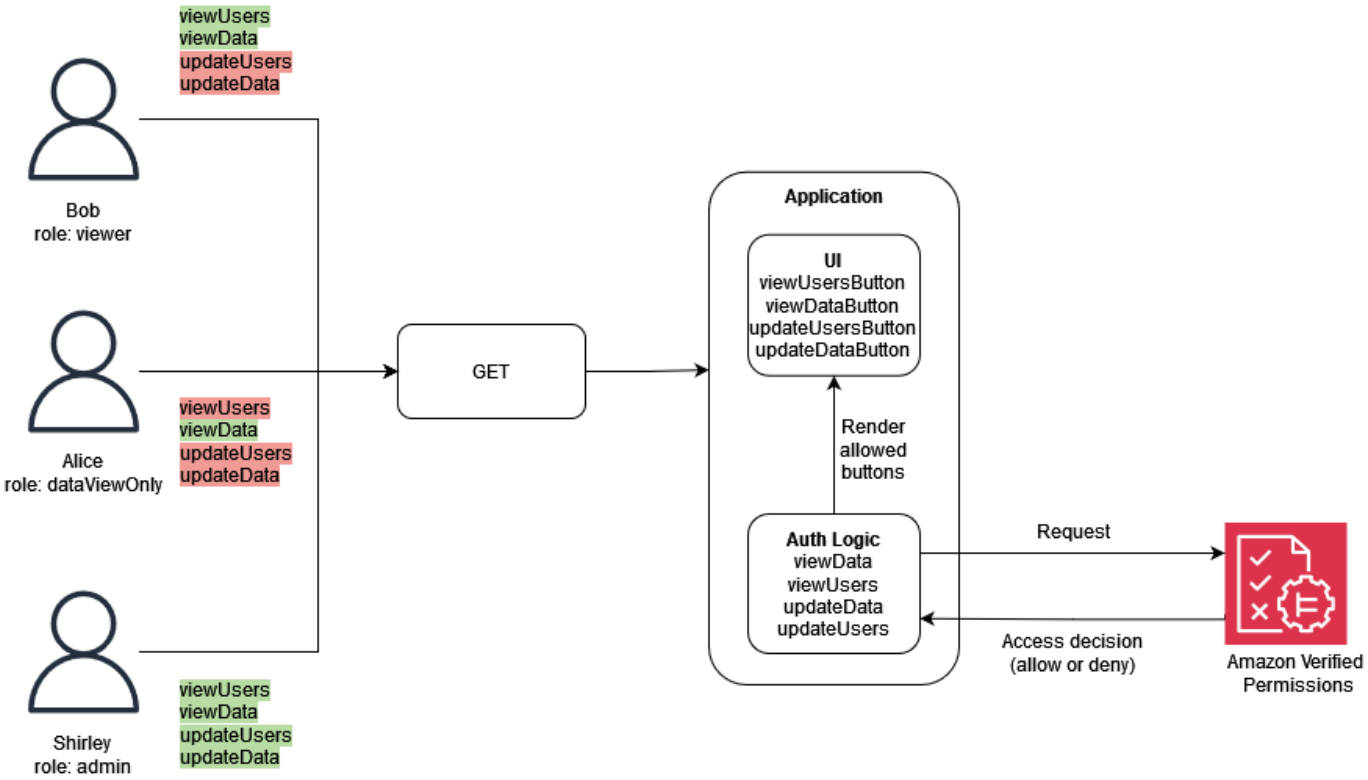
```
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "uses_mfa": {
            "boolean": true
        }
    }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

Beispiel 5: UI-Filterung mit verifizierten Berechtigungen und Cedar

Sie können Verified Permissions auch verwenden, um die RBAC-Filterung von UI-Elementen auf der Grundlage autorisierter Aktionen zu implementieren. Dies ist äußerst nützlich für Anwendungen mit kontextsensitiven Benutzeroberflächenelementen, die im Fall einer mehrinstanzenfähigen SaaS-Anwendung bestimmten Benutzern oder Mandanten zugeordnet sein könnten.

Im folgenden Beispiel dürfen `Users` von ihnen `Role` `viewer` keine Updates durchführen. Für diese Benutzer sollte die Benutzeroberfläche keine Aktualisierungsschaltflächen rendern.



In diesem Beispiel hat eine einseitige Webanwendung vier Schaltflächen. Welche Schaltflächen sichtbar sind, hängt `Role` von dem Benutzer ab, der gerade bei der Anwendung angemeldet ist. Beim Rendern der Benutzeroberfläche durch die einseitige Webanwendung werden verifizierte Berechtigungen abgefragt, um zu ermitteln, zu welchen Aktionen der Benutzer berechtigt ist, und generiert dann die Schaltflächen auf der Grundlage der Autorisierungsentscheidung.

Die folgende Richtlinie legt fest, dass der Typ `Role` mit dem Wert sowohl Benutzer als auch Daten anzeigen `viewer` kann. Eine `ALLOW` Autorisierungsentscheidung für diese Richtlinie erfordert eine `viewData` `viewUsers` OR-Aktion und erfordert außerdem, dass dem Typ `Data` oder eine Ressource zugeordnet wird`Users`. Eine `ALLOW` Entscheidung ermöglicht es der Benutzeroberfläche, zwei Schaltflächen zu rendern: `viewDataButton` und`viewUsersButton`.

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

Die folgende Richtlinie legt fest, dass der Typ `Role` mit dem Wert von nur Daten anzeigen `viewerDataOnly` kann. Eine `ALLOW` Autorisierungsentscheidung für diese Richtlinie erfordert eine `viewData` Aktion und erfordert außerdem, dass dem Typ eine Ressource zugeordnet wird`Data`. Eine `ALLOW` Entscheidung ermöglicht es der Benutzeroberfläche, die Schaltfläche zu rendern`viewDataButton`.

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

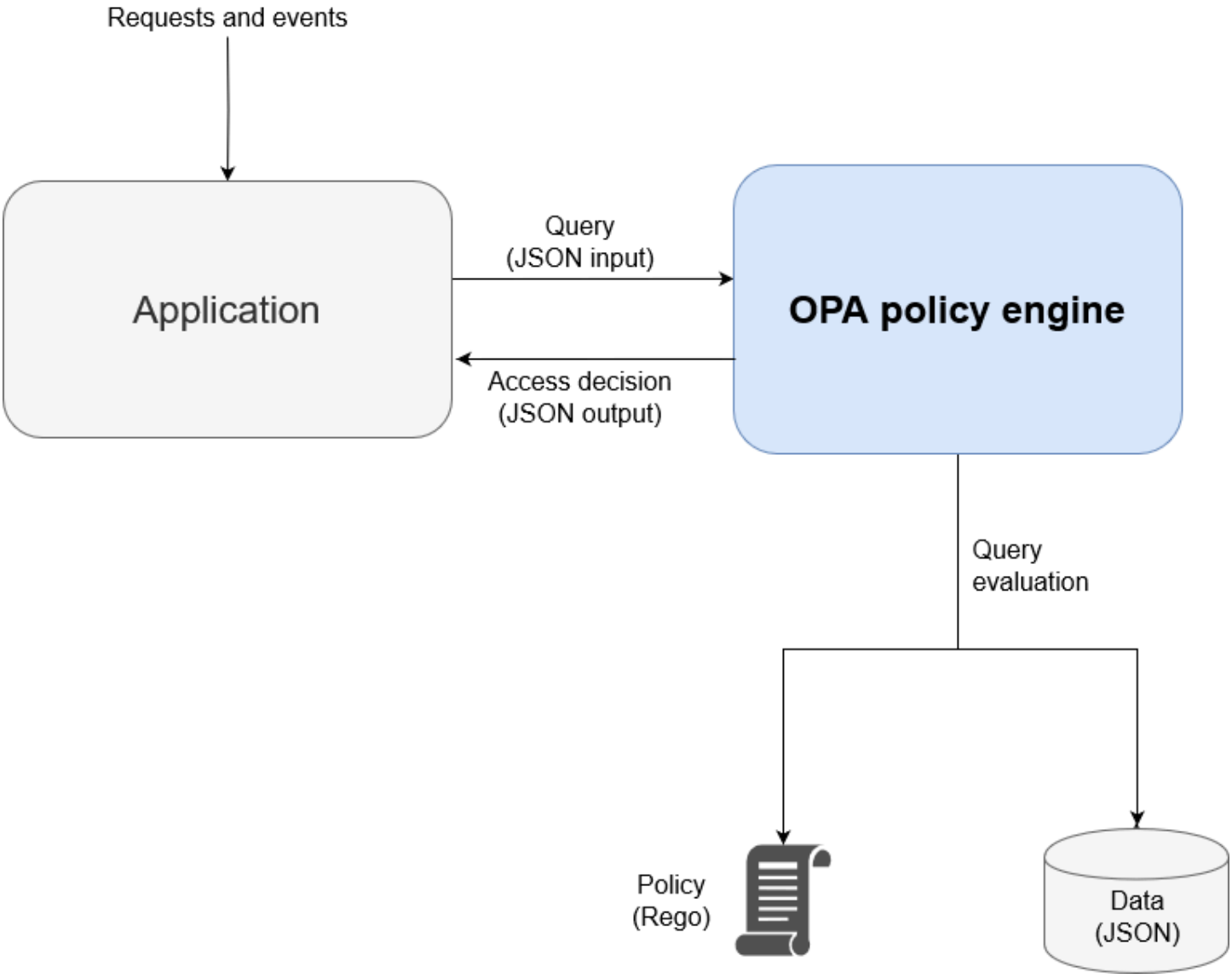
Die folgende Richtlinie legt fest, dass der Typ `Role` mit dem Wert Daten und Benutzer bearbeiten und anzeigen `admin` kann. Eine `ALLOW` Autorisierungsentscheidung für diese Richtlinie erfordert die Aktion `updateData` `updateUsers``viewUsers`, `viewData,` oder und außerdem muss dem Typ `Data` oder eine Ressource zugeordnet werden`Users`. Eine `ALLOW` Entscheidung ermöglicht es der Benutzeroberfläche, alle vier Schaltflächen zu rendern: `updateDataButton``updateUsersButton`,`viewDataButton`, und`viewUsersButton`.

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

Implementierung eines PDP mithilfe von OPA

Der Open Policy Agent (OPA) ist eine Open-Source-Engine für allgemeine Richtlinien. OPA hat viele Anwendungsfälle, aber der für die PDP-Implementierung relevante Anwendungsfall ist die Fähigkeit, die Autorisierungslogik von einer Anwendung zu entkoppeln. Dies wird als Richtlinienentkopplung bezeichnet. OPA ist bei der Implementierung eines PDP aus mehreren Gründen nützlich. Es

verwendet eine deklarative Sprache auf hoher Ebene namens Rego, um Richtlinien und Regeln zu entwerfen. Diese Richtlinien und Regeln existieren unabhängig von einer Anwendung und können Autorisierungsentscheidungen ohne anwendungsspezifische Logik treffen. OPA stellt außerdem eine RESTful-API zur Verfügung, um das Abrufen von Autorisierungsentscheidungen einfach und unkompliziert zu gestalten. Um eine Autorisierungsentscheidung zu treffen, fragt eine Anwendung OPA mit JSON-Eingabe ab, und OPA bewertet die Eingabe anhand der angegebenen Richtlinien, um eine Zugriffsentscheidung in JSON zurückzugeben. OPA ist auch in der Lage, externe Daten zu importieren, die für eine Autorisierungsentscheidung relevant sein könnten.



OPA bietet mehrere Vorteile gegenüber benutzerdefinierten Policy-Engines:

- OPA und seine Richtlinienbewertung mit Rego bieten eine flexible, vorgefertigte Richtlinien-Engine, die nur das Einfügen von Richtlinien und allen Daten erfordert, die für

Autorisierungsentscheidungen erforderlich sind. Diese Logik zur Richtlinienbewertung müsste in einer benutzerdefinierten Policy-Engine-Lösung neu erstellt werden.

- OPA vereinfacht die Autorisierungslogik, indem Richtlinien in einer deklarativen Sprache geschrieben werden. Sie können diese Richtlinien und Regeln unabhängig von jedem Anwendungscode ändern und verwalten, ohne Kenntnisse in der Anwendungsentwicklung zu haben.
- OPA stellt eine RESTful-API zur Verfügung, die die Integration mit Points zur Durchsetzung von Richtlinien (PEPs) vereinfacht.
- OPA bietet integrierte Unterstützung für die Validierung und Dekodierung von JSON-Web-Tokens (JWTs).
- OPA ist ein anerkannter Autorisierungsstandard, was bedeutet, dass es zahlreiche Dokumentationen und Beispiele gibt, falls Sie Hilfe oder Nachforschungen zur Lösung eines bestimmten Problems benötigen.
- Durch die Einführung eines Autorisierungsstandards wie OPA können in Rego geschriebene Richtlinien von allen Teams gemeinsam genutzt werden, unabhängig von der Programmiersprache, die von der jeweiligen Teamanwendung verwendet wird.

Es gibt zwei Dinge, die OPA nicht automatisch bereitstellt:

- OPA verfügt nicht über eine robuste Kontrollebene für die Aktualisierung und Verwaltung von Richtlinien. OPA bietet einige grundlegende Muster für die Implementierung von Richtlinienaktualisierungen, Überwachung und Protokollaggregation, indem eine Verwaltungs-API verfügbar gemacht wird. Die Integration mit dieser API muss jedoch vom OPA-Benutzer vorgenommen werden. Als bewährte Methode sollten Sie eine CI/CD-Pipeline (Continuous Integration and Continuous Deployment) verwenden, um Richtlinienversionen zu verwalten, zu ändern und nachzuverfolgen und Richtlinien in OPA zu verwalten.
- OPA kann standardmäßig keine Daten aus externen Quellen abrufen. Eine externe Datenquelle für eine Autorisierungsentscheidung könnte eine Datenbank sein, die Benutzerattribute enthält. Es gibt eine gewisse Flexibilität bei der Bereitstellung externer Daten für OPA — sie können vorab lokal zwischengespeichert oder dynamisch von einer API abgerufen werden, wenn eine Autorisierungsentscheidung angefordert wird —, aber OPA kann diese Informationen nicht in Ihrem Namen abrufen.

Überblick über Rego

Rego ist eine allgemeine Richtliniensprache, was bedeutet, dass sie für jede Ebene des Stacks und jede Domäne funktioniert. Der Hauptzweck von Rego besteht darin, JSON/YAML-Eingaben und -Daten zu akzeptieren, die ausgewertet werden, um richtliniengestützte Entscheidungen über Infrastrukturressourcen, Identitäten und Abläufe zu treffen. Mit Rego können Sie Richtlinien für jede Ebene eines Stacks oder einer Domain schreiben, ohne dass eine Änderung oder Erweiterung der Sprache erforderlich ist. Hier sind einige Beispiele für Entscheidungen, die Rego treffen kann:

- Ist diese API-Anfrage zulässig oder verweigert?
- Wie lautet der Hostname des Backup-Servers für diese Anwendung?
- Wie hoch ist die Risikobewertung für diese vorgeschlagene Infrastrukturänderung?
- In welchen Clustern sollte dieser Container für hohe Verfügbarkeit bereitgestellt werden?
- Welche Routing-Informationen sollten für diesen Microservice verwendet werden?

Um diese Fragen zu beantworten, verwendet Rego eine grundlegende Philosophie darüber, wie diese Entscheidungen getroffen werden können. Die beiden wichtigsten Grundsätze bei der Ausarbeitung von Richtlinien in Rego sind:

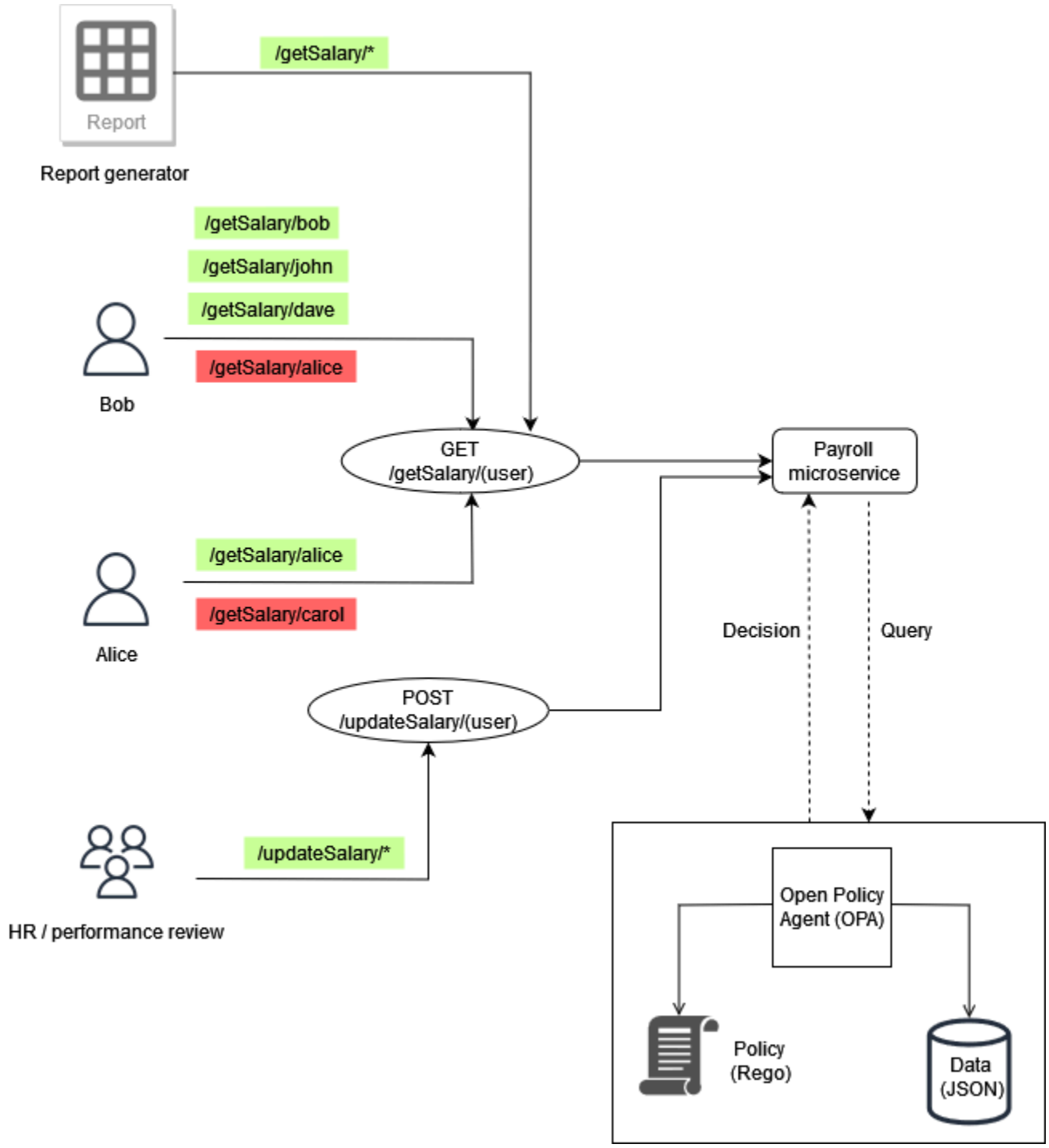
- Jede Ressource, Identität oder Operation kann als JSON- oder YAML-Daten dargestellt werden.
- Eine Richtlinie ist Logik, die auf Daten angewendet wird.

Rego hilft Softwaresystemen dabei, Autorisierungsentscheidungen zu treffen, indem es eine Logik dafür definiert, wie Eingaben von JSON/YAML-Daten ausgewertet werden. Programmiersprachen wie C, Java, Go und Python sind die übliche Lösung für dieses Problem, aber Rego wurde so konzipiert, dass es sich auf die Daten und Eingaben konzentriert, die Ihr System repräsentieren, und auf die Logik, mit der Sie anhand dieser Informationen politische Entscheidungen treffen können.

Beispiel 1: Grundlegendes ABAC mit OPA und Rego

In diesem Abschnitt wird ein Szenario beschrieben, in dem OPA verwendet wird, um Zugriffsentscheidungen darüber zu treffen, welche Benutzer auf Informationen in einem fiktiven Payroll-Microservice zugreifen dürfen. Es werden Rego-Codefragmente bereitgestellt, um zu demonstrieren, wie Sie Rego verwenden können, um Entscheidungen zur Zugriffskontrolle zu treffen. Diese Beispiele sind weder erschöpfend noch stellen sie eine vollständige Untersuchung der

Funktionen von Rego und OPA dar. Für einen umfassenderen Überblick über Rego empfehlen wir Ihnen, die Rego-Dokumentation auf der OPA-Website zu lesen.



Beispiel für grundlegende OPA-Regeln

Im vorherigen Diagramm lautet eine der Zugriffskontrollregeln, die von OPA für den Payroll-Microservice durchgesetzt wurden:

Mitarbeiter können ihr Gehalt selbst ablesen.

Wenn Bob versucht, auf den Payroll-Microservice zuzugreifen, um sein eigenes Gehalt einzusehen, kann der Payroll-Microservice den API-Aufruf an die OPA RESTful API weiterleiten, um eine Zugriffsentscheidung zu treffen. Der Payroll-Service fragt OPA mit der folgenden JSON-Eingabe nach einer Entscheidung ab:

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

OPA wählt auf der Grundlage der Abfrage eine oder mehrere Richtlinien aus. In diesem Fall wertet die folgende Richtlinie, die in Rego geschrieben ist, die JSON-Eingabe aus.

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

Diese Richtlinie verweigert standardmäßig den Zugriff. Anschließend wird die Eingabe in der Abfrage ausgewertet, indem sie an die globale Variable gebunden wird. `input` Der Punktoperator wird mit dieser Variablen verwendet, um auf die Werte der Variablen zuzugreifen. Die Rego-Regel `allow` gibt true zurück, wenn die Ausdrücke in der Regel ebenfalls wahr sind. Die Rego-Regel überprüft, ob `method` in der Eingabe GET entspricht. Anschließend wird überprüft, ob das erste Element in der Liste vorhanden `path` ist, `getSalary` bevor der Variablen das zweite Element in der Liste zugewiesen wird. `user` Schließlich überprüft es, ob es sich um den Pfad handelt, auf den zugegriffen wird, `/getSalary/bob` indem überprüft wird, ob die `user` Anfrage,, gestellt wird`input.user`, mit der `user` Variablen übereinstimmt. Die Regel `allow` wendet die Wenn-Dann-Logik an, um einen booleschen Wert zurückzugeben, wie in der Ausgabe gezeigt:

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

Teilregel, die externe Daten verwendet

Um zusätzliche OPA-Funktionen zu demonstrieren, können Sie der Zugriffsregel, die Sie durchsetzen, Anforderungen hinzufügen. Nehmen wir an, Sie möchten diese Zugriffskontrollanforderung im Kontext der vorherigen Abbildung durchsetzen:

Mitarbeiter können das Gehalt aller Mitarbeiter ablesen, die ihnen unterstellt sind.

In diesem Beispiel hat OPA Zugriff auf externe Daten, die importiert werden können, um eine Zugriffsentscheidung zu treffen:

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

Sie können eine beliebige JSON-Antwort generieren, indem Sie in OPA eine Teilregel erstellen, die statt einer festen Antwort eine Reihe von Werten zurückgibt. Dies ist ein Beispiel für eine Teilregel:

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

Diese Regel gibt eine Gruppe aller Benutzer zurück, die über den Wert von berichten`input.user`, der in diesem Fall ist`bob`. Das `[_]` Konstrukt in der Regel wird verwendet, um über die Werte der Menge zu iterieren. Dies ist die Ausgabe der Regel:

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

Durch das Abrufen dieser Informationen kann festgestellt werden, ob ein Benutzer einem Manager direkt unterstellt ist. Für einige Anwendungen ist die Rückgabe von dynamischem JSON der Rückgabe einer einfachen booleschen Antwort vorzuziehen.

Zusammenführung

Die letzte Zugriffsanforderung ist komplexer als die ersten beiden, da sie die in beiden Anforderungen angegebenen Bedingungen kombiniert:

Die Mitarbeiter können ihr eigenes Gehalt und das Gehalt aller Personen, die ihnen unterstellt sind, ablesen.

Um diese Anforderung zu erfüllen, können Sie diese Rego-Richtlinie verwenden:

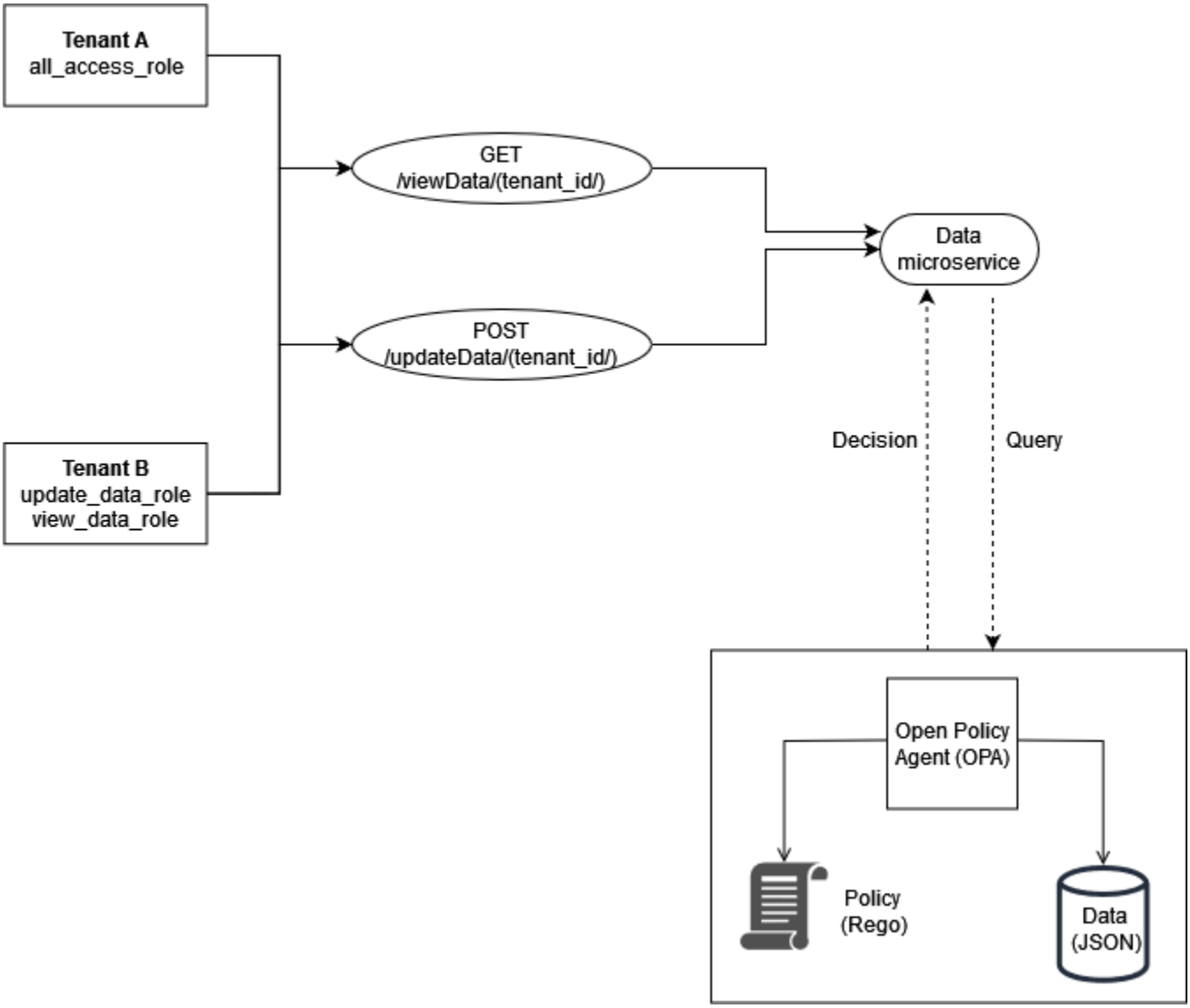
```
default allow = false

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)
}
```

Die erste Regel in der Richtlinie gewährt jedem Benutzer Zugriff, der versucht, seine eigenen Gehaltsinformationen einzusehen, wie bereits beschrieben. Zwei Regeln mit demselben Namen zu haben`allow`, funktioniert in Rego als logischer Operator oder. Die zweite Regel ruft die Liste aller direkt unterstellten Mitarbeiter ab, die mit `input.user` (aus den Daten im vorherigen Diagramm) verknüpft sind, und weist diese Liste der Variablen zu. `managers` Schließlich überprüft die Regel, ob der Benutzer, der versucht, sein Gehalt einzusehen, direkt unterstellt ist, `input.user` indem überprüft wird, ob sein Name in der Variablen enthalten ist. `managers`

Die Beispiele in diesem Abschnitt sind sehr einfach und bieten keine vollständige oder gründliche Erläuterung der Funktionen von Rego und OPA. Weitere Informationen finden Sie in der OPA-Dokumentation, in der GitHub OPA-README-Datei und experimentieren Sie im Rego Playground.

Beispiel 2: Mehrmandantenfähige Zugriffskontrolle und benutzerdefiniertes RBAC mit OPA und Rego

In diesem Beispiel wird anhand von OPA und Rego demonstriert, wie die Zugriffskontrolle auf einer API für eine Mehrmandantenanwendung mit benutzerdefinierten Rollen implementiert werden kann, die von Mandantenbenutzern definiert werden. Es zeigt auch, wie der Zugriff je nach Mandant eingeschränkt werden kann. Dieses Modell zeigt, wie OPA detaillierte Genehmigungsentscheidungen auf der Grundlage von Informationen treffen kann, die in einer hochrangigen Rolle bereitgestellt werden.



Die Rollen für die Mandanten werden in externen Daten (RBAC-Daten) gespeichert, die verwendet werden, um Zugriffsentscheidungen für OPA zu treffen:

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

Wenn diese Rollen von einem Mandantenbenutzer definiert werden, sollten sie in einer externen Datenquelle oder einem Identitätsanbieter (IdP) gespeichert werden, der als Informationsquelle bei der Zuordnung von mandantendefinierten Rollen zu Berechtigungen und zum Mandanten selbst dienen kann.

In diesem Beispiel werden zwei Richtlinien in OPA verwendet, um Autorisierungsentscheidungen zu treffen und zu untersuchen, wie diese Richtlinien die Mandantenisolierung erzwingen. Diese Richtlinien verwenden die zuvor definierten RBAC-Daten.

```
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")
}
```

Um zu verdeutlichen, wie diese Regel funktioniert, stellen Sie sich eine OPA-Abfrage mit der folgenden Eingabe vor:

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

Eine Autorisierungsentscheidung für diesen API-Aufruf wird wie folgt getroffen, indem die RBAC-Daten, die OPA-Richtlinien und die OPA-Abfrageeingabe kombiniert werden:

1. Ein Benutzer von tätigt einen `Tenant A` API-Aufruf an. `/viewData/tenant_a`
2. Der Data-Microservice empfängt den Aufruf und fragt die `allowViewData` Regel ab. Dabei wird die im Beispiel für die OPA-Abfrageeingabe gezeigte Eingabe übergeben.
3. OPA verwendet die abgefragte Regel in OPA-Richtlinien, um die bereitgestellten Eingaben auszuwerten. OPA verwendet auch die Daten aus RBAC-Daten, um die Eingabe auszuwerten. OPA macht Folgendes:
   a. Überprüft, ob die für den API-Aufruf verwendete Methode `GET`
   b. Überprüft, ob der angeforderte Pfad `viewData`
   c. Überprüft, ob `tenant_id` der Pfad mit dem Pfad übereinstimmt, der dem Benutzer `input.tenant_id` zugeordnet ist. Dadurch wird sichergestellt, dass die Mandantenisolierung aufrechterhalten wird. Ein anderer Mandant kann, selbst mit einer identischen Rolle, nicht autorisiert werden, diesen API-Aufruf durchzuführen.
   d. Ruft eine Liste mit Rollenberechtigungen aus den externen Daten der Rollen ab und weist sie der Variablen zu. `role_permissions` Diese Liste wird mithilfe der vom Mandanten definierten Rolle abgerufen, die dem Benutzer in zugeordnet ist `input.role.`
   e. Überprüft`role_permissions`, ob es die Berechtigung enthält `viewData.`
4. OPA gibt die folgende Entscheidung an den Data-Microservice zurück:

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

Dieser Prozess zeigt, wie RBAC und Tenant Awareness dazu beitragen können, eine Autorisierungsentscheidung mit OPA zu treffen. Um diesen Punkt weiter zu veranschaulichen, sollten Sie einen API-Aufruf von `/viewData/tenant_b` mit der folgenden Abfrageeingabe in Betracht ziehen:

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

```
}
```

Diese Regel würde dieselbe Ausgabe wie die OPA-Abfrageeingabe zurückgeben, obwohl sie für einen anderen Mandanten gilt, der eine andere Rolle innehat. Das liegt daran, dass dieser Aufruf für die RBAC-Daten bestimmt ist `/tenant_b` und den `view_data_role` darin enthaltenen RBAC-Daten immer noch die `viewData` entsprechende Berechtigung zugewiesen ist. Um dieselbe Art von Zugriffskontrolle durchzusetzen`/updateData`, können Sie eine ähnliche OPA-Regel verwenden:

```
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")
}
```

Diese Regel entspricht funktionell der `allowViewData` Regel, überprüft jedoch einen anderen Pfad und eine andere Eingabemethode. Die Regel gewährleistet weiterhin die Mandantenisolierung und überprüft, ob die vom Mandanten definierte Rolle dem API-Aufrufer die Berechtigung erteilt. Um zu sehen, wie dies durchgesetzt werden könnte, überprüfen Sie die folgende Abfrageeingabe für einen API-Aufruf an: `/updateData/tenant_b`

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

Diese Abfrageeingabe gibt, wenn sie mit der `allowUpdateData` Regel ausgewertet wird, die folgende Autorisierungsentscheidung zurück:

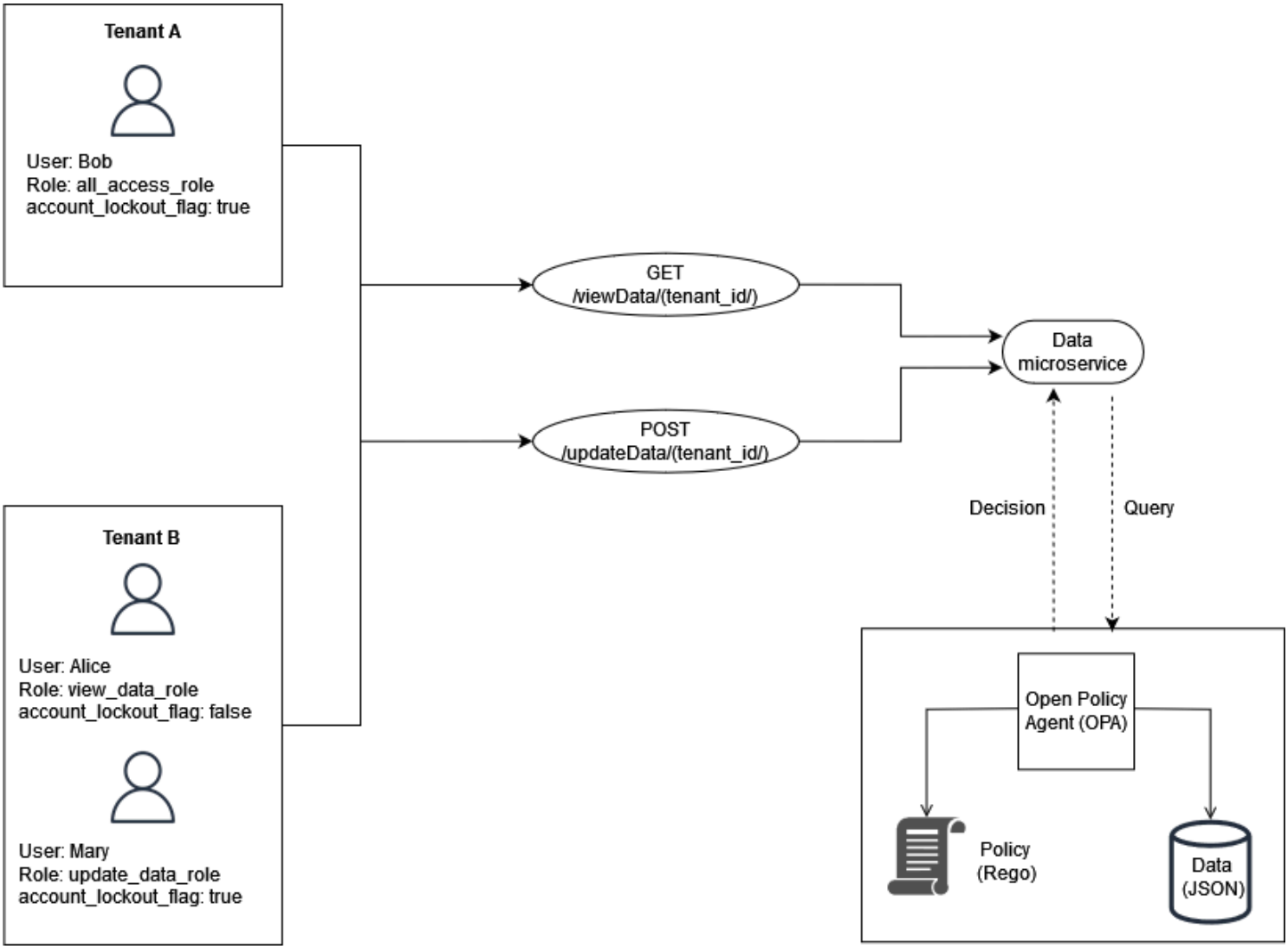
```
{
    "allowUpdateData": false
}
```

Dieser Anruf wird nicht autorisiert. Der API-Aufrufer ist zwar mit der richtigen verknüpft `tenant_id` und ruft die API mithilfe einer zugelassenen Methode auf, aber die `input.role` ist die vom

Mandanten `view_data_role` definierte. Der `view_data_role` hat nicht die `updateData` Erlaubnis, daher ist der Aufruf von nicht autorisiert. `/updateData` Dieser Aufruf wäre für einen `tenant_b` Benutzer erfolgreich gewesen, der über das verfügt`update_data_role`.

Beispiel 3: Mehrmandantenfähige Zugriffskontrolle für RBAC und ABAC mit OPA und Rego

Um das RBAC-Beispiel aus dem vorherigen Abschnitt zu erweitern, können Sie Benutzern Attribute hinzufügen.



Dieses Beispiel enthält dieselben Rollen wie im vorherigen Beispiel, fügt jedoch das Benutzerattribut hinzu. `account_lockout_flag` Dies ist ein benutzerspezifisches Attribut, das keiner bestimmten Rolle zugeordnet ist. Sie können dieselben externen RBAC-Daten verwenden, die Sie zuvor für dieses Beispiel verwendet haben:

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

Das `account_lockout_flag` Benutzerattribut kann als Teil der Eingabe für eine OPA-Abfrage `/viewData/tenant_a` für den Benutzer Bob an den Datendienst übergeben werden:

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

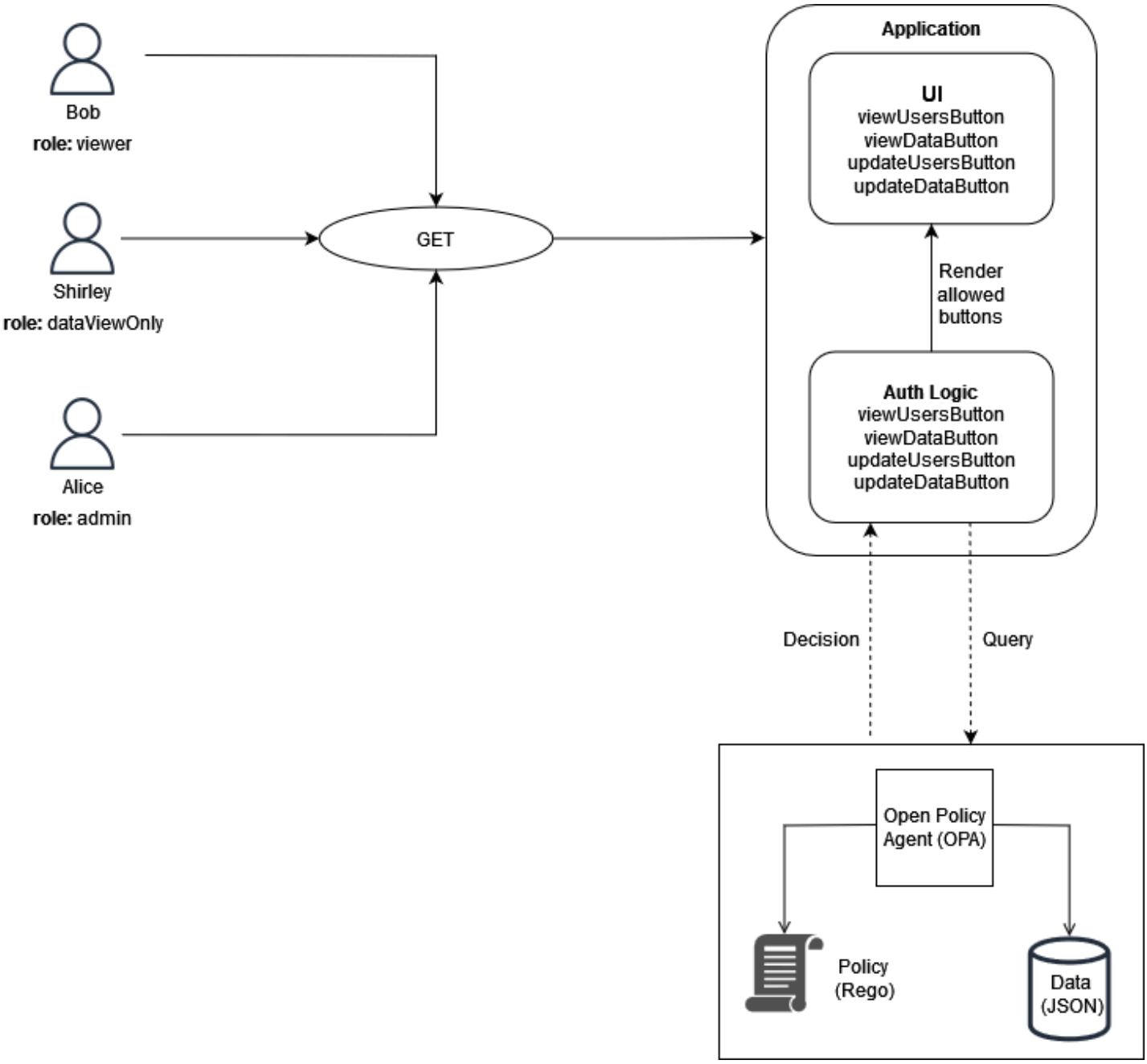
Die Regel, die für die Zugriffsentscheidung abgefragt wird, ähnelt den vorherigen Beispielen, enthält jedoch eine zusätzliche Zeile, um nach dem Attribut zu suchen: `account_lockout_flag`

```
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"
}
```

Diese Abfrage gibt eine Autorisierungsentscheidung von zurück. `false` Das liegt daran, dass die `account_lockout_flag attribute` `true` für Bob gilt und die Rego-Regel den Zugriff `allowViewData` verweigert, obwohl Bob die richtige Rolle und den richtigen Mandanten hat.

Beispiel 4: UI-Filterung mit OPA und Rego

Die Flexibilität von OPA und Rego unterstützt die Möglichkeit, UI-Elemente zu filtern. Das folgende Beispiel zeigt, wie eine OPA-Teilregel Autorisierungsentscheidungen darüber treffen kann, welche Elemente in einer Benutzeroberfläche mit RBAC angezeigt werden sollen. Diese Methode ist eine von vielen verschiedenen Möglichkeiten, UI-Elemente mit OPA zu filtern.



In diesem Beispiel hat eine einseitige Webanwendung vier Schaltflächen. Nehmen wir an, Sie möchten die Benutzeroberfläche von Bob, Shirley und Alice filtern, sodass sie nur die Schaltflächen

sehen können, die ihren Rollen entsprechen. Wenn die Benutzeroberfläche eine Anfrage vom Benutzer erhält, fragt sie eine OPA-Teilregel ab, um zu bestimmen, welche Schaltflächen auf der Benutzeroberfläche angezeigt werden sollen. Die Abfrage übergibt Folgendes als Eingabe an OPA, wenn Bob (mit der Rolle`viewer`) eine Anfrage an die Benutzeroberfläche stellt:

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

OPA verwendet externe Daten, die für RBAC strukturiert sind, um eine Zugriffsentscheidung zu treffen:

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

Die OPA-Teilregel verwendet sowohl die externen Daten als auch die Eingabe, um eine Liste der zulässigen Aktionen zu erstellen:

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

In der Teilregel verwendet OPA die als Teil der Abfrage `input.role` angegebenen Werte, um zu bestimmen, welche Schaltflächen angezeigt werden sollen. Bob hat die Rolle`viewer`, und die externen Daten geben an, dass Zuschauer über zwei Berechtigungen verfügen: `viewUsers` und`viewData`. Daher sieht die Ausgabe dieser Regel für Bob (und für alle anderen Benutzer, die eine Zuschauerrolle haben) wie folgt aus:

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

Die Ausgabe für Shirley, die die `dataViewOnly` Rolle innehat, würde eine Schaltfläche mit Berechtigungen enthalten:`viewData`. Die Ausgabe für Alice, die die `admin` Rolle innehat, würde all diese Berechtigungen enthalten. Diese Antworten werden an die Benutzeroberfläche zurückgegeben, wenn nach OPA gefragt wird. `user_permissions` Die Anwendung kann diese Antwort dann verwenden, um die,`viewUsersButton`, `viewDataButton` und die ein- oder auszublenden. `updateUsersButton` `updateDataButton`

Verwenden einer benutzerdefinierten Policy-Engine

Eine alternative Methode zur Implementierung eines PDP ist die Erstellung einer benutzerdefinierten Policy-Engine. Das Ziel dieser Policy-Engine besteht darin, die Autorisierungslogik von einer Anwendung zu entkoppeln. Die benutzerdefinierte Policy-Engine ist dafür verantwortlich, Autorisierungsentscheidungen zu treffen, ähnlich wie Verified Permissions oder OPA, um die Richtlinienentkopplung zu erreichen. Der Hauptunterschied zwischen dieser Lösung und der Verwendung von Verified Permissions oder OPA besteht darin, dass die Logik für das Schreiben und Auswerten von Richtlinien auf eine benutzerdefinierte Policy-Engine zugeschnitten ist. Alle Interaktionen mit der Engine müssen über eine API oder eine andere Methode offengelegt werden, damit Autorisierungsentscheidungen an eine Anwendung weitergeleitet werden können. Sie können eine benutzerdefinierte Policy-Engine in einer beliebigen Programmiersprache schreiben oder andere Mechanismen zur Richtlinienbewertung verwenden, z. B. die Common Expression Language (CEL).

Implementierung eines PEP

Eine Policy Enforcement Point (PEP) ist für die Entgegennahme von Autorisierungsanfragen zuständig, die zur Bewertung an den Policy Decision Point (PDP) gesendet werden. Ein PEP kann sich überall in einer Anwendung befinden, wo Daten und Ressourcen geschützt werden müssen oder wo Autorisierungslogik angewendet wird. PEPs sind im Vergleich zu PDPs relativ einfach. Ein PEP ist nur dafür verantwortlich, eine Autorisierungsentscheidung anzufordern und auszuwerten, und benötigt keine Autorisierungslogik. PEPs können im Gegensatz zu PDPs nicht in einer SaaS-Anwendung zentralisiert werden. Dies liegt daran, dass Autorisierung und Zugriffskontrolle in der gesamten Anwendung und ihren Zugriffspunkten implementiert werden müssen. PEPs können auf APIs, Microservices, Backend for Frontend (BFF) -Ebenen oder an jeder Stelle in der Anwendung angewendet werden, an der Zugriffskontrolle gewünscht oder erforderlich ist. Wenn PEPs in einer Anwendung allgegenwärtig sind, wird sichergestellt, dass die Autorisierung häufig und unabhängig an mehreren Stellen überprüft wird.

Um ein PEP zu implementieren, muss zunächst festgelegt werden, wo die Durchsetzung der Zugriffskontrolle in einer Anwendung erfolgen soll. Beachten Sie diesen Grundsatz, wenn Sie entscheiden, wo PEPs in Ihre Anwendung integriert werden sollten:

Wenn eine Anwendung eine API verfügbar macht, sollte es für diese API eine Autorisierung und Zugriffskontrolle geben.

Dies liegt daran, dass APIs in einer microservices-orientierten oder serviceorientierten Architektur als Trennzeichen zwischen verschiedenen Anwendungsfunktionen dienen. Es ist sinnvoll, die Zugriffskontrolle als logische Checkpoints zwischen Anwendungsfunktionen einzubeziehen. Wir empfehlen dringend, PEPs als Voraussetzung für den Zugriff auf jede API in einer SaaS-Anwendung einzubeziehen. Es ist auch möglich, die Autorisierung an anderen Stellen einer Anwendung zu integrieren. Bei monolithischen Anwendungen kann es erforderlich sein, PEPs in die Logik der Anwendung selbst zu integrieren. Es gibt keinen einzigen Ort, an dem PEPs aufgenommen werden sollten. Sie sollten jedoch das API-Prinzip als Ausgangspunkt verwenden.

Beantragung einer Autorisierungsentscheidung

Ein PEP muss beim PDP eine Autorisierungsentscheidung beantragen. Die Anfrage kann verschiedene Formen annehmen. Die einfachste und zugänglichste Methode, um eine Autorisierungsentscheidung anzufordern, besteht darin, eine Autorisierungsanfrage oder -abfrage an eine RESTful-API zu senden, die vom PDP (OPA oder Verified Permissions) verfügbar gemacht wird.

Wenn Sie Verified Permissions verwenden, können Sie die IsAuthorizedMethode auch mithilfe des AWS SDK aufrufen, um eine Autorisierungsentscheidung abzurufen. Die einzige Funktion eines PEP in diesem Muster besteht darin, die Informationen weiterzuleiten, die für die Autorisierungsanfrage oder -abfrage benötigt werden. Dies kann so einfach sein wie das Weiterleiten einer von einer API empfangenen Anfrage als Eingabe an das PDP. Es gibt andere Methoden zum Erstellen von PEPs. Sie können beispielsweise eine OPA-PDP lokal in eine in der Programmiersprache Go geschriebene Anwendung als Bibliothek integrieren, anstatt eine API zu verwenden.

Bewertung einer Autorisierungsentscheidung

PEPs müssen Logik zur Bewertung der Ergebnisse einer Autorisierungsentscheidung beinhalten. Wenn PDPs als APIs verfügbar gemacht werden, ist die Autorisierungsentscheidung wahrscheinlich im JSON-Format und wird durch einen API-Aufruf zurückgegeben. Das PEP muss diesen JSON-Code auswerten, um festzustellen, ob die ergriffene Aktion autorisiert ist. Wenn ein PDP beispielsweise so konzipiert ist, dass es eine boolesche Entscheidung über das Zulassen oder Verweigern der Autorisierung bereitstellt, könnte das PEP einfach diesen Wert überprüfen und dann den HTTP-Statuscode 200 für Zulassen und den HTTP-Statuscode 403 für Verweigern zurückgeben. Dieses Muster der Integration eines PEP als Voraussetzung für den Zugriff auf eine API ist ein einfach zu implementierendes und hocheffektives Muster für die Implementierung der Zugriffskontrolle in einer SaaS-Anwendung. In komplizierteren Szenarien könnte das PEP für die Auswertung von beliebigem JSON-Code verantwortlich sein, der vom PDP zurückgegeben wird. Das PEP muss so geschrieben werden, dass es die Logik enthält, die zur Interpretation der vom PDP zurückgegebenen Autorisierungsentscheidung erforderlich ist. Da ein PEP wahrscheinlich an vielen verschiedenen Stellen in Ihrer Anwendung implementiert wird, empfehlen wir Ihnen, Ihren PEP-Code als wiederverwendbare Bibliothek oder Artefakt in der Programmiersprache Ihrer Wahl zu verpacken. Auf diese Weise kann Ihr PEP problemlos und mit minimalem Nachaufwand an jeder Stelle in Ihrer Anwendung integriert werden.

Entwerfen Sie Modelle für mehrinstanzenfähige SaaS-Architekturen

Es gibt viele Möglichkeiten, API-Zugriffskontrolle und -autorisierung zu implementieren. Dieser Leitfaden konzentriert sich auf drei Entwurfsmodelle, die für SaaS-Architekturen mit mehreren Mandanten wirksam sind. Diese Designs dienen als allgemeine Referenz für die Implementierung von Policy Decision Points (PDP) und Policy Enforcement Points (PEPs), um ein kohärentes und allgegenwärtiges Autorisierungsmodell für Anwendungen zu bilden.

Entwurfsmodelle:

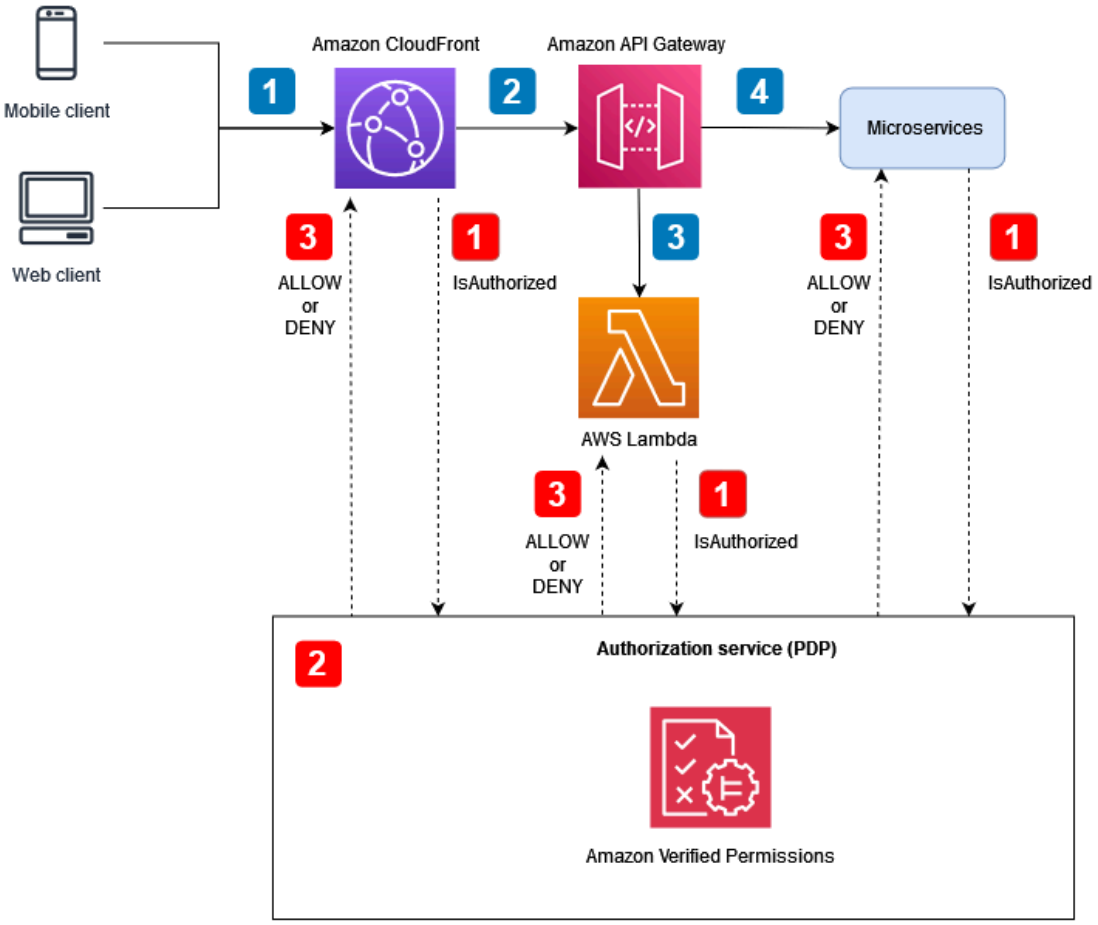
- Entwerfen Sie Modelle für Amazon Verified Permissions
- Entwerfen Sie Modelle für OPA

Entwerfen Sie Modelle für Amazon Verified Permissions

Verwendung eines zentralisierten PDP mit PEPs auf APIs

Das Modell eines zentralisierten Policy-Decision-Punkts (PDP) mit Policy-Enforcement-Punkten (PEPs) auf APIs folgt branchenweit bewährten Verfahren zur Schaffung eines effektiven und einfach zu wartenden Systems für die API-Zugriffskontrolle und -autorisierung. Dieser Ansatz unterstützt mehrere wichtige Prinzipien:

- Autorisierung und API-Zugriffskontrolle werden an mehreren Stellen in der Anwendung angewendet.
- Die Autorisierungslogik ist unabhängig von der Anwendung.
- Entscheidungen zur Zugriffskontrolle sind zentralisiert.



Anwendungsablauf (im Diagramm mit blau nummerierten Callouts dargestellt):

1. Ein authentifizierter Benutzer mit einem JSON Web Token (JWT) generiert eine HTTP-Anfrage an Amazon. CloudFront
2. CloudFront leitet die Anfrage an Amazon API Gateway weiter, das als CloudFront Ursprung konfiguriert ist.
3. Ein benutzerdefinierter API Gateway Gateway-Authorizer wird aufgerufen, um das JWT zu verifizieren.
4. Microservices antworten auf die Anfrage.

Ablauf der Autorisierung und API-Zugriffskontrolle (im Diagramm mit roten nummerierten Callouts dargestellt):

1. Das PEP ruft den Autorisierungsdienst auf und leitet die Anforderungsdaten, einschließlich aller JWTs, weiter.

2. Der Autorisierungsdienst (PDP), in diesem Fall Verified Permissions, verwendet die Anforderungsdaten als Abfrageeingabe und bewertet sie auf der Grundlage der in der Abfrage angegebenen relevanten Richtlinien.
3. Die Autorisierungsentscheidung wird an das PEP zurückgesendet und ausgewertet.

Dieses Modell verwendet ein zentralisiertes PDP, um Autorisierungsentscheidungen zu treffen. PEPs werden an verschiedenen Stellen implementiert, um Autorisierungsanfragen an das PDP zu stellen. Das folgende Diagramm zeigt, wie Sie dieses Modell in einer hypothetischen SaaS-Anwendung mit mehreren Mandanten implementieren können.

In dieser Architektur fordern PEPs Autorisierungsentscheidungen an den Service-Endpunkten für Amazon CloudFront und Amazon API Gateway sowie für jeden Microservice an. Die Autorisierungsentscheidung wird vom Autorisierungsdienst Amazon Verified Permissions (PDP) getroffen. Da es sich bei Verified Permissions um einen vollständig verwalteten Service handelt, müssen Sie die zugrunde liegende Infrastruktur nicht verwalten. Sie können mit verifizierten Berechtigungen interagieren, indem Sie eine RESTful-API oder das AWS SDK verwenden.

Sie können diese Architektur auch mit benutzerdefinierten Policy-Engines verwenden. Alle Vorteile, die sich aus verifizierten Berechtigungen ergeben, müssen jedoch durch die Logik ersetzt werden, die von der benutzerdefinierten Policy-Engine bereitgestellt wird.

Ein zentralisiertes PDP mit PEPs auf APIs bietet eine einfache Möglichkeit, ein robustes Autorisierungssystem für APIs zu erstellen. Dies vereinfacht den Autorisierungsprozess und bietet außerdem eine easy-to-use wiederholbare Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder andere Anwendungskomponenten.

Verwenden des Cedar SDK

Amazon Verified Permissions verwendet die Sprache Cedar, um detaillierte Berechtigungen in Ihren benutzerdefinierten Anwendungen zu verwalten. Mit Verified Permissions können Sie Cedar-Richtlinien an einem zentralen Ort speichern, die Vorteile der niedrigen Latenz bei der Verarbeitung im Millisekundenbereich nutzen und Berechtigungen für verschiedene Anwendungen prüfen. Optional können Sie das Cedar SDK auch direkt in Ihre Anwendung integrieren, um Autorisierungsentscheidungen zu treffen, ohne verifizierte Berechtigungen zu verwenden. Diese Option erfordert zusätzliche kundenspezifische Anwendungsentwicklung, um Richtlinien für Ihren Anwendungsfall zu verwalten und zu speichern. Sie kann jedoch eine praktikable Alternative sein, insbesondere in Fällen, in denen der Zugriff auf verifizierte Berechtigungen nur sporadisch oder aufgrund einer inkonsistenten Internetverbindung nicht möglich ist.

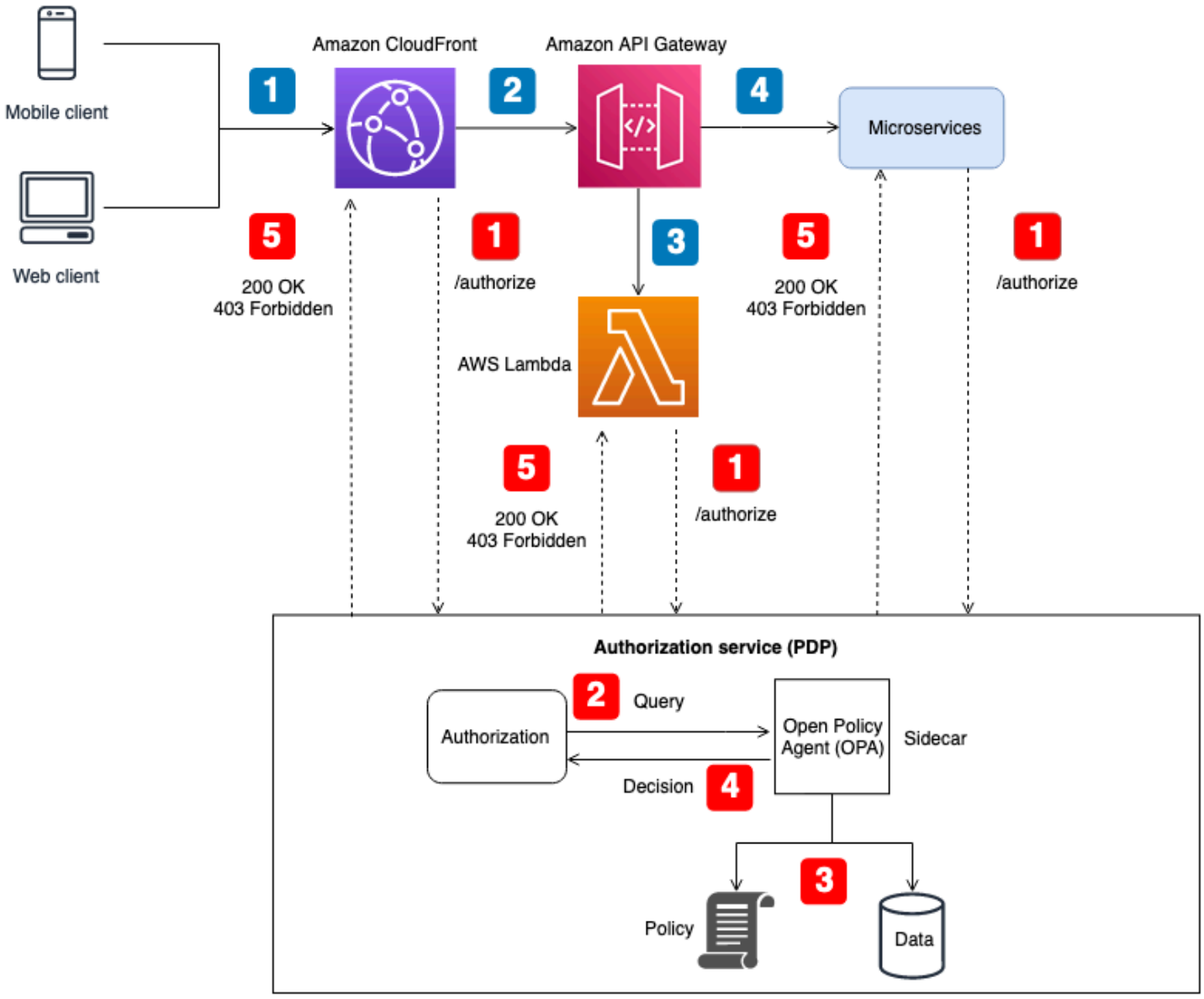
Entwerfen Sie Modelle für OPA

Verwendung eines zentralisierten PDP mit PEPs auf APIs

Das Modell eines zentralisierten Policy-Decision-Punkts (PDP) mit Policy-Enforcement-Punkten (PEPs) auf APIs folgt branchenweit bewährten Verfahren zur Schaffung eines effektiven und einfach zu wartenden Systems für die API-Zugriffskontrolle und -autorisierung. Dieser Ansatz unterstützt mehrere wichtige Prinzipien:

- Autorisierung und API-Zugriffskontrolle werden an mehreren Stellen in der Anwendung angewendet.
- Die Autorisierungslogik ist unabhängig von der Anwendung.
- Entscheidungen zur Zugriffskontrolle sind zentralisiert.

Dieses Modell verwendet ein zentralisiertes PDP, um Autorisierungsentscheidungen zu treffen. PEPs werden an allen APIs implementiert, um Autorisierungsanfragen an das PDP zu stellen. Das folgende Diagramm zeigt, wie Sie dieses Modell in einer hypothetischen SaaS-Anwendung mit mehreren Mandanten implementieren können.



Anwendungsablauf (im Diagramm mit blau nummerierten Callouts dargestellt):

1. Ein authentifizierter Benutzer mit einem JWT generiert eine HTTP-Anfrage an Amazon. CloudFront
2. CloudFront leitet die Anfrage an Amazon API Gateway weiter, das als CloudFront Ursprung konfiguriert ist.
3. Ein benutzerdefinierter API Gateway Gateway-Authorizer wird aufgerufen, um das JWT zu verifizieren.
4. Microservices antworten auf die Anfrage.

Ablauf der Autorisierung und API-Zugriffskontrolle (im Diagramm mit roten nummerierten Callouts dargestellt):

1. Das PEP ruft den Autorisierungsdienst auf und leitet die Anforderungsdaten, einschließlich aller JWTs, weiter.
2. Der Autorisierungsdienst (PDP) verwendet die Anforderungsdaten und fragt eine REST-API des OPA-Agenten ab, die als Sidecar ausgeführt wird. Die Anforderungsdaten dienen als Eingabe für die Abfrage.
3. OPA bewertet die Eingabe auf der Grundlage der in der Abfrage angegebenen relevanten Richtlinien. Daten werden importiert, um bei Bedarf eine Autorisierungsentscheidung zu treffen.
4. OPA gibt eine Entscheidung an den Autorisierungsdienst zurück.
5. Die Autorisierungsentscheidung wird an das PEP zurückgesendet und bewertet.

In dieser Architektur fordern PEPs Autorisierungsentscheidungen an den Service-Endpunkten für Amazon CloudFront und Amazon API Gateway sowie für jeden Microservice an. Die Autorisierungsentscheidung wird von einem Autorisierungsdienst (dem PDP) mit einem OPA-Sidecar getroffen. Sie können diesen Autorisierungsdienst als Container oder als herkömmliche Serverinstanz betreiben. Der OPA-Sidecar stellt seine RESTful-API lokal zur Verfügung, sodass nur der Autorisierungsdienst auf die API zugreifen kann. Der Autorisierungsdienst stellt eine separate API zur Verfügung, die PEPs zur Verfügung steht. Wenn der Autorisierungsdienst als Vermittler zwischen PEPs und OPA fungiert, kann jede erforderliche Transformationslogik zwischen PEPs und OPA eingefügt werden, z. B. wenn die Autorisierungsanfrage eines PEP nicht der von OPA erwarteten Abfrageeingabe entspricht.

Sie können diese Architektur auch mit benutzerdefinierten Policy-Engines verwenden. Alle Vorteile von OPA müssen jedoch durch die Logik ersetzt werden, die von der benutzerdefinierten Policy-Engine bereitgestellt wird.

Ein zentralisiertes PDP mit PEPs auf APIs bietet eine einfache Möglichkeit, ein robustes Autorisierungssystem für APIs zu erstellen. Es ist einfach zu implementieren und bietet außerdem eine easy-to-use wiederholbare Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder andere Anwendungskomponenten. Dieser Ansatz kann jedoch zu viel Latenz in Ihrer Anwendung führen, da Autorisierungsentscheidungen den Aufruf einer separaten API erfordern. Wenn die Netzwerklatenz ein Problem darstellt, könnten Sie ein verteiltes PDP in Betracht ziehen.

Verwenden eines verteilten PDP mit PEPs auf APIs

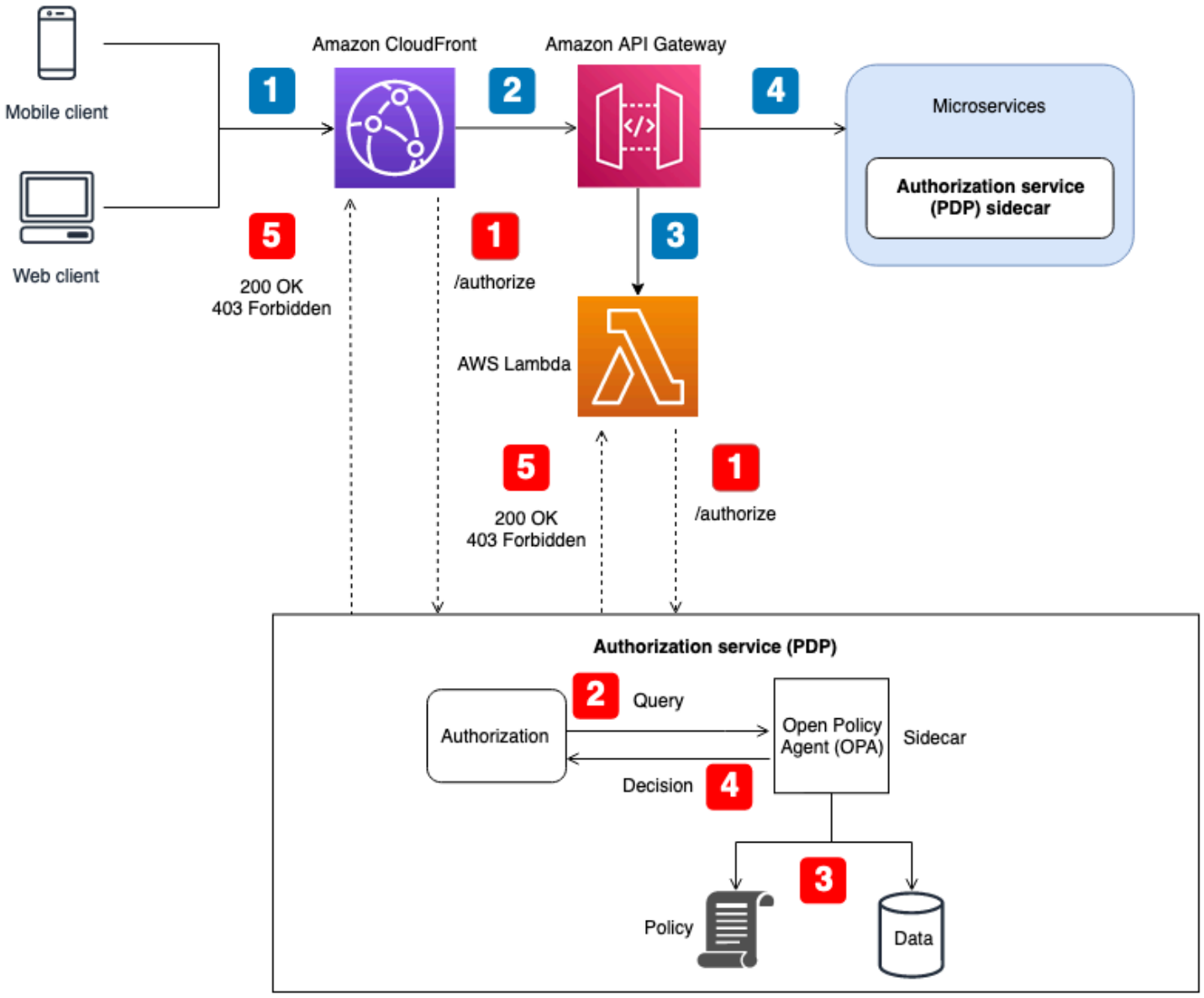
Das Modell des Distributed Policy Decision Point (PDP) mit Policy Enforcement Points (PEPs) auf APIs folgt den branchenweit bewährten Verfahren zur Schaffung eines effektiven Systems für die API-Zugriffskontrolle und -autorisierung. Wie beim Modell des zentralisierten PDP mit PEPs auf APIs unterstützt dieser Ansatz die folgenden Hauptprinzipien:

- Autorisierung und API-Zugriffskontrolle werden an mehreren Stellen in der Anwendung angewendet.
- Die Autorisierungslogik ist unabhängig von der Anwendung.
- Entscheidungen zur Zugriffskontrolle sind zentralisiert.

Sie fragen sich vielleicht, warum Entscheidungen zur Zugriffskontrolle zentralisiert werden, wenn das PDP verteilt wird. Obwohl das PDP möglicherweise an mehreren Stellen in einer Anwendung vorhanden ist, muss es dieselbe Autorisierungslogik verwenden, um Entscheidungen zur Zugriffskontrolle zu treffen. Alle PDPs bieten dieselben Zugriffskontrollentscheidungen bei denselben Eingaben. PEPs werden an allen APIs implementiert, um Autorisierungsanfragen an den PDP zu stellen. Die folgende Abbildung zeigt, wie dieses verteilte Modell in einer hypothetischen SaaS-Anwendung mit mehreren Mandanten implementiert werden kann.

Bei diesem Ansatz werden PDPs an mehreren Stellen in der Anwendung implementiert. Bei Anwendungskomponenten mit integrierten Rechenfunktionen, die OPA ausführen und eine PDP unterstützen, wie z. B. einen containerisierten Service mit Sidecar oder eine Amazon Elastic Compute Cloud (Amazon EC2) -Instance, können PDP-Entscheidungen direkt in die Anwendungskomponente integriert werden, ohne dass ein RESTful-API-Aufruf an einen zentralen PDP-Service getätigt werden muss. Dies hat den Vorteil, dass die Latenz reduziert wird, die beim zentralisierten PDP-Modell auftreten kann, da nicht jede Anwendungskomponente zusätzliche API-Aufrufe tätigen muss, um Autorisierungsentscheidungen zu erhalten. In diesem Modell ist jedoch immer noch ein zentralisiertes PDP für Anwendungskomponenten erforderlich, die nicht über integrierte Rechenfunktionen verfügen, die eine direkte Integration eines PDP ermöglichen, wie z. B. der Amazon CloudFront - oder Amazon API Gateway Gateway-Service.

Das folgende Diagramm zeigt, wie diese Kombination aus einem zentralisierten PDP und einem verteilten PDP in einer hypothetischen SaaS-Anwendung für mehrere Mandanten implementiert werden kann.



Anwendungsablauf (im Diagramm mit blau nummerierten Callouts dargestellt):

1. Ein authentifizierter Benutzer mit einem JWT generiert eine HTTP-Anfrage an Amazon. CloudFront
2. CloudFront leitet die Anfrage an Amazon API Gateway weiter, das als CloudFront Ursprung konfiguriert ist.
3. Ein benutzerdefinierter API Gateway Gateway-Authorizer wird aufgerufen, um das JWT zu verifizieren.
4. Microservices antworten auf die Anfrage.

Ablauf der Autorisierung und API-Zugriffskontrolle (im Diagramm mit roten nummerierten Callouts dargestellt):

1. Das PEP ruft den Autorisierungsdienst auf und leitet die Anforderungsdaten, einschließlich aller JWTs, weiter.
2. Der Autorisierungsdienst (PDP) verwendet die Anforderungsdaten und fragt eine REST-API des OPA-Agenten ab, die als Sidecar ausgeführt wird. Die Anforderungsdaten dienen als Eingabe für die Abfrage.
3. OPA bewertet die Eingabe auf der Grundlage der in der Abfrage angegebenen relevanten Richtlinien. Daten werden importiert, um bei Bedarf eine Autorisierungsentscheidung zu treffen.
4. OPA gibt eine Entscheidung an den Autorisierungsdienst zurück.
5. Die Autorisierungsentscheidung wird an das PEP zurückgesendet und bewertet.

In dieser Architektur fordern PEPs Autorisierungsentscheidungen an den Service-Endpunkten für ein API Gateway CloudFront und für jeden Microservice an. Die Autorisierungsentscheidung für Microservices wird von einem Autorisierungsdienst (dem PDP) getroffen, der als Sidecar mit der Anwendungskomponente fungiert. Dieses Modell ist für Microservices (oder Services) möglich, die auf Containern oder Amazon Elastic Compute Cloud (Amazon EC2) -Instances ausgeführt werden. Autorisierungsentscheidungen für Dienste wie API Gateway CloudFront würden immer noch die Kontaktaufnahme mit einem externen Autorisierungsdienst erfordern. Unabhängig davon stellt der Autorisierungsdienst eine API zur Verfügung, die PEPs zur Verfügung steht. Wenn der Autorisierungsdienst als Vermittler zwischen PEPs und OPA fungiert, kann jede Transformationslogik zwischen PEPs und OPA eingefügt werden, die erforderlich sein könnte, z. B. wenn die Autorisierungsanfrage eines PEP nicht der von OPA erwarteten Abfrageeingabe entspricht.

Sie können diese Architektur auch mit benutzerdefinierten Policy-Engines verwenden. Alle Vorteile von OPA müssen jedoch durch die Logik ersetzt werden, die von der benutzerdefinierten Policy-Engine bereitgestellt wird.

Ein verteiltes PDP mit PEPs auf APIs bietet die Möglichkeit, ein robustes Autorisierungssystem für APIs zu erstellen. Es ist einfach zu implementieren und bietet eine easy-to-use wiederholbare Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder andere Anwendungskomponenten. Dieser Ansatz hat auch den Vorteil, dass die Latenz reduziert wird, die beim zentralisierten PDP-Modell auftreten kann.

Verwenden eines verteilten PDP als Bibliothek

Sie können Autorisierungsentscheidungen auch von einem PDP anfordern, das als Bibliothek oder Paket zur Verwendung in einer Anwendung zur Verfügung gestellt wird. OPA kann als Go-Bibliothek eines Drittanbieters verwendet werden. Für andere Programmiersprachen bedeutet die Übernahme dieses Modells im Allgemeinen, dass Sie eine benutzerdefinierte Policy-Engine erstellen müssen.

Überlegungen zum Mehrmandanten-Design von Amazon Verified Permissions

Bei der Implementierung der Autorisierung mithilfe von Amazon Verified Permissions in einer SaaS-Lösung mit mehreren Mandanten sind mehrere Designoptionen zu berücksichtigen. Bevor wir uns mit diesen Optionen befassen, sollten wir den Unterschied zwischen Isolierung und Autorisierung in einem SaaS-Kontext mit mehreren Mandanten klären. Durch die Isolierung eines Mandanten wird verhindert, dass eingehende und ausgehende Daten dem falschen Mandanten zugänglich gemacht werden. Durch die Autorisierung wird sichergestellt, dass ein Benutzer über die erforderlichen Berechtigungen für den Zugriff auf einen Mandanten verfügt.

Unter Verifizierte Berechtigungen werden Richtlinien in einem Richtlinienspeicher gespeichert. Wie in der Dokumentation Verifizierte Berechtigungen beschrieben, können Sie entweder die Richtlinien von Mandanten isolieren, indem Sie für jeden Mandanten einen eigenen Richtlinienspeicher verwenden, oder Mandanten die gemeinsame Nutzung von Richtlinien ermöglichen, indem Sie einen einzigen Richtlinienspeicher für alle Mandanten verwenden. In diesem Abschnitt werden die Vor- und Nachteile dieser beiden Isolationsstrategien beschrieben und es wird beschrieben, wie sie mithilfe eines mehrstufigen Bereitstellungsmodells eingesetzt werden können. Weitere Informationen finden Sie in der Dokumentation Verifizierte Berechtigungen.

Obwohl sich die in diesem Abschnitt erörterten Kriterien auf verifizierte Berechtigungen konzentrieren, basieren die allgemeinen Konzepte auf der isolierten Denkweise und den damit verbundenen Leitlinien. SaaS-Anwendungen müssen die Mandantenisolierung immer als Teil ihres Designs berücksichtigen, und dieses allgemeine Prinzip der Isolierung erstreckt sich auch auf die Einbeziehung verifizierter Berechtigungen in eine SaaS-Anwendung. In diesem Abschnitt wird auch auf zentrale SaaS-Isolationsmodelle wie das isolierte SaaS-Modell und das gepoolte SaaS-Modell verwiesen. Weitere Informationen finden Sie in den Kernkonzepten zur Isolierung im AWS Well-Architected Framework, SaaS Lens.

Die wichtigsten Überlegungen bei der Entwicklung von SaaS-Lösungen für mehrere Mandanten sind die Isolierung von Mandanten und das Onboarding von Mandanten. Die Isolierung von Mandanten wirkt sich auf Sicherheit, Datenschutz, Belastbarkeit und Leistung aus. Das Onboarding von Mandanten wirkt sich auf Ihre betrieblichen Prozesse aus, da es sich auf den betrieblichen Aufwand und die Beobachtbarkeit bezieht. Organizations, die sich auf eine SaaS-Reise begeben oder Multi-Tenant-Lösungen implementieren, müssen immer priorisieren, wie die SaaS-Anwendung mit dem Mietverhältnis umgeht. Obwohl sich eine SaaS-Lösung möglicherweise auf ein bestimmtes

Isolationsmodell stützt, ist Konsistenz nicht unbedingt für die gesamte SaaS-Lösung erforderlich. Beispielsweise ist das Isolationsmodell, das Sie für die Frontend-Komponenten Ihrer Anwendung wählen, möglicherweise nicht dasselbe wie das Isolationsmodell, das Sie für einen Microservice oder Autorisierungsdienste wählen.

Überlegungen zum Design:

- Onboarding von Mandanten und Registrierung von Benutzern
- Richtlinienspeicher pro Mandant
- Ein gemeinsam genutzter Richtlinienspeicher für mehrere Mandanten
- Mehrstufiges Bereitstellungsmodell

Onboarding von Mandanten und Registrierung von Benutzern

SaaS-Anwendungen folgen dem Konzept der SaaS-Identitäten und folgen der allgemeinen bewährten Methode, eine Benutzeridentität an eine Mandantenidentität zu binden. Beim Binden wird eine Mandanten-ID als Anspruch oder Attribut für den Benutzer im Identitätsanbieter gespeichert. Dadurch wird die Verantwortung für die Zuordnung von Identitäten zu Mandanten von jeder Anwendung auf den Benutzerregistrierungsprozess verlagert. Jeder authentifizierte Benutzer hat dann die richtige Mandantenidentität als Teil des JSON Web Tokens (JWT).

Ebenso sollte die Auswahl des richtigen Richtlinienspeichers für eine Autorisierungsanfrage nicht durch die Anwendungslogik bestimmt werden. Um zu bestimmen, welcher Richtlinienspeicher für eine bestimmte Autorisierungsanfrage verwendet werden soll, sollten Sie eine Zuordnung von Benutzern zu Richtlinienspeichern oder Mandanten zu Richtlinienspeichern verwalten. Diese Zuordnungen werden normalerweise in einem Datenspeicher wie Amazon DynamoDB oder Amazon Relational Database Service (Amazon RDS) verwaltet, auf den Ihre Anwendung verweist. Sie können diese Zuordnungen auch durch Daten in einem Identity Provider (IdP) bereitstellen oder ergänzen. Die Beziehung zwischen Mandanten, Benutzern und Richtlinienspeichern wird einem Benutzer dann in der Regel über ein JWT bereitgestellt, das alle Beziehungen enthält, die für eine Autorisierungsanfrage erforderlich sind.

Dieses Beispiel zeigt, wie das JWT für den Benutzer aussehen könnte`Alice`, der dem Mandanten angehört `TenantA` und den Richtlinienspeicher mit der Richtlinienspeicher-ID `ps-43214321` für die Autorisierung verwendet.

```
{
```

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

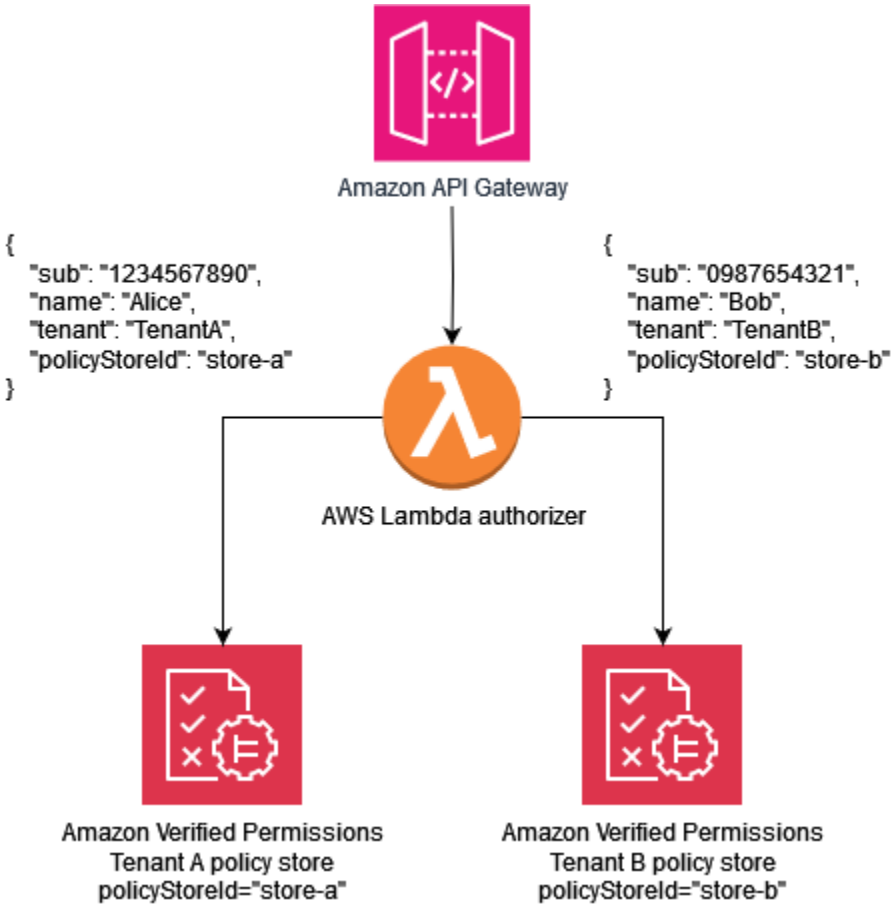
Richtlinienspeicher pro Mandant

Das Entwurfsmodell pro Mandant in Amazon Verified Permissions ordnet jedem Mandanten in einer SaaS-Anwendung seinen eigenen Richtlinienspeicher zu. Dieses Modell ähnelt dem SaaS-Silo-Isolationsmodell. Beide Modelle erfordern die Schaffung einer mieterspezifischen Infrastruktur und haben ähnliche Vor- und Nachteile. Die Hauptvorteile dieses Ansatzes sind die durch die Infrastruktur erzwungene Mandantenisolierung, die Unterstützung einzigartiger Autorisierungsmodelle pro Mandant, die Beseitigung von Problemen mit störenden Nachbarn und ein geringerer Umfang der Auswirkungen von Fehlern bei Richtlinienaktualisierungen oder -bereitstellungen. Zu den Nachteilen dieses Ansatzes gehören komplexere Onboarding-Prozesse, Bereitstellungen und Betriebsabläufe für Mandanten. Ein mandantenspezifischer Richtlinienspeicher ist der empfohlene Ansatz, wenn die Lösung über eigene Richtlinien pro Mandant verfügt.

Das Policy-Store-Modell pro Mandant kann einen stark isolierten Ansatz zur Mandantenisolierung bieten, falls Ihre SaaS-Anwendung dies erfordert. Sie können dieses Modell auch mit Poolisolierung verwenden, aber Ihre Implementierung von Verified Permissions bietet nicht die Standardvorteile des umfassenderen Poolisolationsmodells, wie z. B. vereinfachte Verwaltung und Betrieb.

In einem mandantenspezifischen Richtlinienspeicher wird die Mandantenisolierung erreicht, indem die Richtlinienspeicher-ID eines Mandanten während des Benutzerregistrierungsprozesses der SaaS-Identität des Benutzers zugeordnet wird, wie bereits erwähnt. Dieser Ansatz verknüpft den Richtlinienspeicher des Mandanten stark mit dem Benutzerprinzipal und bietet eine konsistente Möglichkeit, das Mapping in der gesamten SaaS-Lösung gemeinsam zu nutzen. Sie können das Mapping einer SaaS-Anwendung bereitstellen, indem Sie es als Teil eines IdP oder in einer externen Datenquelle wie DynamoDB verwalten. Dadurch wird auch sichergestellt, dass der Principal Teil des Mandanten ist und dass der Richtlinienspeicher des Mandanten verwendet wird.

Dieses Beispiel zeigt, wie das JWT, das das `policyStoreId` und `tenant` eines Benutzers enthält, von einem API-Endpunkt an den Richtlinienbewertungspunkt in einem AWS Lambda Autorisierer übergeben wird, der die Anforderung an den richtigen Richtlinienspeicher weiterleitet.



Die folgende Beispielrichtlinie veranschaulicht das Paradigma der Gestaltung eines Policy-Stores pro Mandant. Der Benutzer `Alice` gehört zu `TenantA`. The, policyStoreId `store-a` ist auch der Mandantenidentität von zugeordnet `Alice`, und erzwingt die Verwendung des richtigen Richtlinienspeichers. Dadurch wird sichergestellt, dass die Richtlinien von verwendet `TenantA` werden.

> **Note**
>
> Das Modell des Richtlinienspeichers pro Mandant isoliert die Richtlinien der Mandanten. Die Autorisierung erzwingt die Aktionen, die Benutzer mit ihren Daten ausführen dürfen. Die Ressourcen jeder hypothetischen Anwendung, die dieses Modell verwendet, sollten mithilfe anderer Isolationsmechanismen isoliert werden, wie in der Dokumentation AWS Well-Architected Framework, SaaS Lens definiert.

In dieser Richtlinie `Alice` ist er berechtigt, die Daten aller Ressourcen einzusehen.

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

Um eine Autorisierungsanfrage zu stellen und eine Evaluierung mit einer Richtlinie für verifizierte Berechtigungen zu starten, müssen Sie die ID des Richtlinienspeichers angeben, die der eindeutigen ID entspricht, `store-a` die dem Mandanten zugeordnet ist.

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

```
            }
         ]
      ]
   }
}
```

Der Benutzer Bob gehört zu Mandant B und policyStoreId `store-b` ist auch der Mandantenidentität von zugeordnetBob, wodurch die Verwendung des richtigen Richtlinienspeichers erzwungen wird. Dadurch wird sichergestellt, dass die Richtlinien von Mandant B verwendet werden.

In dieser Richtlinie Bob ist er berechtigt, die Daten aller Ressourcen anzupassen. In diesem Beispiel `customizeData` könnte es sich um eine Aktion handeln, die nur für Mandant B spezifisch ist, sodass die Richtlinie nur für Mandant B gilt. Das mandantenspezifische Richtlinienspeichermodell unterstützt grundsätzlich benutzerdefinierte Richtlinien pro Mandant.

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

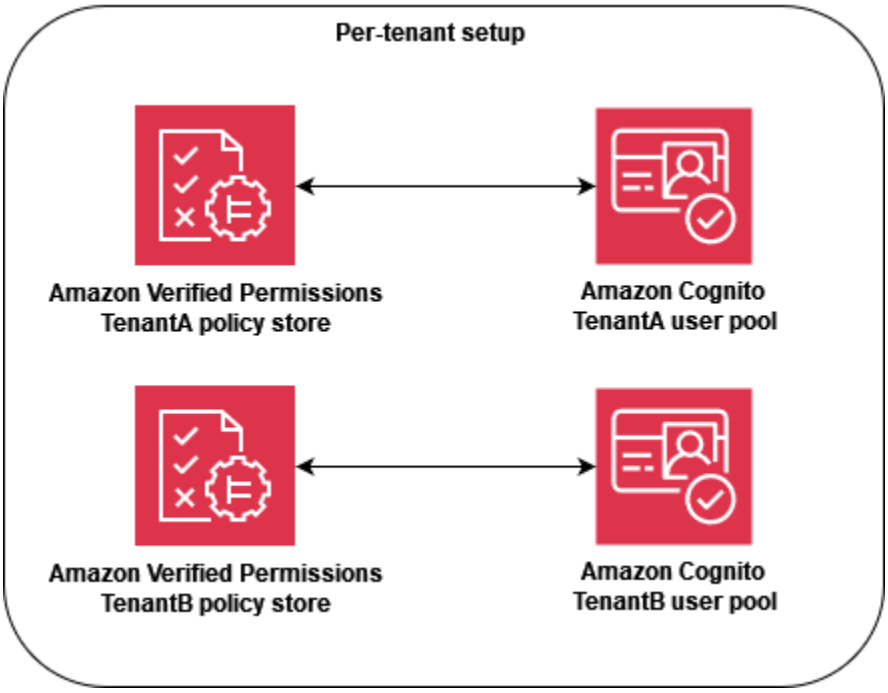
Um eine Autorisierungsanfrage zu stellen und eine Evaluierung mit einer Richtlinie für verifizierte Berechtigungen zu starten, müssen Sie die ID des Richtlinienspeichers angeben, die der eindeutigen ID entspricht, die dem Mandanten zugeordnet ist. `store-b`

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

Mit verifizierten Berechtigungen ist es möglich, aber nicht erforderlich, einen IdP in einen Richtlinienspeicher zu integrieren. Diese Integration ermöglicht es Richtlinien, den Prinzipal im Identitätsspeicher explizit als Prinzipal der Richtlinien zu referenzieren. Weitere Informationen zur Integration mit Amazon Cognito als IdP für verifizierte Berechtigungen finden Sie in der Dokumentation Verified Permissions und in der Amazon Cognito Cognito-Dokumentation.

Wenn Sie einen Richtlinienspeicher in einen IdP integrieren, können Sie nur eine Identitätsquelle pro Richtlinienspeicher verwenden. Wenn Sie sich beispielsweise dafür entscheiden, Verified Permissions in Amazon Cognito zu integrieren, müssen Sie die Strategie widerspiegeln, die für die Mandantenisolierung von Richtlinienspeichern verifizierter Berechtigungen und Amazon Cognito Cognito-Benutzerpools verwendet wird. Die Richtlinienspeicher und Benutzerpools müssen sich ebenfalls im selben System befinden. AWS-Konto



Auf betrieblicher Ebene bietet der Policy-Store pro Mandant einen Audit-Vorteil, da Sie die protokollierten Aktivitäten problemlos AWS CloudTrail unabhängig für jeden Mandanten abfragen können. Wir empfehlen jedoch weiterhin, zusätzliche benutzerdefinierte Metriken für eine Dimension pro Mandant bei Amazon CloudWatch zu protokollieren.

Beim Policy-Store-Ansatz pro Mandant müssen außerdem zwei Kontingente für verifizierte Berechtigungen genau beachtet werden, um sicherzustellen, dass sie den Betrieb Ihrer SaaS-Lösung nicht beeinträchtigen. Bei diesen Kontingenten handelt es sich um Policy-Stores pro Region pro Konto und IsAuthorized Anfragen pro Sekunde pro Region pro Konto. Sie können Erhöhungen für beide Kontingente beantragen.

Ein detaillierteres Beispiel für die Implementierung des Policy-Store-Modells pro Mandant finden Sie im AWS Blogbeitrag SaaS-Zugriffskontrolle mithilfe von Amazon Verified Permissions mit einem Policy-Store pro Mandant.

Ein gemeinsam genutzter Richtlinienspeicher für mehrere Mandanten

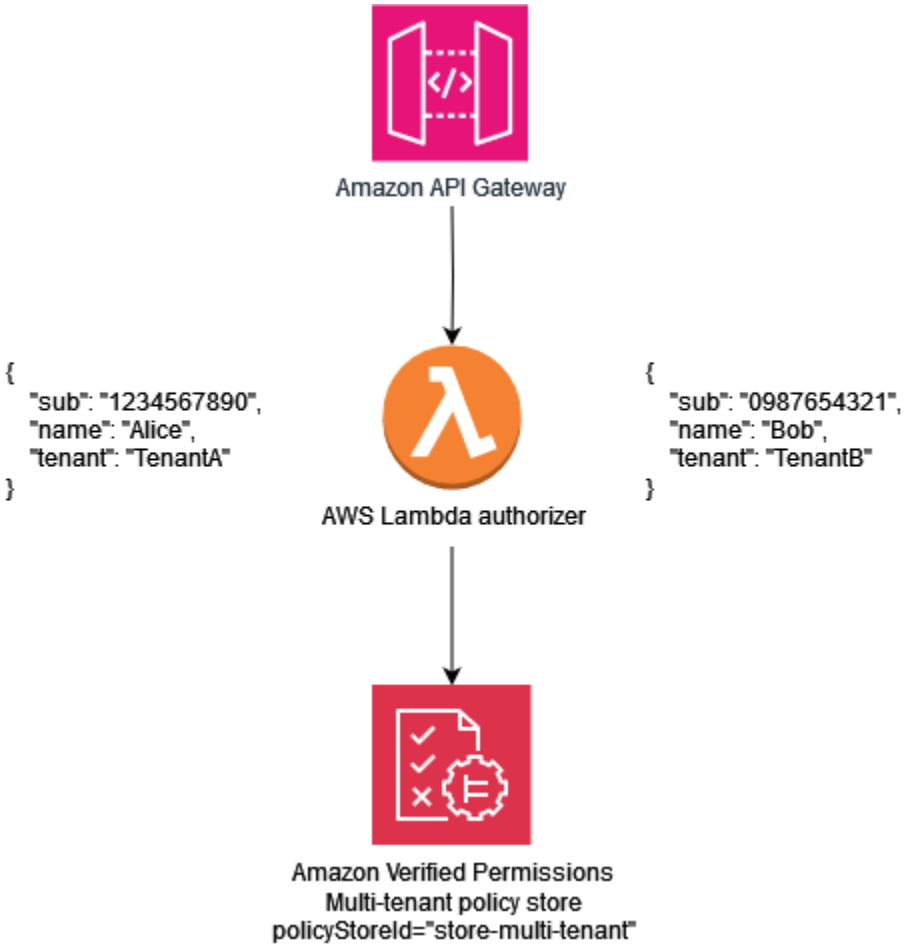
Das Designmodell für einen gemeinsam genutzten Mehrmandanten-Richtlinienspeicher verwendet einen einzigen Mehrmandanten-Richtlinienspeicher in Amazon Verified Permissions für alle Mandanten in der SaaS-Lösung. Der Hauptvorteil dieses Ansatzes ist die vereinfachte Verwaltung

und der Betrieb, insbesondere weil Sie beim Onboarding von Mandanten keine zusätzlichen Policy-Stores erstellen müssen. Zu den Nachteilen dieses Ansatzes gehören ein größeres Ausmaß an Auswirkungen von Ausfällen oder Fehlern bei Richtlinienaktualisierungen oder -bereitstellungen sowie die größere Gefahr von Nebengeräuschen. Darüber hinaus empfehlen wir diesen Ansatz nicht, wenn Ihre Lösung individuelle Richtlinien für jeden Mandanten erfordert. Verwenden Sie in diesem Fall stattdessen das Policy-Store-Modell pro Mandant, um sicherzustellen, dass die Richtlinien des richtigen Mandanten verwendet werden.

Der Ansatz eines gemeinsam genutzten Policy-Speichers für mehrere Mandanten ähnelt dem SaaS-Pool-Isolationsmodell. Es kann einen gebündelten Ansatz zur Mandantenisolierung bieten, falls Ihre SaaS-Anwendung dies erfordert. Sie können dieses Modell auch verwenden, wenn Ihre SaaS-Lösung isolierte Isolierung auf ihre Microservices anwendet. Wenn Sie sich für ein Modell entscheiden, sollten Sie die Anforderungen an die Isolierung von Mandantendaten und die Struktur der Richtlinien für verifizierte Berechtigungen, die für eine SaaS-Anwendung erforderlich sind, unabhängig voneinander bewerten.

Um eine einheitliche Art der gemeinsamen Nutzung der Mandanten-ID in Ihrer gesamten SaaS-Lösung durchzusetzen, empfiehlt es sich, die Kennung bei der Benutzerregistrierung der SaaS-Identität des Benutzers zuzuordnen, wie bereits beschrieben. Sie können diese Zuordnung für eine SaaS-Anwendung bereitstellen, indem Sie sie als Teil eines IdP oder in einer externen Datenquelle wie DynamoDB verwalten. Wir empfehlen außerdem, die ID des gemeinsamen Richtlinienspeichers Benutzern zuzuordnen. Obwohl die ID nicht im Rahmen der Mandantenisolierung verwendet wird, ist dies eine bewährte Methode, da sie future Änderungen erleichtert.

Das folgende Beispiel zeigt, wie der API-Endpunkt ein JWT für die Benutzer `Alice` und sendetBob, die verschiedenen Mandanten angehören, sich aber den Richtlinienspeicher mit der Richtlinienspeicher-ID `store-multi-tenant` für die Autorisierung teilen. Da sich alle Mandanten einen einzigen Richtlinienspeicher teilen, müssen Sie die Richtlinienspeicher-ID nicht in einem Token oder einer Datenbank verwalten. Da sich alle Mandanten eine einzige Richtlinienspeicher-ID teilen, können Sie die ID als Umgebungsvariable angeben, mit der Ihre Anwendung den Richtlinienspeicher aufrufen kann.



Die folgende Beispielrichtlinie veranschaulicht das Paradigma des gemeinsamen Richtlinienentwurfs für mehrere Mandanten. In dieser Richtlinie ist der Prinzipal, dem `MultiTenantApp::User` das übergeordnete Objekt zugewiesen `MultiTenantApp::Role Admin` ist, berechtigt, die Daten aller Ressourcen einzusehen.

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

Da ein einziger Richtlinienspeicher verwendet wird, muss der Richtlinienspeicher Verified Permissions sicherstellen, dass ein Mandantenattribut, das dem Prinzipal zugeordnet ist, mit dem Mandantenattribut übereinstimmt, das der Ressource zugeordnet ist. Dies kann erreicht werden, indem die folgende Richtlinie in den Richtlinienspeicher aufgenommen wird, um sicherzustellen, dass

alle Autorisierungsanfragen, die keine übereinstimmenden Mandantenattribute für die Ressource und den Prinzipal haben, abgelehnt werden.

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

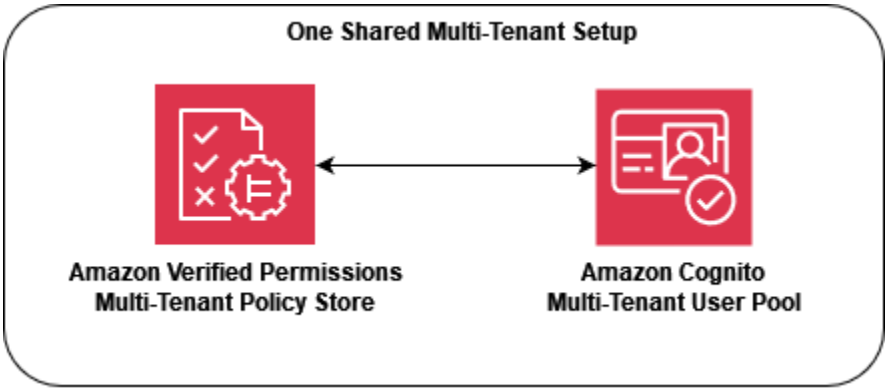
Bei einer Autorisierungsanfrage, die ein Modell mit einem gemeinsamen Richtlinienspeicher für mehrere Mandanten verwendet, ist die ID des Richtlinienspeichers die ID des gemeinsam genutzten Richtlinienspeichers. In der folgenden Anfrage `User Alice` wird der Zugriff gewährt, da sie einen Wert `Role` von `Admin` hat und die der Ressource und dem Prinzipal zugewiesenen `Tenant` Attribute beide `TenantA` sind.

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

Mit verifizierten Berechtigungen ist es möglich, aber nicht erforderlich, einen IdP in einen Richtlinienspeicher zu integrieren. Diese Integration ermöglicht es Richtlinien, den Prinzipal im Identitätsspeicher explizit als Prinzipal der Richtlinien zu referenzieren. Weitere Informationen zur Integration mit Amazon Cognito als IdP für verifizierte Berechtigungen finden Sie in der Dokumentation Verified Permissions und in der Amazon Cognito Cognito-Dokumentation.

Wenn Sie einen Richtlinienspeicher in einen IdP integrieren, können Sie nur eine Identitätsquelle pro Richtlinienspeicher verwenden. Wenn Sie sich beispielsweise dafür entscheiden, Verified Permissions in Amazon Cognito zu integrieren, müssen Sie die Strategie widerspiegeln, die für die Mandantenisolierung von Richtlinienspeichern verifizierter Berechtigungen und Amazon Cognito Cognito-Benutzerpools verwendet wird. Die Richtlinienspeicher und Benutzerpools müssen sich ebenfalls im selben System befinden. AWS-Konto



Aus betrieblicher und prüferischer Sicht hat das Modell mit einem gemeinsamen Richtlinienspeicher für mehrere Mandanten den Nachteil, dass für die angemeldete Aktivität aufwändigere Abfragen AWS CloudTrail erforderlich sind, um einzelne Aktivitäten des Mandanten herauszufiltern, da für jeden protokollierten CloudTrail Anruf derselbe Richtlinienspeicher verwendet wird. In diesem Szenario ist es hilfreich, zusätzliche benutzerdefinierte Metriken für eine Dimension pro Mandant bei Amazon zu protokollieren, CloudWatch um ein angemessenes Maß an Beobachtbarkeit und Auditfähigkeit sicherzustellen.

Der Ansatz eines gemeinsamen Policy-Stores für mehrere Mandanten erfordert auch eine genaue Beachtung der Kontingente für verifizierte Berechtigungen, um sicherzustellen, dass sie den Betrieb Ihrer SaaS-Lösung nicht beeinträchtigen. Insbesondere empfehlen wir Ihnen, die IsAuthorized Anfragen pro Sekunde, Region und Kontingent zu überwachen, um sicherzustellen, dass die Beschränkungen nicht überschritten werden. Sie können eine Erhöhung dieses Kontingents beantragen.

Mehrstufiges Bereitstellungsmodell

Durch die Erstellung eines gestaffelten Bereitstellungsmodells können Sie Mandanten mit hoher Priorität von der potenziell größeren Anzahl von Kunden mit „Standard Tier“ isolieren. Bei diesem Modell können Sie alle Änderungen, die an den Richtlinien vorgenommen wurden, in den Policy-Stores für jede Stufe separat bereitstellen. Dadurch wird jede Kundenstufe von Änderungen isoliert,

die außerhalb ihrer Stufe vorgenommen wurden. Beim Modell der mehrstufigen Bereitstellung werden die Richtlinienspeicher in der Regel als Teil der anfänglichen Infrastrukturbereitstellung für jede Stufe erstellt, anstatt sie beim Onboarding eines Mandanten bereitzustellen.

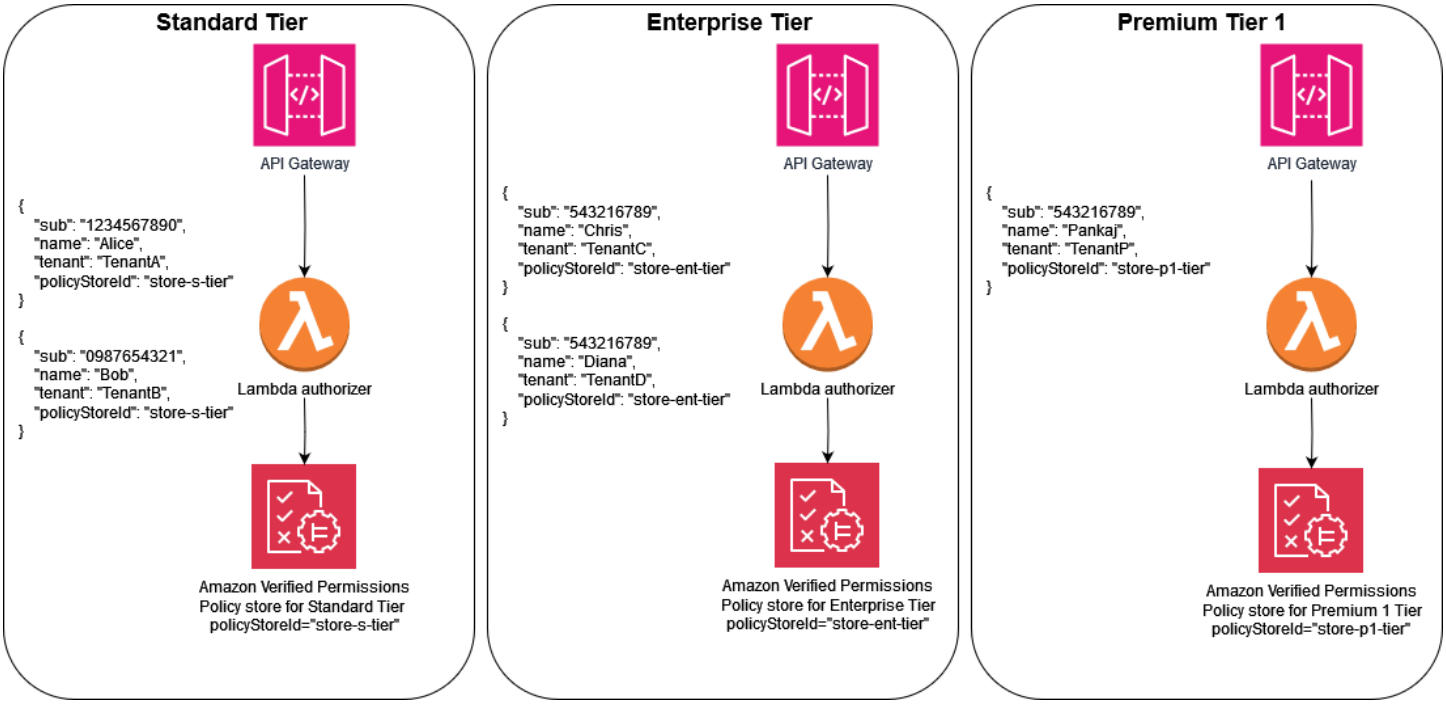
Wenn Ihre Lösung hauptsächlich ein isoliertes Poolmodell verwendet, ist möglicherweise eine zusätzliche Isolierung oder Anpassung erforderlich. Sie können beispielsweise ein „Premium-Tier“ erstellen, bei dem jeder Mandant seine eigene Infrastruktur auf Mandantenebene erhält. Dadurch entsteht ein isoliertes Modell, indem eine gepoolte Instanz mit nur einem Mandanten bereitgestellt wird. Dies könnte die Form von Infrastrukturen wie „Premium Tier Tenant A“ und „Premium Tier Tenant B“ annehmen, die vollständig voneinander getrennt sind, einschließlich Policystores. Dieser Ansatz führt zu einem isolierten Isolationsmodell für Kunden auf höchstem Niveau.

Beim mehrstufigen Bereitstellungsmodell sollte jeder Richtlinienspeicher demselben Isolationsmodell folgen, obwohl es separat bereitgestellt wird. Da mehrere Richtlinienspeicher verwendet werden, müssen Sie in der gesamten SaaS-Lösung eine einheitliche Methode zur gemeinsamen Nutzung der dem Mandanten zugewiesenen Richtlinienspeicher-ID durchsetzen. Wie beim Policy-Store-Modell pro Mandant empfiehlt es sich, die Mandanten-ID bei der Benutzerregistrierung der SaaS-Identität des Benutzers zuzuordnen.

Das folgende Diagramm zeigt drei Stufen: `Standard Tier`, `Enterprise Tier`, und `Premium Tier 1`. Jede Stufe wird separat in ihrer eigenen Infrastruktur bereitgestellt und verwendet einen gemeinsamen Richtlinienspeicher innerhalb der Stufe. Die Stufen Standard und Enterprise enthalten mehrere Mandanten. `TenantA` und `TenantB` gehören zur Stufe `Standard Tier`, und `TenantC` und `TenantD` gehören zur Enterprise-Stufe.

`Premium Tier 1` enthält nur `TenantP`, sodass Sie den Premium-Mandanten so bedienen können, als ob die Lösung über ein vollständig isoliertes Isolationsmodell verfügte, und Funktionen wie benutzerdefinierte Richtlinien bereitstellen können. Die Aufnahme eines neuen Premiumkunden würde zur Schaffung einer Infrastruktur führen. `Premium Tier 2`

> **Note**
>
> Die Anwendung, die Bereitstellung und das Onboarding von Mandanten im Premium-Tarif sind identisch mit den Tarifen Standard und Enterprise. Der einzige Unterschied besteht darin, dass der Onboarding-Workflow der Premium-Stufe mit der Bereitstellung einer neuen Tier-Infrastruktur beginnt.


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"

Überlegungen zum Mehrmandanten-Design von OPA

Der Open Policy Agent (OPA) ist ein flexibler Dienst, der auf zahlreiche Anwendungsfälle angewendet werden kann, in denen Anwendungen Richtlinien- und Autorisierungsentscheidungen treffen müssen. Die Verwendung von OPA mit mehrinstanzenfähigen SaaS-Anwendungen erfordert die Berücksichtigung einzigartiger Kriterien, um sicherzustellen, dass wichtige bewährte SaaS-Methoden wie die Mandantenisolierung Teil der OPA-Implementierung bleiben. Zu diesen Kriterien gehören OPA-Bereitstellungsmuster, Mandantenisolierung und das OPA-Dokumentenmodell sowie das Onboarding von Mandanten. Jedes dieser Kriterien wirkt sich auf das optimale Design für OPA aus, da es sich um Anwendungen mit mehreren Mandanten handelt.

Obwohl sich die Diskussion in diesem Abschnitt auf OPA konzentriert, basieren die allgemeinen Konzepte auf der isolierten Denkweise und den damit verbundenen Leitlinien. SaaS-Anwendungen müssen die Mandantenisolierung immer als Teil ihres Designs berücksichtigen, und dieses allgemeine Prinzip der Isolierung erstreckt sich auch auf die Einbeziehung von OPA in eine SaaS-Anwendung. OPA kann, wenn es richtig eingesetzt wird, ein wichtiger Bestandteil der Durchsetzung der Isolation in SaaS-Anwendungen sein. In diesem Abschnitt wird auch auf zentrale SaaS-Isolationsmodelle wie das isolierte SaaS-Modell und das gepoolte SaaS-Modell verwiesen. Weitere Informationen finden Sie in den Kernkonzepten zur Isolierung im AWS Well-Architected Framework, SaaS Lens.

Überlegungen zum Design:

- Vergleich von zentralisierten und verteilten Bereitstellungsmustern
- Mandantenisolierung mit dem OPA-Dokumentenmodell
- Onboarding von Mandanten

Vergleich von zentralisierten und verteilten Bereitstellungsmustern

Sie können OPA in einem zentralisierten oder verteilten Bereitstellungsmuster bereitstellen, und die ideale Methode für eine Mehrmandantenanwendung hängt vom Anwendungsfall ab. Beispiele für diese Muster finden Sie in den Abschnitten Verwenden einer zentralisierten PDP mit PEPs auf APIs und Verwenden einer verteilten PDP und PEPs auf APIs weiter oben in diesem Handbuch. Da OPA als Daemon in einem Betriebssystem oder Container bereitgestellt werden kann, kann es auf verschiedene Arten implementiert werden, um eine Mehrmandantenanwendung zu unterstützen.

In einem zentralisierten Bereitstellungsmuster wird OPA als Container oder Daemon bereitgestellt, wobei seine RESTful-API für andere Dienste in der Anwendung verfügbar ist. Wenn für einen Dienst eine Entscheidung von OPA erforderlich ist, wird die zentrale OPA-RESTful-API aufgerufen, um diese Entscheidung zu treffen. Dieser Ansatz ist einfach zu implementieren und zu verwalten, da es nur eine einzige Bereitstellung von OPA gibt. Der Nachteil dieses Ansatzes besteht darin, dass er keinen Mechanismus zur Aufrechterhaltung der Trennung von Mandantendaten bietet. Da es nur eine einzige Bereitstellung von OPA gibt, müssen alle Mandantendaten, die in einer OPA-Entscheidung verwendet werden, einschließlich aller externen Daten, auf die OPA verweist, für OPA verfügbar sein. Mit diesem Ansatz können Sie die Isolierung der Mandantendaten aufrechterhalten, dies muss jedoch durch die Richtlinien- und Dokumentenstruktur von OPA oder den Zugriff auf externe Daten durchgesetzt werden. Ein zentralisiertes Bereitstellungsmuster erfordert auch eine höhere Latenz, da bei jeder Autorisierungsentscheidung ein RESTful-API-Aufruf an einen anderen Dienst erfolgen muss.

In einem verteilten Bereitstellungsmuster wird OPA als Container oder Daemon zusammen mit den Diensten der Mehrmandantenanwendung bereitgestellt. Es kann als Sidecar-Container oder als Daemon bereitgestellt werden, der lokal auf dem Betriebssystem ausgeführt wird. Um eine Entscheidung von OPA abzurufen, führt der Dienst einen RESTful-API-Aufruf an die lokale OPA-Bereitstellung durch. (Da OPA als Go-Paket bereitgestellt werden kann, können Sie Go nativ verwenden, um eine Entscheidung abzurufen, anstatt einen RESTful-API-Aufruf zu verwenden.) Im Gegensatz zum zentralisierten Bereitstellungsmuster erfordert das verteilte Muster einen viel größeren Aufwand bei der Bereitstellung, Wartung und Aktualisierung, da es in mehreren Bereichen der Anwendung vorhanden ist. Ein Vorteil des verteilten Bereitstellungsmusters ist die Möglichkeit, die Isolierung von Mandantendaten aufrechtzuerhalten, insbesondere für Anwendungen, die ein isoliertes SaaS-Modell verwenden. Mandantenspezifische Daten können in OPA-Bereitstellungen isoliert werden, die für diesen Mandanten spezifisch sind, da OPA in einem verteilten Modell zusammen mit dem Mandanten bereitgestellt wird. Darüber hinaus weist ein verteiltes Bereitstellungsmuster eine wesentlich geringere Latenz auf als ein zentralisiertes Bereitstellungsmuster, da jede Autorisierungsentscheidung lokal getroffen werden kann.

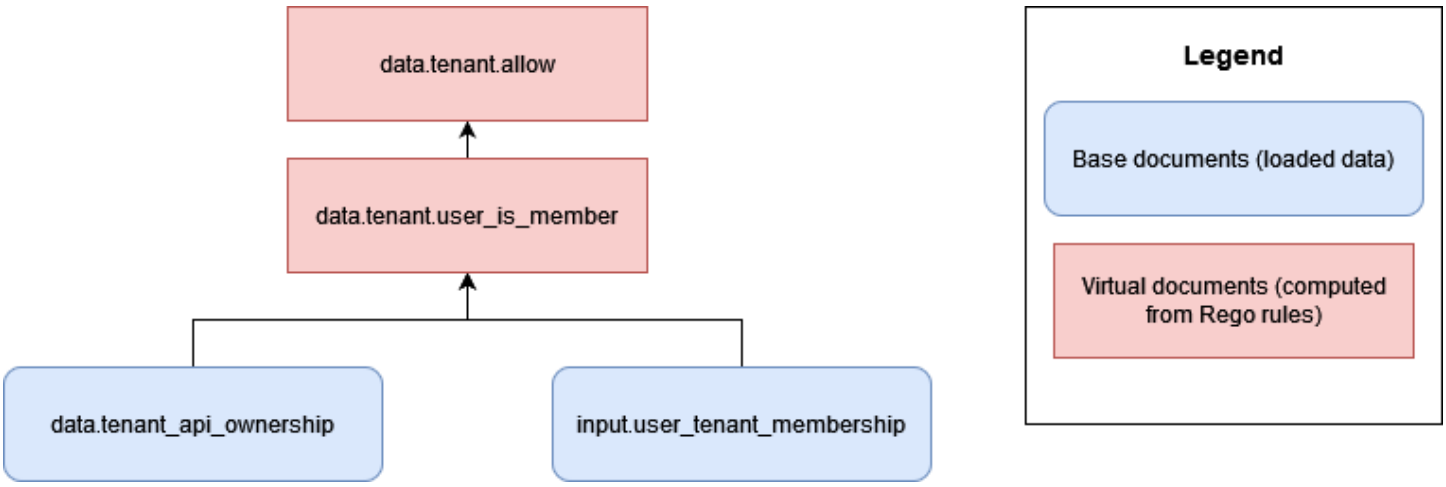
Achten Sie bei der Auswahl eines OPA-Bereitstellungsmusters in Ihrer mandantenfähigen Anwendung darauf, die Kriterien zu bewerten, die für Ihre Anwendung am wichtigsten sind. Wenn Ihre Mehrmandantenanwendung empfindlich auf Latenz reagiert, bietet ein verteiltes Bereitstellungsmuster eine bessere Leistung auf Kosten einer komplexeren Bereitstellung und Wartung. Obwohl Sie einen Teil dieser Komplexität durch DevOps Automatisierung bewältigen können, erfordert dies im Vergleich zu einem zentralisierten Bereitstellungsmuster dennoch zusätzlichen Aufwand.

Wenn Ihre Mehrmandantenanwendung ein isoliertes SaaS-Modell verwendet, können Sie ein verteiltes OPA-Bereitstellungsmuster verwenden, um den isolierten Ansatz zur Isolierung von Mandantendaten nachzuahmen. Dies liegt daran, dass Sie, wenn OPA zusammen mit jedem mandantenspezifischen Anwendungsdienst ausgeführt wird, jede OPA-Bereitstellung so anpassen können, dass sie nur Daten enthält, die diesem Mandanten zugeordnet sind. Das Isolieren von OPA-Daten in einem zentralisierten OPA-Bereitstellungsmuster ist nicht möglich. Wenn Sie entweder ein zentralisiertes Bereitstellungsmuster oder ein verteiltes Muster in Verbindung mit einem gepoolten SaaS-Modell verwenden, muss die Mandantendatenisolierung im OPA-Dokumentenmodell beibehalten werden.

Mandantenisolierung mit dem OPA-Dokumentenmodell

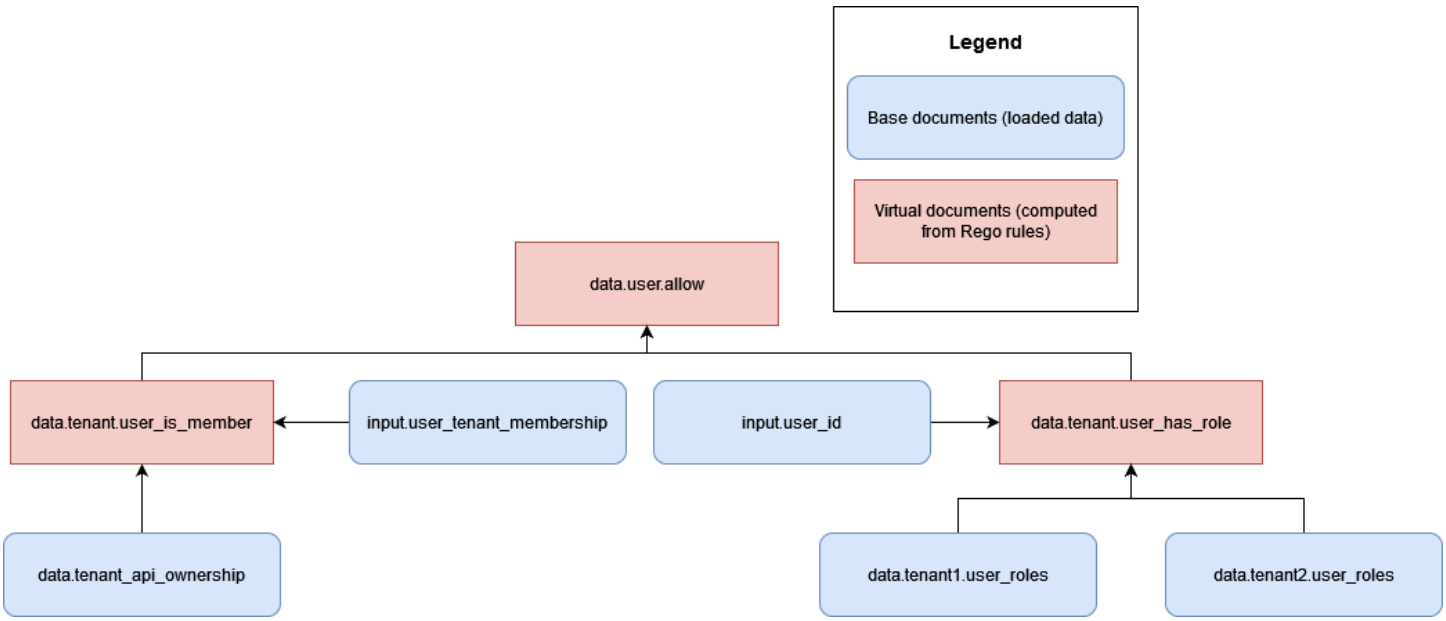
OPA verwendet Dokumente, um Entscheidungen zu treffen. Diese Dokumente können mandantenspezifische Daten enthalten. Sie müssen also überlegen, wie Sie die Mandantendaten isolieren können. OPA-Dokumente bestehen aus Basisdokumenten und virtuellen Dokumenten. Basisdokumente enthalten Daten aus der Außenwelt. Dazu gehören Daten, die OPA direkt zur Verfügung gestellt werden, Daten über die OPA-Anfrage und Daten, die möglicherweise als Eingabe an OPA weitergegeben werden. Virtuelle Dokumente werden anhand von Richtlinien berechnet und enthalten OPA-Richtlinien und -Regeln. Weitere Informationen finden Sie in der OPA-Dokumentation.

Um in OPA ein Dokumentenmodell für eine Mehrmandantenanwendung zu entwerfen, müssen Sie zunächst überlegen, welche Art von Basisdokumenten Sie benötigen, um eine Entscheidung in OPA zu treffen. Wenn diese Basisdokumente mandantenspezifische Daten enthalten, müssen Sie Maßnahmen ergreifen, um sicherzustellen, dass diese Daten nicht versehentlich einem mandantenübergreifenden Zugriff ausgesetzt werden. Zum Glück sind in vielen Fällen keine mandantenspezifischen Daten erforderlich, um eine Entscheidung in OPA zu treffen. Das folgende Beispiel zeigt ein hypothetisches OPA-Dokumentmodell, das den Zugriff auf eine API basierend darauf ermöglicht, welchem Mandanten die API gehört und ob der Benutzer Mitglied des Mandanten ist, wie im Eingabedokument angegeben.



Bei diesem Ansatz hat OPA keinen Zugriff auf mandantenspezifische Daten, mit Ausnahme von Informationen darüber, welche Mandanten eine API besitzen. In diesem Fall besteht kein Grund zur Sorge, dass OPA den mandantenübergreifenden Zugriff erleichtert, da die einzigen Informationen, die OPA verwendet, um eine Zugriffsentscheidung zu treffen, die Zuordnung eines Benutzers zu einem Mandanten und die Zuordnung des Mandanten zu APIs sind. Sie könnten diese Art von OPA-Dokumentenmodell auf ein isoliertes SaaS-Modell anwenden, da jeder Mandant Eigentümer unabhängiger Ressourcen wäre.

Bei vielen RBAC-Autorisierungsansätzen besteht jedoch die Möglichkeit, dass Informationen mandantenübergreifend offengelegt werden. Im folgenden Beispiel ermöglicht ein hypothetisches OPA-Dokumentenmodell den Zugriff auf eine API auf der Grundlage, ob ein Benutzer Mitglied eines Mandanten ist und ob der Benutzer die richtige Rolle für den Zugriff auf die API hat.



Dieses Modell birgt das Risiko eines mandantenübergreifenden Zugriffs, da die Rollen und Berechtigungen mehrerer Mandanten nun OPA zugänglich gemacht werden `data.tenant2.user_roles` müssen, um Autorisierungsentscheidungen treffen zu können. `data.tenant1.user_roles` Um die Mandantenisolierung und den Datenschutz bei der Rollenzuweisung zu wahren, sollten sich diese Daten nicht innerhalb von OPA befinden. RBAC-Daten sollten sich in einer externen Datenquelle wie einer Datenbank befinden. Darüber hinaus sollte OPA nicht verwendet werden, um vordefinierte Rollen bestimmten Berechtigungen zuzuordnen, da es für Mandanten dadurch schwierig wird, ihre eigenen Rollen und Berechtigungen zu definieren. Außerdem ist Ihre Autorisierungslogik dadurch starr und muss ständig aktualisiert werden. Hinweise zur sicheren Einbindung von RBAC-Daten in den OPA-Entscheidungsprozess finden Sie im Abschnitt Empfehlungen zur Mandantenisolierung und zum Datenschutz weiter unten in diesem Leitfaden.

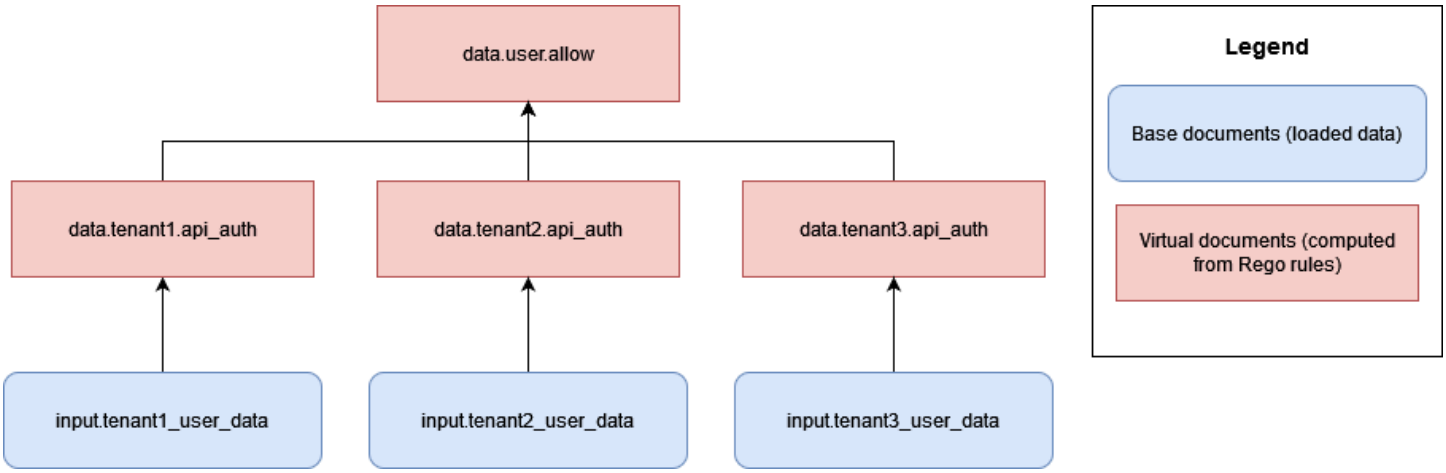
Sie können die Mandantenisolierung in OPA problemlos aufrechterhalten, indem Sie keine mandantenspezifischen Daten als asynchrones Basisdokument speichern. Bei einem asynchronen Basisdokument handelt es sich um Daten, die im Arbeitsspeicher gespeichert sind und in OPA regelmäßig aktualisiert werden können. Andere Basisdokumente, wie z. B. OPA-Eingaben, werden synchron übergeben und sind nur zur Entscheidungszeit verfügbar. Beispielsweise würde die Bereitstellung mandantenspezifischer Daten als Teil der OPA-Eingabe für eine Abfrage keinen Verstoß gegen die Mandantenisolierung darstellen, da diese Daten während des Entscheidungsprozesses nur synchron verfügbar sind.

Onboarding von Mandanten

Die Struktur der OPA-Dokumente muss ein unkompliziertes Onboarding von Mandanten ermöglichen, ohne umständliche Anforderungen einzuführen. Sie können virtuelle Dokumente in der OPA-Dokumentenmodellhierarchie mit Paketen organisieren, und diese Pakete können viele Regeln enthalten. Wenn Sie ein OPA-Dokumentenmodell für eine Mehrmandantenanwendung planen, müssen Sie zunächst ermitteln, welche Daten OPA benötigt, um eine Entscheidung zu treffen. Sie können Daten als Eingabe bereitstellen, sie vorab in OPA laden oder sie zur Entscheidungszeit oder in regelmäßigen Abständen aus externen Datenquellen bereitstellen. Weitere Informationen zur Verwendung externer Daten mit OPA finden Sie im Abschnitt Abrufen externer Daten für ein PDP in OPA weiter unten in diesem Handbuch.

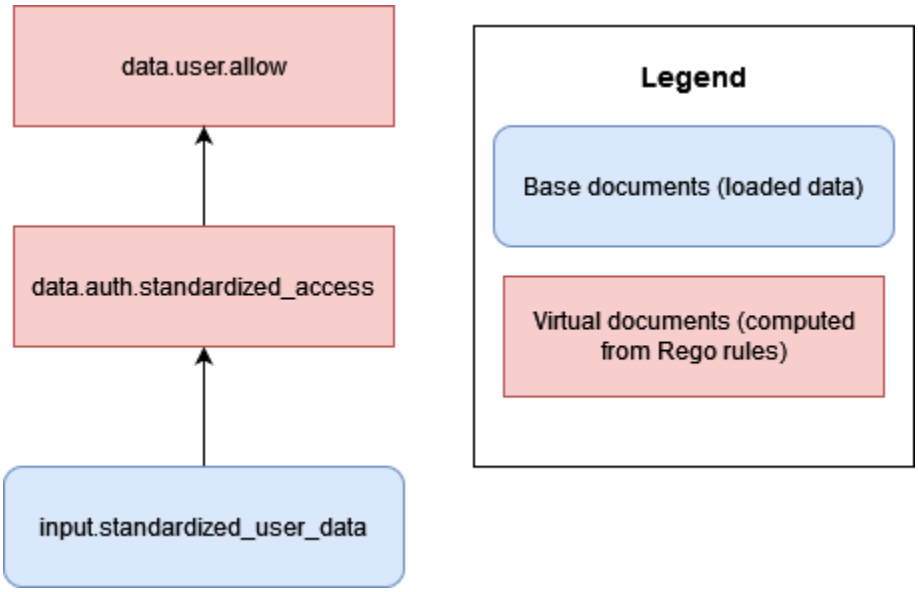
Nachdem Sie die Daten ermittelt haben, die für eine Entscheidung in OPA erforderlich sind, sollten Sie überlegen, wie Sie als Pakete organisierte OPA-Regeln implementieren können, um Entscheidungen anhand dieser Daten zu treffen. In einem isolierten SaaS-Modell, bei dem jeder

Mandant individuelle Anforderungen an die Art und Weise hat, wie Autorisierungsentscheidungen getroffen werden, könnten Sie mandantenspezifische OPA-Regelpakete implementieren.



Der Nachteil dieses Ansatzes besteht darin, dass Sie für jeden Mandanten, den Sie Ihrer SaaS-Anwendung hinzufügen, einen neuen Satz von OPA-Regeln hinzufügen müssen, die für jeden Mandanten spezifisch sind. Dies ist umständlich und schwer zu skalieren, kann aber je nach den Anforderungen Ihrer Mieter unvermeidlich sein.

Wenn in einem SaaS-Poolmodell alle Mandanten Autorisierungsentscheidungen auf der Grundlage derselben Regeln treffen und dieselbe Datenstruktur verwenden, könnten Sie alternativ Standard-OPA-Pakete mit allgemein gültigen Regeln verwenden, um das Onboarding von Mandanten und die Skalierung Ihrer OPA-Implementierung zu vereinfachen.



Wenn möglich, empfehlen wir, generalisierte OPA-Regeln und -Pakete (oder virtuelle Dokumente) zu verwenden, um Entscheidungen auf der Grundlage standardisierter Daten zu treffen, die von jedem

Mandanten bereitgestellt werden. Durch diesen Ansatz ist OPA leicht skalierbar, da Sie nur die Daten ändern, die OPA für jeden Mandanten zur Verfügung gestellt werden, und nicht die Art und Weise, wie OPA seine Entscheidungen anhand seiner Regeln trifft. Die Einführung eines rules-per-tenant Modells ist nur dann erforderlich, wenn einzelne Mandanten individuelle Entscheidungen benötigen oder OPA andere Daten zur Verfügung stellen müssen als andere Mandanten.

DevOps, Überwachung, Protokollierung und Abrufen von Daten für ein PDP

In diesem vorgeschlagenen Autorisierungsparadigma sind Richtlinien im Autorisierungsdienst zentralisiert. Diese Zentralisierung ist gewollt, da eines der Ziele der in diesem Leitfaden erörterten Entwurfsmodelle darin besteht, Richtlinien zu entkoppeln, d. h. die Autorisierungslogik von anderen Komponenten der Anwendung zu entfernen. Sowohl Amazon Verified Permissions als auch der Open Policy Agent (OPA) bieten Mechanismen zur Aktualisierung von Richtlinien, wenn Änderungen an der Autorisierungslogik erforderlich sind.

Im Fall von Verified Permissions bietet das AWS SDK Mechanismen zur programmatischen Aktualisierung von Richtlinien (siehe Amazon Verified Permissions API Reference Guide). Mit dem SDK können Sie neue Richtlinien bei Bedarf bereitstellen. Da es sich bei Verified Permissions um einen verwalteten Dienst handelt, müssen Sie außerdem keine Kontrollebenen oder Agenten verwalten, konfigurieren oder verwalten, um Updates durchzuführen. Wir empfehlen jedoch, eine CI/CD-Pipeline (Continuous Integration and Continuous Deployment) zu verwenden, um die Bereitstellung von Richtlinienspeichern und Richtlinienaktualisierungen für verifizierte Berechtigungen mithilfe des SDK zu verwalten. AWS

Verified Permissions bietet einfachen Zugriff auf Observability-Funktionen. Es kann so konfiguriert werden, dass alle Zugriffsversuche auf CloudWatch Amazon-Protokollgruppen AWS CloudTrail, Amazon Simple Storage Service (Amazon S3) -Buckets oder Amazon Data Firehose-Lieferstreams protokolliert werden, um eine schnelle Reaktion auf Sicherheitsvorfälle und Prüfanfragen zu ermöglichen. Darüber hinaus können Sie den Status des Dienstes Verified Permissions über den überwachen. AWS Health Dashboard Da es sich bei Verified Permissions um einen verwalteten Dienst handelt, wird dessen Integrität durch andere verwaltete Dienste gewährleistet AWS, und Sie können Observability-Funktionen mithilfe anderer AWS verwalteter Dienste konfigurieren.

Im Fall von OPA bieten REST-APIs Möglichkeiten, Richtlinien programmgesteuert zu aktualisieren. Sie können die APIs so konfigurieren, dass sie neue Versionen von Richtlinienpaketen von etablierten Standorten abrufen oder Richtlinien bei Bedarf bereitstellen. Darüber hinaus bietet OPA einen grundlegenden Erkennungsservice, mit dem neue Agenten dynamisch konfiguriert und zentral über eine Kontrollebene verwaltet werden können, die Discovery-Pakete verteilt. (Die Steuerungsebene für OPA muss vom OPA-Operator eingerichtet und konfiguriert werden.) Wir empfehlen Ihnen, eine robuste CI/CD-Pipeline für die Versionierung, Überprüfung und Aktualisierung

von Richtlinien zu erstellen, unabhängig davon, ob es sich bei der Policy-Engine um Verified Permissions, OPA oder eine andere Lösung handelt.

Für OPA bietet die Kontrollebene auch Optionen für die Überwachung und Prüfung. Sie können die Protokolle, die die Autorisierungsentscheidungen von OPA enthalten, zur Protokollaggregation auf entfernte HTTP-Server exportieren. Diese Entscheidungsprotokolle sind für Prüfzwecke von unschätzbarem Wert.

Wenn Sie erwägen, ein Autorisierungsmodell einzuführen, bei dem Entscheidungen zur Zugriffskontrolle von Ihrer Anwendung abgekoppelt sind, sollten Sie sicherstellen, dass Ihr Autorisierungsservice über effektive Überwachungs-, Protokollierungs- und CI/CD-Managementfunktionen für die Einführung neuer PDPs oder die Aktualisierung von Richtlinien verfügt.

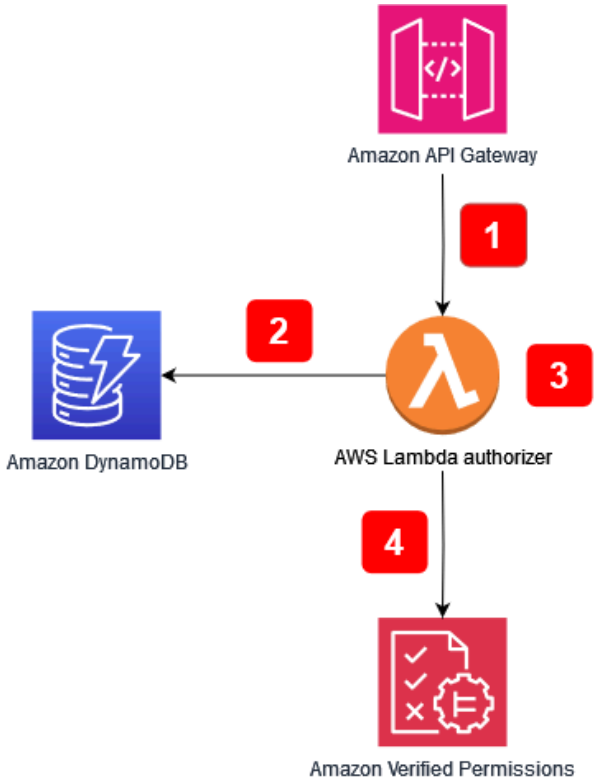
Themen

Abrufen externer Daten für ein PDP in Amazon Verified Permissions

Amazon Verified Permissions unterstützt nicht das Abrufen externer Daten für ein PDP, kann aber vom Benutzer bereitgestellte Daten als Teil seines Schemas speichern. Wenn wie in OPA alle Daten für eine Autorisierungsentscheidung als Teil einer Autorisierungsanfrage oder als Teil eines JSON Web Tokens (JWT) bereitgestellt werden können, das als Teil der Anfrage übergeben wird, ist keine zusätzliche Konfiguration erforderlich. Sie können Verified Permissions jedoch zusätzliche Daten aus externen Quellen über die Autorisierungsanfrage als Teil des Autorisierungsdienstes einer Anwendung zur Verfügung stellen, der Verified Permissions aufruft. Beispielsweise kann der Authorizer-Service einer Anwendung Daten von einer externen Quelle wie DynamoDB oder Amazon RDS abfragen, und diese Dienste können dann die extern bereitgestellten Daten als Teil einer Autorisierungsanfrage einbeziehen.

Das folgende Diagramm zeigt ein Beispiel dafür, wie zusätzliche Daten abgerufen und in eine Autorisierungsanfrage mit verifizierten Berechtigungen integriert werden können. Es kann erforderlich sein, diese Methode zu verwenden, um Daten wie RBAC-Rollenzuordnungen abzurufen, um

zusätzliche Attribute abzurufen, die für Ressourcen oder Prinzipale relevant sind, oder in Fällen, in denen sich Daten in verschiedenen Teilen einer Anwendung befinden und nicht über ein Identity Provider (IdP) -Token bereitgestellt werden können.



Ablauf der Anwendung:

1. Die Anwendung empfängt einen API-Aufruf an Amazon API Gateway und leitet den Anruf an den AWS Lambda Autorisierer weiter.
2. Der Lambda-Autorisierer ruft Amazon DynamoDB auf, um zusätzliche Daten über den Principal abzurufen, der die Anfrage gestellt hat.
3. Der Lambda-Autorisierer nimmt die zusätzlichen Daten in die Autorisierungsanfrage auf, die an Verified Permissions gestellt wurde.
4. Der Lambda-Autorisierer stellt eine Autorisierungsanfrage an Verified Permissions und erhält eine Autorisierungsentscheidung.

Das Diagramm enthält eine Funktion von Amazon API Gateway, die als Lambda-Authorizer bezeichnet wird. Obwohl diese Funktion möglicherweise nicht für APIs verfügbar ist, die von anderen Diensten oder Anwendungen bereitgestellt werden, können Sie das allgemeine Modell der Verwendung eines Autorisierers zum Abrufen zusätzlicher Daten für eine Vielzahl von

Anwendungsfällen replizieren, um sie in eine Autorisierungsanfrage mit verifizierten Berechtigungen zu integrieren.

Abrufen externer Daten für ein PDP in OPA

Für OPA ist keine zusätzliche Konfiguration erforderlich, wenn alle für eine Autorisierungsentscheidung erforderlichen Daten als Eingabe oder als Teil eines JSON Web Tokens (JWT) bereitgestellt werden können, das als Komponente der Abfrage übergeben wird. (Es ist relativ einfach, JWTs- und SaaS-Kontextdaten als Teil der Abfrageeingabe an OPA zu übergeben.) OPA kann beliebige JSON-Eingaben im sogenannten Overload-Input-Ansatz akzeptieren. Wenn ein PDP Daten benötigt, die über das hinausgehen, was als Eingabe oder als JWT aufgenommen werden kann, bietet OPA mehrere Optionen zum Abrufen dieser Daten. Dazu gehören Bündelung, Übertragung von Daten (Replikation) und dynamischer Datenabruf.

OPA-Bündelung

Die OPA-Bündelungsfunktion unterstützt den folgenden Prozess für den externen Datenabruf:

1. Der Policy Enforcement Point (PEP) fordert eine Autorisierungsentscheidung an.
2. OPA lädt neue Richtlinienpakete herunter, einschließlich externer Daten.
3. Der Bündelungsdienst repliziert Daten aus Datenquellen.

Wenn Sie die Bündelungsfunktion verwenden, lädt OPA regelmäßig Richtlinien- und Datenpakete von einem zentralen Paketdienst herunter. (OPA bietet nicht die Implementierung und Einrichtung eines Paketdienstes.) Alle Richtlinien und externen Daten, die aus dem Bundle-Dienst abgerufen werden, werden im Arbeitsspeicher gespeichert. Diese Option funktioniert nicht, wenn die externe Datengröße zu groß ist, um im Speicher gespeichert zu werden, oder wenn sich die Daten zu häufig ändern.

Weitere Informationen zur Bündelungsfunktion finden Sie in der OPA-Dokumentation.

OPA-Replikation (Übertragung von Daten)

Der OPA-Replikationsansatz unterstützt den folgenden Prozess für den externen Datenabruf:

1. Das PEP fordert eine Autorisierungsentscheidung an.
2. Der Datenreplikator überträgt Daten an OPA.

3. Der Datenreplikator repliziert Daten aus Datenquellen.

Bei dieser Alternative zum Bündelungsansatz werden Daten an OPA weitergeleitet, anstatt sie regelmäßig von dort abzurufen. (OPA bietet nicht die Implementierung und Einrichtung eines Replikators.) Der Push-Ansatz hat dieselben Datengrößenbeschränkungen wie der Bündelungsansatz, da OPA alle Daten im Arbeitsspeicher speichert. Der Hauptvorteil der Push-Option besteht darin, dass Sie Daten in OPA mit Deltas aktualisieren können, anstatt jedes Mal alle externen Daten zu ersetzen. Dadurch eignet sich die Push-Option besser für Datensätze, die sich häufig ändern.

Weitere Informationen zur Replikationsoption finden Sie in der OPA-Dokumentation.

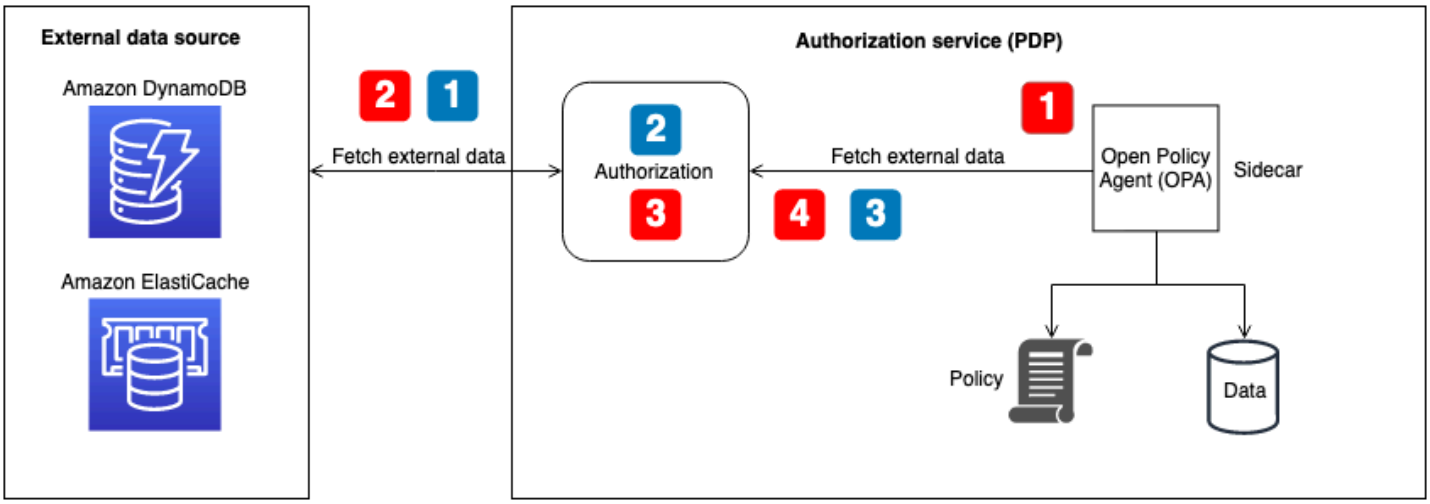
Dynamischer OPA-Datenabruf

Wenn die abzurufenden externen Daten zu groß sind, um im OPA-Speicher zwischengespeichert zu werden, können die Daten während der Auswertung einer Autorisierungsentscheidung dynamisch aus einer externen Quelle abgerufen werden. Wenn Sie diesen Ansatz verwenden, sind die Daten immer auf dem neuesten Stand. Dieser Ansatz hat zwei Nachteile: Netzwerklatenz und Zugänglichkeit. Derzeit kann OPA Daten zur Laufzeit nur über eine HTTP-Anfrage abrufen. Wenn die Aufrufe, die an eine externe Datenquelle gehen, keine Daten als HTTP-Antwort zurückgeben können, benötigen sie eine benutzerdefinierte API oder einen anderen Mechanismus, um diese Daten für OPA bereitzustellen. Da OPA Daten nur über HTTP-Anfragen abrufen kann und die Geschwindigkeit beim Abrufen der Daten entscheidend ist, empfehlen wir, dass Sie nach Möglichkeit eine Lösung AWS-Service wie Amazon DynamoDB verwenden, um externe Daten zu speichern.

Weitere Informationen zum Pull-Ansatz finden Sie in der OPA-Dokumentation.

Verwendung eines Autorisierungsdienstes für die Implementierung mit OPA

Wenn Sie externe Daten mithilfe von Bündelung, Replikation oder einem dynamischen Pull-Ansatz abrufen, empfehlen wir, dass der Autorisierungsdienst diese Interaktion ermöglicht. Das liegt daran, dass der Autorisierungsdienst externe Daten abrufen und in JSON umwandeln kann, damit OPA Autorisierungsentscheidungen treffen kann. Das folgende Diagramm zeigt, wie ein Autorisierungsdienst mit diesen drei Ansätzen für den externen Datenabruf funktionieren kann.



Abrufen externer Daten für den OPA-Flow — Bündel- oder dynamischer Datenabruf zur Entscheidungszeit (im Diagramm mit roten nummerierten Callouts dargestellt):

1. OPA ruft den lokalen API-Endpunkt für den Autorisierungsdienst auf, der als Bundle-Endpunkt oder als Endpunkt für den dynamischen Datenabruf bei Autorisierungsentscheidungen konfiguriert ist.
2. Der Autorisierungsdienst fragt die externe Datenquelle ab oder ruft sie auf, um externe Daten abzurufen. (Für einen Bundle-Endpunkt sollten diese Daten auch OPA-Richtlinien und -Regeln enthalten. Bundle-Updates ersetzen alles — sowohl Daten als auch Richtlinien — im OPA-Cache.)
3. Der Autorisierungsdienst führt alle erforderlichen Transformationen an den zurückgegebenen Daten durch, um sie in die erwartete JSON-Eingabe umzuwandeln.
4. Die Daten werden an OPA zurückgegeben. Sie werden für die Bundle-Konfiguration im Arbeitsspeicher zwischengespeichert und sofort für dynamische Autorisierungsentscheidungen verwendet.

Abrufen externer Daten für den OPA-Flow-Replikator (im Diagramm mit blauen nummerierten Callouts dargestellt):

1. Der Replikator (Teil des Autorisierungsdienstes) ruft die externe Datenquelle auf und ruft alle Daten ab, die in OPA aktualisiert werden sollen. Dies kann Richtlinien, Regeln und externe Daten beinhalten. Dieser Aufruf kann in einem festgelegten Rhythmus erfolgen oder als Reaktion auf Datenaktualisierungen in der externen Quelle erfolgen.
2. Der Autorisierungsdienst führt alle erforderlichen Transformationen an den zurückgegebenen Daten durch, um sie in die erwartete JSON-Eingabe umzuwandeln.

3. Der Autorisierungsdienst ruft OPA auf und speichert die Daten im Speicher zwischen. Der Autorisierungsdienst kann Daten, Richtlinien und Regeln selektiv aktualisieren.

Empfehlungen zur Isolierung von Mandanten und zum Datenschutz

Im vorherigen Abschnitt wurden verschiedene Ansätze für die Verwendung externer Daten mit OPA und Amazon Verified Permissions vorgestellt, um Autorisierungsentscheidungen zu erleichtern. Wenn möglich, empfehlen wir, den Overload-Input-Ansatz zu verwenden, um SaaS-Kontextdaten an OPA zu übergeben, um Autorisierungsentscheidungen zu treffen, anstatt Daten im OPA-Speicher zu speichern. Dieser Anwendungsfall gilt nicht für AWS Cloud Map, da er das Speichern externer Daten im Service nicht unterstützt.

Bei Hybridmodellen mit rollenbasierter Zugriffskontrolle (RBAC) oder RBAC und attribute-Based Access Control (ABAC) reichen die Daten, die ausschließlich durch eine Autorisierungsanfrage oder -abfrage bereitgestellt werden, möglicherweise nicht aus, da Rollen und Berechtigungen referenziert werden müssen, um Autorisierungsentscheidungen zu treffen. Um die Mandantenisolierung und den Datenschutz bei der Rollenzuweisung zu wahren, sollten sich diese Daten nicht innerhalb von OPA befinden. RBAC-Daten sollten sich in einer externen Datenquelle wie einer Datenbank befinden oder als Teil von Ansprüchen in einem JWT von einem IdP weitergegeben werden. In Verified Permissions können RBAC-Daten als Teil von Richtlinien und Schemas im Policy-Store-Modell pro Mandant verwaltet werden, da jeder Mandant seinen eigenen, logisch getrennten Richtlinienspeicher hat. Bei einem Modell mit einem gemeinsamen Richtlinienspeicher für mehrere Mandanten sollten sich die Rollenzuordnungsdaten jedoch nicht innerhalb der verifizierten Berechtigungen befinden, um die Mandantenisolierung aufrechtzuerhalten.

Darüber hinaus sollten OPA und verifizierte Berechtigungen nicht verwendet werden, um vordefinierte Rollen bestimmten Berechtigungen zuzuordnen, da es für Mandanten dadurch schwierig wird, ihre eigenen Rollen und Berechtigungen zu definieren. Außerdem ist Ihre Autorisierungslogik dadurch starr und muss ständig aktualisiert werden. Eine Ausnahme von dieser Richtlinie bildet das mandantenspezifische Richtlinienspeichermodell in Verified Permissions, da bei diesem Modell für jeden Mandanten eigene Richtlinien gelten, die für jeden Mandanten unabhängig voneinander bewertet werden können.

Amazon Verified Permissions

Der einzige Ort, an dem Verified Permissions potenziell private RBAC-Daten speichern können, ist das Schema. Dies ist im Modell des Richtlinienspeichers pro Mandant akzeptabel, da jeder

Mandant seinen eigenen logisch getrennten Richtlinienspeicher hat. Dies könnte jedoch die Mandantenisolierung im Modell eines gemeinsamen Richtlinienspeichers mit mehreren Mandanten gefährden. In Fällen, in denen diese Daten für eine Autorisierungsentscheidung erforderlich sind, sollten sie aus einer externen Quelle wie DynamoDB oder Amazon RDS abgerufen und in die Autorisierungsanfrage für verifizierte Berechtigungen aufgenommen werden.

OPA

Zu den sicheren Ansätzen mit OPA zur Wahrung des Datenschutzes und der Mandantenisolierung von RBAC-Daten gehört der dynamische Datenabruf oder die Replikation, um externe Daten abzurufen. Dies liegt daran, dass Sie den im vorherigen Diagramm dargestellten Autorisierungsdienst verwenden können, um nur mandanten- oder benutzerspezifische externe Daten für eine Autorisierungsentscheidung bereitzustellen. Sie können beispielsweise einen Replikator verwenden, um RBAC-Daten oder eine Berechtigungsmatrix für den OPA-Cache bereitzustellen, wenn sich ein Benutzer anmeldet, und die Daten auf der Grundlage eines in den Eingabedaten angegebenen Benutzers referenzieren lassen. Sie können einen ähnlichen Ansatz mit dynamisch abgerufenen Daten verwenden, um nur die Daten abzurufen, die für Autorisierungsentscheidungen relevant sind. Darüber hinaus müssen diese Daten beim dynamischen Datenabruf nicht in OPA zwischengespeichert werden. Der Bündelungsansatz ist bei der Aufrechterhaltung der Mandantenisolierung nicht so effektiv wie der dynamische Abrufansatz, da er alles im OPA-Cache aktualisiert und keine präzisen Aktualisierungen verarbeiten kann. Das Bündelungsmodell ist immer noch ein guter Ansatz für die Aktualisierung von OPA-Richtlinien und Nicht-RBAC-Daten.

Bewährte Methoden

In diesem Abschnitt sind einige wichtige Erkenntnisse aus diesem Leitfaden aufgeführt. Ausführliche Erläuterungen zu den einzelnen Punkten finden Sie unter den Links zu den entsprechenden Abschnitten.

Wählen Sie ein Zugriffskontrollmodell aus, das für Ihre Anwendung geeignet ist

In diesem Handbuch werden verschiedene Zugriffskontrollmodelle beschrieben. Abhängig von Ihrer Anwendung und Ihren Geschäftsanforderungen sollten Sie ein Modell auswählen, das für Sie geeignet ist. Überlegen Sie, wie Sie diese Modelle verwenden können, um Ihre Anforderungen an die Zugriffskontrolle zu erfüllen, und wie sich Ihre Anforderungen an die Zugriffskontrolle entwickeln könnten, sodass Änderungen an Ihrem ausgewählten Ansatz erforderlich sein könnten.

Implementieren Sie ein PDP

Der Policy Decision Point (PDP) kann als Richtlinien- oder Regelmodul bezeichnet werden. Diese Komponente ist dafür verantwortlich, Richtlinien oder Regeln anzuwenden und eine Entscheidung darüber abzugeben, ob ein bestimmter Zugriff zulässig ist. Ein PDP ermöglicht die Auslagerung der Autorisierungslogik im Anwendungscode auf ein separates System. Dies kann den Anwendungscode vereinfachen. Es bietet auch eine easy-to-use idempotente Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder jede andere Anwendungskomponente. Ein PDP kann verwendet werden, um Mietanforderungen in einer Anwendung einheitlich durchzusetzen.

Implementieren Sie PEPs für jede API in Ihrer Anwendung

Für die Implementierung eines Policy-Enforcement-Points (PEP) muss festgelegt werden, wo die Durchsetzung der Zugriffskontrolle in einer Anwendung erfolgen soll. Suchen Sie in einem ersten Schritt nach den Stellen in Ihrem Antrag, an denen Sie PEPs integrieren können. Beachten Sie diesen Grundsatz, wenn Sie entscheiden, wo PEPs hinzugefügt werden sollen:

Wenn eine Anwendung eine API verfügbar macht, sollte es für diese API eine Autorisierung und Zugriffskontrolle geben.

Erwägen Sie die Verwendung von Amazon Verified Permissions oder OPA als Richtlinien-Engine für Ihr PDP

Amazon Verified Permissions bietet Vorteile gegenüber benutzerdefinierten Policy-Engines. Es handelt sich um einen skalierbaren, differenzierten Service zur Verwaltung und Autorisierung von Berechtigungen für die von Ihnen erstellten Anwendungen. Er unterstützt das Schreiben von Richtlinien in der deklarativen Open-Source-Hochsprache Cedar. Daher erfordert die Implementierung einer Policy-Engine mithilfe von Verified Permissions weniger Entwicklungsaufwand als die Implementierung Ihrer eigenen Lösung. Darüber hinaus wird Verified Permissions vollständig verwaltet, sodass Sie die zugrunde liegende Infrastruktur nicht verwalten müssen.

Der Open Policy Agent (OPA) bietet Vorteile gegenüber benutzerdefinierten Policy-Engines. OPA und seine Richtlinienbewertung mit Rego bieten eine flexible, vorgefertigte Policy-Engine, die das Schreiben von Richtlinien in einer deklarativen Sprache auf hoher Ebene unterstützt. Dadurch ist der Aufwand für die Implementierung einer Policy-Engine deutlich geringer als für die Entwicklung einer eigenen Lösung. Darüber hinaus entwickelt sich OPA schnell zu einem gut unterstützten Autorisierungsstandard.

Implementieren Sie eine Kontrollebene für OPA zur Überwachung DevOps und Protokollierung

Da OPA keine Möglichkeit bietet, Änderungen an der Autorisierungslogik über die Quellcodeverwaltung zu aktualisieren und nachzuverfolgen, empfehlen wir Ihnen, eine Steuerungsebene für diese Funktionen zu implementieren. Auf diese Weise können Updates einfacher an OPA-Agenten verteilt werden, insbesondere wenn OPA in einem verteilten System betrieben wird, wodurch der Verwaltungsaufwand bei der Verwendung von OPA reduziert wird. Darüber hinaus kann eine Kontrollebene verwendet werden, um Protokolle für die Aggregation zu sammeln und den Status der OPA-Agenten zu überwachen.

Konfigurieren Sie die Funktionen für Protokollierung und Beobachtbarkeit in Verified Permissions

Verified Permissions bietet einfachen Zugriff auf Observability-Funktionen. Sie können den Service so konfigurieren, dass er alle Zugriffsversuche auf CloudWatch Amazon-Protokollgruppen AWS CloudTrail, S3-Buckets oder Amazon Data Firehose-Lieferstreams protokolliert, um eine schnelle

Reaktion auf Sicherheitsvorfälle und Prüfanfragen zu ermöglichen. Darüber hinaus können Sie den Zustand des Service über die überwachen. AWS Health Dashboard Da es sich bei Verified Permissions um einen verwalteten Dienst handelt, wird dessen Integrität durch andere verwaltete Dienste gewährleistet AWS, und Sie können seine Funktionen zur Überwachung mithilfe anderer AWS verwalteter Dienste konfigurieren.

Verwenden Sie eine CI/CD-Pipeline, um Richtlinienspeicher und Richtlinien in Verified Permissions bereitzustellen und zu aktualisieren

Verified Permissions ist ein verwalteter Service, sodass Sie keine Kontrollebenen oder Agenten verwalten, konfigurieren oder verwalten müssen, um Updates durchzuführen. Wir empfehlen jedoch weiterhin, eine CI/CD-Pipeline (Continuous Integration and Continuous Deployment) zu verwenden, um die Bereitstellung von Richtlinienspeichern und Richtlinienaktualisierungen für verifizierte Berechtigungen mithilfe des SDK zu verwalten. AWS Durch diesen Aufwand können Sie manuellen Aufwand vermeiden und die Wahrscheinlichkeit von Bedienfehlern verringern, wenn Sie Änderungen an Ressourcen mit verifizierten Berechtigungen vornehmen.

Stellen Sie fest, ob externe Daten für Autorisierungsentscheidungen erforderlich sind, und wählen Sie ein Modell aus, das diese Anforderungen berücksichtigt

Wenn ein PDP Autorisierungsentscheidungen ausschließlich auf der Grundlage von Daten treffen kann, die in einem JSON Web Token (JWT) enthalten sind, ist es normalerweise nicht erforderlich, externe Daten zu importieren, um diese Entscheidungen zu treffen. Wenn Sie Verified Permissions oder OPA als PDP verwenden, kann es auch zusätzliche Eingaben akzeptieren, die als Teil der Anfrage übergeben werden, auch wenn diese Daten nicht in einem JWT enthalten sind. Für verifizierte Berechtigungen können Sie einen Kontextparameter für die zusätzlichen Daten verwenden. Für OPA können Sie JSON-Daten als Überladungseingabe verwenden. Wenn Sie ein JWT verwenden, sind Kontext- oder Overload-Eingabemethoden im Allgemeinen viel einfacher als die Verwaltung externer Daten in einer anderen Quelle. Wenn komplexere externe Daten erforderlich sind, um Autorisierungsentscheidungen zu treffen, bietet OPA mehrere Modelle zum Abrufen externer Daten. Verified Permissions kann die Daten in seinen Autorisierungsanfragen ergänzen, indem sie mit einem Autorisierungsdienst auf externe Quellen verweisen.

Häufig gestellte Fragen

Dieser Abschnitt enthält Antworten auf häufig gestellte Fragen zur Implementierung von API-Zugriffskontrolle und -autorisierung in SaaS-Anwendungen mit mehreren Mandanten.

F: Was ist der Unterschied zwischen Autorisierung und Authentifizierung?

Antwort: Bei der Authentifizierung wird überprüft, wer ein Benutzer ist. Die Autorisierung gewährt Benutzern die Erlaubnis, auf eine bestimmte Ressource zuzugreifen.

F: Was ist der Unterschied zwischen Autorisierung und Mandantenisolierung in einer SaaS-Anwendung?

Antwort: Die Mandantenisolierung bezieht sich auf explizite Mechanismen, die in einem SaaS-System verwendet werden, um sicherzustellen, dass die Ressourcen jedes Mandanten isoliert sind, auch wenn sie auf einer gemeinsam genutzten Infrastruktur betrieben werden. Bei der Mehrmandantenautorisierung werden eingehende Aktionen autorisiert und verhindert, dass diese auf dem falschen Mandanten ausgeführt werden. Ein hypothetischer Benutzer könnte authentifiziert und autorisiert werden, könnte aber dennoch auf die Ressourcen eines anderen Mandanten zugreifen. Nichts in Bezug auf Authentifizierung und Autorisierung blockiert diesen Zugriff unbedingt, aber um dieses Ziel zu erreichen, ist eine Isolierung des Mandanten erforderlich. Weitere Informationen zu diesen beiden Konzepten finden Sie in der Diskussion zur Mandantenisolierung im Whitepaper AWS SaaS Architecture Fundamentals.

F: Warum muss ich die Mandantenisolierung für meine SaaS-Anwendung in Betracht ziehen?

Antwort: SaaS-Anwendungen haben mehrere Mandanten. Ein Mandant kann eine Kundenorganisation oder eine externe Entität sein, die diese SaaS-Anwendung verwendet. Je nachdem, wie die Anwendung gestaltet ist, bedeutet dies, dass Mandanten möglicherweise auf gemeinsam genutzte APIs, Datenbanken oder andere Ressourcen zugreifen. Es ist wichtig, die Mandantenisolierung aufrechtzuerhalten, d. h. Konstrukte, die den Zugriff auf Ressourcen streng kontrollieren und jeden Versuch blockieren, auf Ressourcen eines anderen Mandanten zuzugreifen, um zu verhindern, dass Benutzer eines Mandanten auf die privaten Informationen eines anderen Mandanten zugreifen. SaaS-Anwendungen sind häufig so konzipiert, dass sie sicherstellen, dass die Mandantenisolierung während einer gesamten Anwendung aufrechterhalten wird und dass Mandanten nur auf ihre eigenen Ressourcen zugreifen können.

F: Warum benötige ich ein Zugriffskontrollmodell?

Antwort: Zugriffskontrollmodelle werden verwendet, um eine konsistente Methode zu erstellen, mit der bestimmt wird, wie der Zugriff auf Ressourcen in einer Anwendung gewährt wird. Dies kann geschehen, indem Benutzern Rollen zugewiesen werden, die eng an der Geschäftslogik ausgerichtet sind, oder es kann auf anderen kontextuellen Attributen wie der Tageszeit oder der Tatsache basieren, ob ein Benutzer eine vordefinierte Bedingung erfüllt. Zugriffskontrollmodelle bilden die grundlegende Logik, die Ihre Anwendung verwendet, wenn sie Autorisierungsentscheidungen zur Bestimmung der Benutzerberechtigungen trifft.

F: Ist eine API-Zugriffskontrolle für meine Anwendung erforderlich?

Antwort: Ja. APIs sollten immer überprüfen, ob der Aufrufer über den entsprechenden Zugriff verfügt. Eine umfassende API-Zugriffskontrolle stellt außerdem sicher, dass der Zugriff nur auf Mandantenbasis gewährt wird, sodass eine angemessene Isolierung gewährleistet werden kann.

F: Warum werden Policy-Engines oder PDPs für die Autorisierung empfohlen?

Antwort: Ein Policy Decision Point (PDP) ermöglicht die Auslagerung der Autorisierungslogik im Anwendungscode auf ein separates System. Dies kann den Anwendungscode vereinfachen. Es bietet auch eine easy-to-use idempotente Schnittstelle für Autorisierungsentscheidungen für APIs, Microservices, Backend for Frontend (BFF) -Schichten oder jede andere Anwendungskomponente.

F: Was ist ein PEP?

Antwort: Eine Policy-Enforcement-Stelle (PEP) ist für die Entgegennahme von Autorisierungsanfragen zuständig, die zur Prüfung an das PDP gesendet werden. Ein PEP kann sich überall in einer Anwendung befinden, wo Daten und Ressourcen geschützt werden müssen oder wo Autorisierungslogik angewendet wird. PEPs sind im Vergleich zu PDPs relativ einfach. Ein PEP ist nur dafür verantwortlich, eine Autorisierungsentscheidung anzufordern und auszuwerten, und es ist nicht erforderlich, dass eine Autorisierungslogik in diese integriert wird.

F: Wie sollte ich zwischen Amazon Verified Permissions und OPA wählen?

Antwort: Wenn Sie zwischen Verified Permissions und Open Policy Agent (OPA) wählen möchten, sollten Sie immer Ihren Anwendungsfall und Ihre individuellen Anforderungen berücksichtigen. Verified Permissions bietet eine vollständig verwaltete Möglichkeit, detaillierte Berechtigungen zu definieren, Berechtigungen anwendungsübergreifend zu prüfen und das Policy-Verwaltungssystem für Ihre Anwendungen zu zentralisieren und gleichzeitig Ihre Anforderungen an die Anwendungslatenz mit einer Verarbeitung im Millisekundenbereich zu erfüllen. OPA ist eine universell einsetzbare Open-Source-Policy-Engine, die Ihnen auch dabei helfen kann, Richtlinien

in Ihrem gesamten Anwendungsstapel zu vereinheitlichen. Um OPA ausführen zu können, müssen Sie es in Ihrer AWS Umgebung hosten, normalerweise mit einem Container oder Funktionen. AWS Lambda

Verified Permissions verwendet die Open-Source-Richtliniensprache Cedar, wohingegen OPA Rego verwendet. Wenn Sie mit einer dieser Sprachen vertraut sind, könnten Sie sich daher für diese Lösung entscheiden. Wir empfehlen Ihnen jedoch, sich über beide Sprachen zu informieren und dann von dem Problem, das Sie zu lösen versuchen, zurückzugehen, um die beste Lösung für Ihren Anwendungsfall zu finden.

F: Gibt es Open-Source-Alternativen zu Verified Permissions und OPA?

Antwort: Es gibt einige Open-Source-Systeme, die Verified Permissions und dem Open Policy Agent (OPA) ähneln, wie z. B. die Common Expression Language (CEL). Dieser Leitfaden konzentriert sich sowohl auf Verified Permissions als skalierbares Berechtigungsmanagement und differenzierten Autorisierungsservice als auch auf OPA, das weit verbreitet ist, dokumentiert und an viele verschiedene Arten von Anwendungen und Autorisierungsanforderungen anpassbar ist.

F: Muss ich einen Autorisierungsdienst schreiben, um OPA verwenden zu können, oder kann ich direkt mit OPA interagieren?

Antwort: Sie können direkt mit OPA interagieren. Ein Autorisierungsdienst im Kontext dieser Anleitung bezieht sich auf einen Dienst, der Autorisierungsentscheidungsanfragen in OPA-Abfragen übersetzt und umgekehrt. Wenn Ihre Anwendung OPA-Antworten direkt abfragen und annehmen kann, ist es nicht notwendig, diese zusätzliche Komplexität einzuführen.

F: Wie überwache ich meine OPA-Agenten im Hinblick auf Verfügbarkeit und Prüfung?

Antwort: OPA bietet Protokollierung und grundlegende Verfügbarkeitsüberwachung, obwohl die Standardkonfiguration für Unternehmensbereitstellungen wahrscheinlich nicht ausreichend sein wird. Weitere Informationen finden Sie im DevOps Abschnitt „Überwachung und Protokollierung“ weiter oben in diesem Handbuch.

F: Wie kann ich verifizierte Berechtigungen zu Verfügbarkeits- und Prüfungszwecken überwachen?

Antwort: Verified Permissions ist ein AWS verwalteter Dienst, dessen Verfügbarkeit über den überwacht werden kann. AWS Health Dashboard Darüber hinaus ist Verified Permissions in der Lage AWS CloudTrail, sich bei Amazon CloudWatch Logs, Amazon S3 und Amazon Data Firehose anzumelden.

F: Mit welchen Betriebssystemen und AWS-Services kann ich OPA ausführen?

Antwort: Sie können OPA unter macOS, Windows und Linux ausführen. OPA-Agenten können auf Amazon Elastic Compute Cloud (Amazon EC2) -Agenten sowie auf Containerisierungsdiensten wie Amazon Elastic Container Service (Amazon ECS) und Amazon Elastic Kubernetes Service (Amazon EKS) konfiguriert werden.

F: Welche Betriebssysteme und AWS Dienste kann ich verwenden, um Verified Permissions auszuführen?

Antwort: Verified Permissions ist ein AWS verwalteter Dienst und wird betrieben von AWS. Für die Verwendung von Verified Permissions ist keine zusätzliche Konfiguration, Installation oder Hosting erforderlich, mit Ausnahme der Möglichkeit, Autorisierungsanfragen an den Dienst zu stellen.

F: Kann ich OPA auf ausführen? AWS Lambda

Antwort: Sie können OPA auf Lambda als Go-Bibliothek ausführen. Informationen dazu, wie Sie dies für einen API Gateway Lambda-Authorizer tun können, finden Sie im AWS Blogbeitrag Creating a custom Lambda Authorizer using Open Policy Agent.

F: Wie sollte ich mich zwischen einem verteilten PDP und einem zentralisierten PDP-Ansatz entscheiden?

Antwort: Das hängt von Ihrer Anwendung ab. Sie wird höchstwahrscheinlich anhand des Latenzunterschieds zwischen einem verteilten und einem zentralisierten PDP-Modell bestimmt. Wir empfehlen Ihnen, einen Machbarkeitsnachweis zu erstellen und die Leistung Ihrer Anwendung zu testen, um Ihre Lösung zu verifizieren.

F: Kann ich OPA neben APIs auch für Anwendungsfälle verwenden?

Antwort: Ja. Die OPA-Dokumentation enthält Beispiele für Kubernetes, Envoy, Docker, Kafka, SSH und sudo sowie Terraform. Darüber hinaus kann OPA mithilfe von Rego-Teilregeln beliebige JSON-Antworten auf Abfragen zurückgeben. Je nach Abfrage kann OPA verwendet werden, um viele Fragen mit JSON-Antworten zu beantworten.

F: Kann ich Verified Permissions neben APIs auch für Anwendungsfälle verwenden?

Antwort: Ja. Verified Permissions kann auf jede eingegangene Autorisierungsanfrage eine `DENY` Antwort `ALLOW` oder eine Antwort geben. Verified Permissions kann Autorisierungsantworten für jede Anwendung oder jeden Dienst bereitstellen, für den Autorisierungsentscheidungen erforderlich sind.

F: Kann ich Richtlinien in Verified Permissions mithilfe der IAM-Richtliniensprache erstellen?

Antwort: Nein. Sie müssen die Sprache der Cedar-Richtlinien verwenden, um Richtlinien zu verfassen. Cedar wurde entwickelt, um das Berechtigungsmanagement für Anwendungsressourcen von Kunden zu unterstützen, wohingegen die Richtliniensprache AWS Identity and Access Management (IAM) weiterentwickelt wurde, um die Zugriffskontrolle für AWS Ressourcen zu unterstützen.

Nächste Schritte

Die Komplexität der Autorisierung und API-Zugriffskontrolle für Multi-Tenant-SaaS-Anwendungen kann durch einen standardisierten, sprachunabhängigen Ansatz für Autorisierungsentscheidungen überwunden werden. Diese Ansätze umfassen politische Entscheidungspunkte (PDPs) und Policy-Enforcement-Punkte (PDPs), die den Zugang auf flexible und umfassende Weise durchsetzen. Verschiedene Ansätze zur Zugriffskontrolle – wie die rollenbasierte Zugriffskontrolle (RBAC), die attributbasierte Zugriffskontrolle (ABAC) oder eine Kombination aus beiden – können in eine kohärente Zugriffskontrollstrategie integriert werden. Durch das Entfernen der Autorisierungslogik aus einer Anwendung entfällt der Aufwand, der mit der Aufnahme von Ad-hoc-Lösungen zur Zugriffskontrolle in den Anwendungscode verbunden ist. Die in diesem Leitfaden erörterten Implementierungen und bewährten Methoden sollen als Grundlage zur Standardisierung eines Ansatzes für die Implementierung von Autorisierung und API-Zugriffskontrolle in Multi-Tenant-SaaS-Anwendungen dienen. Sie können diese Anleitung als ersten Schritt beim Sammeln von Informationen und beim Entwerfen eines robusten Zugriffskontroll- und Autorisierungssystems für Ihre Anwendung verwenden. Die nächsten Schritte:

- Prüfen Sie Ihre Anforderungen an Autorisierung und Mandantenisolierung und wählen Sie ein Modell für die Zugriffskontrolle für Ihre Anwendung aus.
- Erstellen Sie einen Machbarkeitsnachweis für Tests, indem Sie entweder Amazon Verified Permissions oder Open Policy Agent (OPA) verwenden oder indem Sie Ihre eigene benutzerdefinierte Policy-Engine schreiben.
- Identifizieren Sie APIs und Standorte in Ihrer Anwendung, an denen PEPs implementiert werden sollten.

Ressourcen

Referenzen

- Dokumentation „Amazon Verified Permissions“ (AWS Dokumentation)
- So verwenden Sie Amazon Verified Permissions für die Autorisierung (AWS Blogbeitrag)
- Implementieren Sie einen benutzerdefinierten Autorisierungsrichtlinienanbieter für ASP.NET Core-Apps mithilfe von Amazon Verified Permissions (AWS Blogbeitrag)
- Rollen und Berechtigungen mit PBAC mithilfe von Amazon Verified Permissions verwalten (Blogbeitrag)AWS
- SaaS-Zugriffskontrolle mithilfe von Amazon Verified Permissions mit einem Policystore pro Mandant (AWS Blogbeitrag)
- Die offizielle OPA-Dokumentation
- Warum Unternehmen das zuletzt abgeschlossene CNCF-Projekt — Open Policy Agent — begrüßen müssen (Forbes-Artikel von Janakiram MSV, 8. Februar 2021)
- Erstellen eines benutzerdefinierten Lambda-Autorisierers mit Open Policy Agent (AWS Blogbeitrag)
- Realisieren Sie Richtlinien als Code mit dem AWS Cloud Development Kit über Open Policy Agent (AWS Blogbeitrag)
- Cloud-Governance und Einhaltung von Richtlinien als Code (AWS Blogbeitrag) AWS
- Verwenden von Open Policy Agent auf Amazon EKS (AWS Blogbeitrag)
- Compliance als Code für Amazon ECS mit Open Policy Agent EventBridge, Amazon und AWS Lambda (AWS Blogbeitrag)
- Richtlinienbasierte Gegenmaßnahmen für Kubernetes — Teil 1 (Blogbeitrag)AWS
- Verwenden von API Gateway Lambda-Autorisierern (Dokumentation)AWS

Tools

- The Cedar Playground (zum Testen von Cedar in einem Browser)
- Cedar Github-Repository
- Cedar-Sprachreferenz
- Der Rego Playground (zum Testen von Rego in einem Browser)
- OPA-Repository GitHub

Partner

- Partner für Identity and Access Management
- Partner für Anwendungssicherheit
- Partner für Cloud-Governance
- Partner für Sicherheit und Compliance
- Partner für Sicherheitsbetrieb und Automatisierung
- Partner für Sicherheitstechnik

Dokumentverlauf

In der folgenden Tabelle werden wichtige Änderungen in diesem Leitfaden beschrieben. Um Benachrichtigungen über zukünftige Aktualisierungen zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Details und Beispiele für Amazon Verified Permissions hinzugefügt	Es wurden ausführliche Diskussionen, Beispiele und Code für die Verwendung von Amazon Verified Permissio ns zur Implementierung eines PDP hinzugefügt. Zu den neuen Abschnitten gehören: • Implementierung eines PDP mithilfe von Amazon Verified Permissions • Entwerfen Sie Modelle für Amazon Verified Permissio ns • Überlegungen zum mehrmandantenfähigen Design von Amazon Verified Permissions • Abrufen externer Daten für ein PDP in Amazon Verified Permissions	28. Mai 2024
Klarere Informationen	Das verteilte PDP mit PEPs auf Grundlage des API-Entwu rfsmodells wurde geklärt.	10. Januar 2024
Kurzinformationen zum neuen Service hinzugefügt AWS	Es wurden Informationen zu Amazon Verified Permissio	22. Mai 2023

	ns hinzugefügt, die dieselben Funktionen und Vorteile wie OPA bieten.	
—	Erste Veröffentlichung	17. August 2021

AWS Glossar zu präskriptiven Leitlinien

Im Folgenden finden Sie häufig verwendete Begriffe in Strategien, Leitfäden und Mustern von AWS Prescriptive Guidance. Um Einträge vorzuschlagen, verwenden Sie bitte den Link Feedback geben am Ende des Glossars.

Zahlen

7 Rs

Sieben gängige Migrationsstrategien für die Verlagerung von Anwendungen in die Cloud. Diese Strategien bauen auf den 5 Rs auf, die Gartner 2011 identifiziert hat, und bestehen aus folgenden Elementen:

- Faktorwechsel/Architekturwechsel – Verschieben Sie eine Anwendung und ändern Sie ihre Architektur, indem Sie alle Vorteile cloudnativer Feature nutzen, um Agilität, Leistung und Skalierbarkeit zu verbessern. Dies beinhaltet in der Regel die Portierung des Betriebssystems und der Datenbank. Beispiel: Migrieren Sie Ihre lokale Oracle-Datenbank auf die Amazon Aurora PostgreSQL-kompatible Edition.
- Plattformwechsel (Lift and Reshape) – Verschieben Sie eine Anwendung in die Cloud und führen Sie ein gewisses Maß an Optimierung ein, um die Cloud-Funktionen zu nutzen. Beispiel: Migrieren Sie Ihre lokale Oracle-Datenbank zu Amazon Relational Database Service (Amazon RDS) für Oracle in der. AWS Cloud
- Neukauf (Drop and Shop) – Wechseln Sie zu einem anderen Produkt, indem Sie typischerweise von einer herkömmlichen Lizenz zu einem SaaS-Modell wechseln. Beispiel: Migrieren Sie Ihr CRM-System (Customer Relationship Management) zu Salesforce.com.
- Hostwechsel (Lift and Shift) – Verschieben Sie eine Anwendung in die Cloud, ohne Änderungen vorzunehmen, um die Cloud-Funktionen zu nutzen. Beispiel: Migrieren Sie Ihre lokale Oracle-Datenbank zu Oracle auf einer EC2-Instanz in der. AWS Cloud
- Verschieben (Lift and Shift auf Hypervisor-Ebene) – Verlagern Sie die Infrastruktur in die Cloud, ohne neue Hardware kaufen, Anwendungen umschreiben oder Ihre bestehenden Abläufe ändern zu müssen. Sie migrieren Server von einer lokalen Plattform zu einem Cloud-Dienst für dieselbe Plattform. Beispiel: Migrieren Sie eine Microsoft Hyper-V Anwendung zu AWS.
- Beibehaltung (Wiederaufgreifen) – Bewahren Sie Anwendungen in Ihrer Quellumgebung auf. Dazu können Anwendungen gehören, die einen umfangreichen Faktorwechsel erfordern und

die Sie auf einen späteren Zeitpunkt verschieben möchten, sowie ältere Anwendungen, die Sie beibehalten möchten, da es keine geschäftliche Rechtfertigung für ihre Migration gibt.

- Außerbetriebnahme – Dekommissionierung oder Entfernung von Anwendungen, die in Ihrer Quellumgebung nicht mehr benötigt werden.

A

ABAC

Siehe attributbasierte Zugriffskontrolle.

abstrahierte Dienste

Weitere Informationen finden Sie unter Managed Services.

ACID

Siehe Atomarität, Konsistenz, Isolierung und Haltbarkeit.

Aktiv-Aktiv-Migration

Eine Datenbankmigrationsmethode, bei der die Quell- und Zieldatenbanken synchron gehalten werden (mithilfe eines bidirektionalen Replikationstools oder dualer Schreibvorgänge) und beide Datenbanken Transaktionen von miteinander verbundenen Anwendungen während der Migration verarbeiten. Diese Methode unterstützt die Migration in kleinen, kontrollierten Batches, anstatt einen einmaligen Cutover zu erfordern. Es ist flexibler, erfordert aber mehr Arbeit als eine aktiv-passive Migration.

Aktiv-Passiv-Migration

Eine Datenbankmigrationsmethode, bei der die Quell- und Zieldatenbanken synchron gehalten werden, aber nur die Quelldatenbank Transaktionen von verbindenden Anwendungen verarbeitet, während Daten in die Zieldatenbank repliziert werden. Die Zieldatenbank akzeptiert während der Migration keine Transaktionen.

Aggregatfunktion

Eine SQL-Funktion, die mit einer Gruppe von Zeilen arbeitet und einen einzelnen Rückgabewert für die Gruppe berechnet. Beispiele für Aggregatfunktionen sind `SUM` und`MAX`.

AI

Siehe künstliche Intelligenz.

AIOps

Siehe Operationen mit künstlicher Intelligenz.

Anonymisierung

Der Prozess des dauerhaften Löschens personenbezogener Daten in einem Datensatz. Anonymisierung kann zum Schutz der Privatsphäre beitragen. Anonymisierte Daten gelten nicht mehr als personenbezogene Daten.

Anti-Muster

Eine häufig verwendete Lösung für ein wiederkehrendes Problem, bei dem die Lösung kontraproduktiv, ineffektiv oder weniger wirksam als eine Alternative ist.

Anwendungssteuerung

Ein Sicherheitsansatz, bei dem nur zugelassene Anwendungen verwendet werden können, um ein System vor Schadsoftware zu schützen.

Anwendungsportfolio

Eine Sammlung detaillierter Informationen zu jeder Anwendung, die von einer Organisation verwendet wird, einschließlich der Kosten für die Erstellung und Wartung der Anwendung und ihres Geschäftswerts. Diese Informationen sind entscheidend für den Prozess der Portfoliofindung und -analyse und hilft bei der Identifizierung und Priorisierung der Anwendungen, die migriert, modernisiert und optimiert werden sollen.

künstliche Intelligenz (KI)

Das Gebiet der Datenverarbeitungswissenschaft, das sich der Nutzung von Computertechnologien zur Ausführung kognitiver Funktionen widmet, die typischerweise mit Menschen in Verbindung gebracht werden, wie Lernen, Problemlösen und Erkennen von Mustern. Weitere Informationen finden Sie unter Was ist künstliche Intelligenz?

Operationen mit künstlicher Intelligenz (AIOps)

Der Prozess des Einsatzes von Techniken des Machine Learning zur Lösung betrieblicher Probleme, zur Reduzierung betrieblicher Zwischenfälle und menschlicher Eingriffe sowie zur Steigerung der Servicequalität. Weitere Informationen zur Verwendung von AIOps in der AWS -Migrationsstrategie finden Sie im Leitfaden zur Betriebsintegration.

Asymmetrische Verschlüsselung

Ein Verschlüsselungsalgorithmus, der ein Schlüsselpaar, einen öffentlichen Schlüssel für die Verschlüsselung und einen privaten Schlüssel für die Entschlüsselung verwendet. Sie können den öffentlichen Schlüssel teilen, da er nicht für die Entschlüsselung verwendet wird. Der Zugriff auf den privaten Schlüssel sollte jedoch stark eingeschränkt sein.

Atomizität, Konsistenz, Isolierung, Haltbarkeit (ACID)

Eine Reihe von Softwareeigenschaften, die die Datenvalidität und betriebliche Zuverlässigkeit einer Datenbank auch bei Fehlern, Stromausfällen oder anderen Problemen gewährleisten.

Attributbasierte Zugriffskontrolle (ABAC)

Die Praxis, detaillierte Berechtigungen auf der Grundlage von Benutzerattributen wie Abteilung, Aufgabenrolle und Teamname zu erstellen. Weitere Informationen finden Sie unter ABAC AWS in der AWS Identity and Access Management (IAM-) Dokumentation.

autoritative Datenquelle

Ein Ort, an dem Sie die primäre Version der Daten speichern, die als die zuverlässigste Informationsquelle angesehen wird. Sie können Daten aus der maßgeblichen Datenquelle an andere Speicherorte kopieren, um die Daten zu verarbeiten oder zu ändern, z. B. zu anonymisieren, zu redigieren oder zu pseudonymisieren.

Availability Zone

Ein bestimmter Standort innerhalb einer AWS-Region , der vor Ausfällen in anderen Availability Zones geschützt ist und kostengünstige Netzwerkkonnektivität mit niedriger Latenz zu anderen Availability Zones in derselben Region bietet.

AWS Framework für die Cloud-Einführung (AWS CAF)

Ein Framework mit Richtlinien und bewährten Verfahren, das Unternehmen bei der Entwicklung eines effizienten und effektiven Plans für den erfolgreichen Umstieg auf die Cloud unterstützt. AWS AWS CAF unterteilt die Leitlinien in sechs Schwerpunktbereiche, die als Perspektiven bezeichnet werden: Unternehmen, Mitarbeiter, Unternehmensführung, Plattform, Sicherheit und Betrieb. Die Perspektiven Geschäft, Mitarbeiter und Unternehmensführung konzentrieren sich auf Geschäftskompetenzen und -prozesse, während sich die Perspektiven Plattform, Sicherheit und Betriebsabläufe auf technische Fähigkeiten und Prozesse konzentrieren. Die Personalperspektive zielt beispielsweise auf Stakeholder ab, die sich mit Personalwesen (HR), Personalfunktionen und Personalmanagement befassen. Aus dieser Perspektive bietet AWS CAF Leitlinien für Personalentwicklung, Schulung und Kommunikation, um das Unternehmen auf eine erfolgreiche

Cloud-Einführung vorzubereiten. Weitere Informationen finden Sie auf der AWS -CAF-Webseite und dem AWS -CAF-Whitepaper.

AWS Workload-Qualifizierungsrahmen (AWS WQF)

Ein Tool, das Workloads bei der Datenbankmigration bewertet, Migrationsstrategien empfiehlt und Arbeitsschätzungen bereitstellt. AWS WQF ist in () enthalten. AWS Schema Conversion Tool AWS SCT Es analysiert Datenbankschemas und Codeobjekte, Anwendungscode, Abhängigkeiten und Leistungsmerkmale und stellt Bewertungsberichte bereit.

B

schlechter Bot

Ein Bot, der Einzelpersonen oder Organisationen stören oder ihnen Schaden zufügen soll.

BCP

Siehe Planung der Geschäftskontinuität.

Verhaltensdiagramm

Eine einheitliche, interaktive Ansicht des Ressourcenverhaltens und der Interaktionen im Laufe der Zeit. Sie können ein Verhaltensdiagramm mit Amazon Detective verwenden, um fehlgeschlagene Anmeldeversuche, verdächtige API-Aufrufe und ähnliche Vorgänge zu untersuchen. Weitere Informationen finden Sie unter Daten in einem Verhaltensdiagramm in der Detective-Dokumentation.

Big-Endian-System

Ein System, welches das höchstwertige Byte zuerst speichert. Siehe auch Endianness.

Binäre Klassifikation

Ein Prozess, der ein binäres Ergebnis vorhersagt (eine von zwei möglichen Klassen). Beispielsweise könnte Ihr ML-Modell möglicherweise Probleme wie „Handelt es sich bei dieser E-Mail um Spam oder nicht?“ vorhersagen müssen oder „Ist dieses Produkt ein Buch oder ein Auto?“

Bloom-Filter

Eine probabilistische, speichereffiziente Datenstruktur, mit der getestet wird, ob ein Element Teil einer Menge ist.

Blau/Grün-Bereitstellung

Eine Bereitstellungsstrategie, bei der Sie zwei separate, aber identische Umgebungen erstellen. Sie führen die aktuelle Anwendungsversion in einer Umgebung (blau) und die neue Anwendungsversion in der anderen Umgebung (grün) aus. Mit dieser Strategie können Sie schnell und mit minimalen Auswirkungen ein Rollback durchführen.

Bot

Eine Softwareanwendung, die automatisierte Aufgaben über das Internet ausführt und menschliche Aktivitäten oder Interaktionen simuliert. Manche Bots sind nützlich oder nützlich, wie z. B. Webcrawler, die Informationen im Internet indexieren. Einige andere Bots, die als bösartige Bots bezeichnet werden, sollen Einzelpersonen oder Organisationen stören oder ihnen Schaden zufügen.

Botnetz

Netzwerke von Bots, die mit Malware infiziert sind und unter der Kontrolle einer einzigen Partei stehen, die als Bot-Herder oder Bot-Operator bezeichnet wird. Botnetze sind der bekannteste Mechanismus zur Skalierung von Bots und ihrer Wirkung.

branch

Ein containerisierter Bereich eines Code-Repositorys. Der erste Zweig, der in einem Repository erstellt wurde, ist der Hauptzweig. Sie können einen neuen Zweig aus einem vorhandenen Zweig erstellen und dann Feature entwickeln oder Fehler in dem neuen Zweig beheben. Ein Zweig, den Sie erstellen, um ein Feature zu erstellen, wird allgemein als Feature-Zweig bezeichnet. Wenn das Feature zur Veröffentlichung bereit ist, führen Sie den Feature-Zweig wieder mit dem Hauptzweig zusammen. Weitere Informationen finden Sie unter Über Branches (GitHub Dokumentation).

Zugang durch Glasbruch

Unter außergewöhnlichen Umständen und im Rahmen eines genehmigten Verfahrens ist dies eine schnelle Methode für einen Benutzer, auf einen Bereich zuzugreifen AWS-Konto , für den er normalerweise keine Zugriffsrechte besitzt. Weitere Informationen finden Sie unter dem Indikator Implementation break-glass procedures in den AWS Well-Architected-Leitlinien.

Brownfield-Strategie

Die bestehende Infrastruktur in Ihrer Umgebung. Wenn Sie eine Brownfield-Strategie für eine Systemarchitektur anwenden, richten Sie sich bei der Gestaltung der Architektur nach den

Einschränkungen der aktuellen Systeme und Infrastruktur. Wenn Sie die bestehende Infrastruktur erweitern, könnten Sie Brownfield- und Greenfield-Strategien mischen.

Puffer-Cache

Der Speicherbereich, in dem die am häufigsten abgerufenen Daten gespeichert werden.

Geschäftsfähigkeit

Was ein Unternehmen tut, um Wert zu generieren (z. B. Vertrieb, Kundenservice oder Marketing). Microservices-Architekturen und Entwicklungsentscheidungen können von den Geschäftskapazitäten beeinflusst werden. Weitere Informationen finden Sie im Abschnitt Organisiert nach Geschäftskapazitäten des Whitepapers Ausführen von containerisierten Microservices in AWS.

Planung der Geschäftskontinuität (BCP)

Ein Plan, der die potenziellen Auswirkungen eines störenden Ereignisses, wie z. B. einer groß angelegten Migration, auf den Betrieb berücksichtigt und es einem Unternehmen ermöglicht, den Betrieb schnell wieder aufzunehmen.

C

CAF

Weitere Informationen finden Sie unter Framework für die AWS Cloud-Einführung.

Bereitstellung auf Kanaren

Die langsame und schrittweise Veröffentlichung einer Version für Endbenutzer. Wenn Sie sich sicher sind, stellen Sie die neue Version bereit und ersetzen die aktuelle Version vollständig.

CCoE

Weitere Informationen finden Sie im Cloud Center of Excellence.

CDC

Siehe Erfassung von Änderungsdaten.

Erfassung von Datenänderungen (CDC)

Der Prozess der Nachverfolgung von Änderungen an einer Datenquelle, z. B. einer Datenbanktabelle, und der Aufzeichnung von Metadaten zu der Änderung. Sie können CDC für

verschiedene Zwecke verwenden, z. B. für die Prüfung oder Replikation von Änderungen in einem Zielsystem, um die Synchronisation aufrechtzuerhalten.

Chaos-Technik

Absichtliches Einführen von Ausfällen oder Störungsereignissen, um die Widerstandsfähigkeit eines Systems zu testen. Sie können AWS Fault Injection Service (AWS FIS) verwenden, um Experimente durchzuführen, die Ihre AWS Workloads stress, und deren Reaktion zu bewerten.

CI/CD

Siehe Continuous Integration und Continuous Delivery.

Klassifizierung

Ein Kategorisierungsprozess, der bei der Erstellung von Vorhersagen hilft. ML-Modelle für Klassifikationsprobleme sagen einen diskreten Wert voraus. Diskrete Werte unterscheiden sich immer voneinander. Beispielsweise muss ein Modell möglicherweise auswerten, ob auf einem Bild ein Auto zu sehen ist oder nicht.

clientseitige Verschlüsselung

Lokale Verschlüsselung von Daten, bevor das Ziel sie AWS-Service empfängt.

Cloud-Kompetenzzentrum (CCoE)

Ein multidisziplinäres Team, das die Cloud-Einführung in der gesamten Organisation vorantreibt, einschließlich der Entwicklung bewährter Cloud-Methoden, der Mobilisierung von Ressourcen, der Festlegung von Migrationszeitplänen und der Begleitung der Organisation durch groß angelegte Transformationen. Weitere Informationen finden Sie in den CCoE-Beiträgen im AWS Cloud Enterprise Strategy Blog.

Cloud Computing

Die Cloud-Technologie, die typischerweise für die Ferndatenspeicherung und das IoT-Gerätemanagement verwendet wird. Cloud Computing ist häufig mit Edge-Computing-Technologie verbunden.

Cloud-Betriebsmodell

In einer IT-Organisation das Betriebsmodell, das zum Aufbau, zur Weiterentwicklung und Optimierung einer oder mehrerer Cloud-Umgebungen verwendet wird. Weitere Informationen finden Sie unter Aufbau Ihres Cloud-Betriebsmodells.

Phasen der Einführung der Cloud

Die vier Phasen, die Unternehmen bei der Migration in der Regel durchlaufen AWS Cloud:

- Projekt – Durchführung einiger Cloud-bezogener Projekte zu Machbarkeitsnachweisen und zu Lernzwecken
- Fundament – Grundlegende Investitionen tätigen, um Ihre Cloud-Einführung zu skalieren (z. B. Einrichtung einer Landing Zone, Definition eines CCoE, Einrichtung eines Betriebsmodells)
- Migration – Migrieren einzelner Anwendungen
- Neuentwicklung – Optimierung von Produkten und Services und Innovation in der Cloud

Diese Phasen wurden von Stephen Orban im Blogbeitrag The Journey Toward Cloud-First & the Stages of Adoption im AWS Cloud Enterprise Strategy-Blog definiert. Informationen darüber, wie sie mit der AWS Migrationsstrategie zusammenhängen, finden Sie im Leitfaden zur Vorbereitung der Migration.

CMDB

Siehe Datenbank für das Konfigurationsmanagement.

Code-Repository

Ein Ort, an dem Quellcode und andere Komponenten wie Dokumentation, Beispiele und Skripts gespeichert und im Rahmen von Versionskontrollprozessen aktualisiert werden. Zu den gängigen Cloud-Repositorys gehören GitHub oder AWS CodeCommit. Jede Version des Codes wird Zweig genannt. In einer Microservice-Struktur ist jedes Repository einer einzelnen Funktionalität gewidmet. Eine einzelne CI/CD-Pipeline kann mehrere Repositorien verwenden.

Kalter Cache

Ein Puffer-Cache, der leer oder nicht gut gefüllt ist oder veraltete oder irrelevante Daten enthält. Dies beeinträchtigt die Leistung, da die Datenbank-Instance aus dem Hauptspeicher oder der Festplatte lesen muss, was langsamer ist als das Lesen aus dem Puffercache.

Kalte Daten

Daten, auf die selten zugegriffen wird und die in der Regel historisch sind. Bei der Abfrage dieser Art von Daten sind langsame Abfragen in der Regel akzeptabel. Durch die Verlagerung dieser Daten auf leistungsschwächere und kostengünstigere Speicherstufen oder -klassen können Kosten gesenkt werden.

Computer Vision (CV)

Ein Bereich der KI, der maschinelles Lernen nutzt, um Informationen aus visuellen Formaten wie digitalen Bildern und Videos zu analysieren und zu extrahieren. AWS Panorama Bietet beispielsweise Geräte an, die CV zu lokalen Kameranetzwerken hinzufügen, und Amazon SageMaker stellt Bildverarbeitungsalgorithmen für CV bereit.

Drift in der Konfiguration

Bei einer Arbeitslast eine Änderung der Konfiguration gegenüber dem erwarteten Zustand. Dies kann dazu führen, dass der Workload nicht mehr richtlinienkonform wird, und zwar in der Regel schrittweise und unbeabsichtigt.

Verwaltung der Datenbankkonfiguration (CMDB)

Ein Repository, das Informationen über eine Datenbank und ihre IT-Umgebung speichert und verwaltet, inklusive Hardware- und Softwarekomponenten und deren Konfigurationen. In der Regel verwenden Sie Daten aus einer CMDB in der Phase der Portfolioerkennung und -analyse der Migration.

Konformitätspaket

Eine Sammlung von AWS Config Regeln und Abhilfemaßnahmen, die Sie zusammenstellen können, um Ihre Konformitäts- und Sicherheitsprüfungen individuell anzupassen. Mithilfe einer YAML-Vorlage können Sie ein Conformance Pack als einzelne Entität in einer AWS-Konto AND-Region oder unternehmensweit bereitstellen. Weitere Informationen finden Sie in der Dokumentation unter Conformance Packs. AWS Config

Kontinuierliche Bereitstellung und kontinuierliche Integration (CI/CD)

Der Prozess der Automatisierung der Quell-, Build-, Test-, Staging- und Produktionsphasen des Softwareveröffentlichungsprozesses. CI/CD wird allgemein als Pipeline beschrieben. CI/CD kann Ihnen helfen, Prozesse zu automatisieren, die Produktivität zu steigern, die Codequalität zu verbessern und schneller zu liefern. Weitere Informationen finden Sie unter Vorteile der kontinuierlichen Auslieferung. CD kann auch für kontinuierliche Bereitstellung stehen. Weitere Informationen finden Sie unterKontinuierliche Auslieferung im Vergleich zu kontinuierlicher Bereitstellung.

CV

Siehe Computer Vision.

D

Daten im Ruhezustand

Daten, die in Ihrem Netzwerk stationär sind, z. B. Daten, die sich im Speicher befinden.

Datenklassifizierung

Ein Prozess zur Identifizierung und Kategorisierung der Daten in Ihrem Netzwerk auf der Grundlage ihrer Kritikalität und Sensitivität. Sie ist eine wichtige Komponente jeder Strategie für das Management von Cybersecurity-Risiken, da sie Ihnen hilft, die geeigneten Schutz- und Aufbewahrungskontrollen für die Daten zu bestimmen. Die Datenklassifizierung ist ein Bestandteil der Sicherheitssäule im AWS Well-Architected Framework. Weitere Informationen finden Sie unter Datenklassifizierung.

Datendrift

Eine signifikante Abweichung zwischen den Produktionsdaten und den Daten, die zum Trainieren eines ML-Modells verwendet wurden, oder eine signifikante Änderung der Eingabedaten im Laufe der Zeit. Datendrift kann die Gesamtqualität, Genauigkeit und Fairness von ML-Modellvorhersagen beeinträchtigen.

Daten während der Übertragung

Daten, die sich aktiv durch Ihr Netzwerk bewegen, z. B. zwischen Netzwerkressourcen.

Datennetz

Ein architektonisches Framework, das verteilte, dezentrale Dateneigentum mit zentraler Verwaltung und Steuerung ermöglicht.

Datenminimierung

Das Prinzip, nur die Daten zu sammeln und zu verarbeiten, die unbedingt erforderlich sind. Durch Datenminimierung im AWS Cloud können Datenschutzrisiken, Kosten und der CO2-Fußabdruck Ihrer Analysen reduziert werden.

Datenperimeter

Eine Reihe präventiver Schutzmaßnahmen in Ihrer AWS Umgebung, die sicherstellen, dass nur vertrauenswürdige Identitäten auf vertrauenswürdige Ressourcen von erwarteten Netzwerken zugreifen. Weitere Informationen finden Sie unter Aufbau eines Datenperimeters auf. AWS

Vorverarbeitung der Daten

Rohdaten in ein Format umzuwandeln, das von Ihrem ML-Modell problemlos verarbeitet werden kann. Die Vorverarbeitung von Daten kann bedeuten, dass bestimmte Spalten oder Zeilen entfernt und fehlende, inkonsistente oder doppelte Werte behoben werden.

Herkunft der Daten

Der Prozess der Nachverfolgung des Ursprungs und der Geschichte von Daten während ihres gesamten Lebenszyklus, z. B. wie die Daten generiert, übertragen und gespeichert wurden.

betroffene Person

Eine Person, deren Daten gesammelt und verarbeitet werden.

Data Warehouse

Ein Datenverwaltungssystem, das Business Intelligence wie Analysen unterstützt. Data Warehouses enthalten in der Regel große Mengen an historischen Daten und werden in der Regel für Abfragen und Analysen verwendet.

Datenbankdefinitionssprache (DDL)

Anweisungen oder Befehle zum Erstellen oder Ändern der Struktur von Tabellen und Objekten in einer Datenbank.

Datenbankmanipulationssprache (DML)

Anweisungen oder Befehle zum Ändern (Einfügen, Aktualisieren und Löschen) von Informationen in einer Datenbank.

DDL

Siehe Datenbankdefinitionssprache.

Deep-Ensemble

Mehrere Deep-Learning-Modelle zur Vorhersage kombinieren. Sie können Deep-Ensembles verwenden, um eine genauere Vorhersage zu erhalten oder um die Unsicherheit von Vorhersagen abzuschätzen.

Deep Learning

Ein ML-Teilbereich, der mehrere Schichten künstlicher neuronaler Netzwerke verwendet, um die Zuordnung zwischen Eingabedaten und Zielvariablen von Interesse zu ermitteln.

defense-in-depth

Ein Ansatz zur Informationssicherheit, bei dem eine Reihe von Sicherheitsmechanismen und -kontrollen sorgfältig in einem Computernetzwerk verteilt werden, um die Vertraulichkeit, Integrität und Verfügbarkeit des Netzwerks und der darin enthaltenen Daten zu schützen. Wenn Sie diese Strategie anwenden AWS, fügen Sie mehrere Steuerelemente auf verschiedenen Ebenen der AWS Organizations Struktur hinzu, um die Ressourcen zu schützen. Ein defense-in-depth Ansatz könnte beispielsweise Multi-Faktor-Authentifizierung, Netzwerksegmentierung und Verschlüsselung kombinieren.

delegierter Administrator

In AWS Organizations kann ein kompatibler Dienst ein AWS Mitgliedskonto registrieren, um die Konten der Organisation und die Berechtigungen für diesen Dienst zu verwalten. Dieses Konto wird als delegierter Administrator für diesen Service bezeichnet. Weitere Informationen und eine Liste kompatibler Services finden Sie unter Services, die mit AWS Organizations funktionieren in der AWS Organizations -Dokumentation.

Bereitstellung

Der Prozess, bei dem eine Anwendung, neue Feature oder Codekorrekturen in der Zielumgebung verfügbar gemacht werden. Die Bereitstellung umfasst das Implementieren von Änderungen an einer Codebasis und das anschließende Erstellen und Ausführen dieser Codebasis in den Anwendungsumgebungen.

Entwicklungsumgebung

Siehe Umgebung.

Detektivische Kontrolle

Eine Sicherheitskontrolle, die darauf ausgelegt ist, ein Ereignis zu erkennen, zu protokollieren und zu warnen, nachdem ein Ereignis eingetreten ist. Diese Kontrollen stellen eine zweite Verteidigungslinie dar und warnen Sie vor Sicherheitsereignissen, bei denen die vorhandenen präventiven Kontrollen umgangen wurden. Weitere Informationen finden Sie unter Detektivische Kontrolle in Implementierung von Sicherheitskontrollen in AWS.

Abbildung des Wertstroms in der Entwicklung (DVSM)

Ein Prozess zur Identifizierung und Priorisierung von Einschränkungen, die sich negativ auf Geschwindigkeit und Qualität im Lebenszyklus der Softwareentwicklung auswirken. DVSM erweitert den Prozess der Wertstromanalyse, der ursprünglich für Lean-Manufacturing-Praktiken

konzipiert wurde. Es konzentriert sich auf die Schritte und Teams, die erforderlich sind, um durch den Softwareentwicklungsprozess Mehrwert zu schaffen und zu steigern.

digitaler Zwilling

Eine virtuelle Darstellung eines realen Systems, z. B. eines Gebäudes, einer Fabrik, einer Industrieanlage oder einer Produktionslinie. Digitale Zwillinge unterstützen vorausschauende Wartung, Fernüberwachung und Produktionsoptimierung.

Maßtabelle

In einem Sternschema eine kleinere Tabelle, die Datenattribute zu quantitativen Daten in einer Faktentabelle enthält. Bei Attributen von Dimensionstabellen handelt es sich in der Regel um Textfelder oder diskrete Zahlen, die sich wie Text verhalten. Diese Attribute werden häufig zum Einschränken von Abfragen, zum Filtern und zur Kennzeichnung von Ergebnismengen verwendet.

Katastrophe

Ein Ereignis, das verhindert, dass ein Workload oder ein System seine Geschäftsziele an seinem primären Einsatzort erfüllt. Diese Ereignisse können Naturkatastrophen, technische Ausfälle oder das Ergebnis menschlichen Handelns sein, z. B. unbeabsichtigte Fehlkonfigurationen oder Malware-Angriffe.

Notfallwiederherstellung (DR)

Die Strategie und der Prozess, die Sie zur Minimierung von Ausfallzeiten und Datenverlusten aufgrund einer Katastrophe anwenden. Weitere Informationen finden Sie unter Disaster Recovery von Workloads unter AWS: Wiederherstellung in der Cloud im AWS Well-Architected Framework.

DML

Siehe Sprache zur Datenbankmanipulation.

Domainorientiertes Design

Ein Ansatz zur Entwicklung eines komplexen Softwaresystems, bei dem seine Komponenten mit sich entwickelnden Domains oder Kerngeschäftszielen verknüpft werden, denen jede Komponente dient. Dieses Konzept wurde von Eric Evans in seinem Buch Domaingesteuertes Design: Bewältigen der Komplexität im Herzen der Software (Boston: Addison-Wesley Professional, 2003) vorgestellt. Informationen darüber, wie Sie domaingesteuertes Design mit dem Strangler-Fig-Muster verwenden können, finden Sie unter Schrittweises Modernisieren älterer Microsoft ASP.NET (ASMX)-Webservices mithilfe von Containern und Amazon API Gateway.

DR

Siehe Disaster Recovery.

Erkennung von Driften

Verfolgung von Abweichungen von einer Basiskonfiguration Sie können es beispielsweise verwenden, AWS CloudFormation um Abweichungen bei den Systemressourcen zu erkennen, oder Sie können AWS Control Tower damit Änderungen in Ihrer landing zone erkennen, die sich auf die Einhaltung von Governance-Anforderungen auswirken könnten.

DVSM

Siehe Abbildung des Wertstroms in der Entwicklung.

E

EDA

Siehe explorative Datenanalyse.

Edge-Computing

Die Technologie, die die Rechenleistung für intelligente Geräte an den Rändern eines IoT-Netzwerks erhöht. Im Vergleich zu Cloud Computing kann Edge Computing die Kommunikationslatenz reduzieren und die Reaktionszeit verbessern.

Verschlüsselung

Ein Rechenprozess, der Klartextdaten, die für Menschen lesbar sind, in Chiffretext umwandelt.

Verschlüsselungsschlüssel

Eine kryptografische Zeichenfolge aus zufälligen Bits, die von einem Verschlüsselungsalgorithmus generiert wird. Schlüssel können unterschiedlich lang sein, und jeder Schlüssel ist so konzipiert, dass er unvorhersehbar und einzigartig ist.

Endianismus

Die Reihenfolge, in der Bytes im Computerspeicher gespeichert werden. Big-Endian-Systeme speichern das höchstwertige Byte zuerst. Little-Endian-Systeme speichern das niedrigwertigste Byte zuerst.

Endpunkt

Siehe Service-Endpunkt.

Endpunkt-Services

Ein Service, den Sie in einer Virtual Private Cloud (VPC) hosten können, um ihn mit anderen Benutzern zu teilen. Sie können einen Endpunktdienst mit anderen AWS-Konten oder AWS Identity and Access Management (IAM AWS PrivateLink -) Prinzipalen erstellen und diesen Berechtigungen gewähren. Diese Konten oder Prinzipale können sich privat mit Ihrem Endpunktservice verbinden, indem sie Schnittstellen-VPC-Endpunkte erstellen. Weitere Informationen finden Sie unter Einen Endpunkt-Service erstellen in der Amazon Virtual Private Cloud (Amazon VPC)-Dokumentation.

Unternehmensressourcenplanung (ERP)

Ein System, das wichtige Geschäftsprozesse (wie Buchhaltung, MES und Projektmanagement) für ein Unternehmen automatisiert und verwaltet.

Envelope-Verschlüsselung

Der Prozess der Verschlüsselung eines Verschlüsselungsschlüssels mit einem anderen Verschlüsselungsschlüssel. Weitere Informationen finden Sie unter Envelope-Verschlüsselung in der AWS Key Management Service (AWS KMS) -Dokumentation.

Umgebung

Eine Instance einer laufenden Anwendung. Die folgenden Arten von Umgebungen sind beim Cloud-Computing üblich:

- Entwicklungsumgebung – Eine Instance einer laufenden Anwendung, die nur dem Kernteam zur Verfügung steht, das für die Wartung der Anwendung verantwortlich ist. Entwicklungsumgebungen werden verwendet, um Änderungen zu testen, bevor sie in höhere Umgebungen übertragen werden. Diese Art von Umgebung wird manchmal als Testumgebung bezeichnet.
- Niedrigere Umgebungen – Alle Entwicklungsumgebungen für eine Anwendung, z. B. solche, die für erste Builds und Tests verwendet wurden.
- Produktionsumgebung – Eine Instance einer laufenden Anwendung, auf die Endbenutzer zugreifen können. In einer CI/CD-Pipeline ist die Produktionsumgebung die letzte Bereitstellungsumgebung.

- Höhere Umgebungen – Alle Umgebungen, auf die auch andere Benutzer als das Kernentwicklungsteam zugreifen können. Dies kann eine Produktionsumgebung, Vorproduktionsumgebungen und Umgebungen für Benutzerakzeptanztests umfassen.

Epics

In der agilen Methodik sind dies funktionale Kategorien, die Ihnen helfen, Ihre Arbeit zu organisieren und zu priorisieren. Epics bieten eine allgemeine Beschreibung der Anforderungen und Implementierungsaufgaben. Zu den Sicherheitsthemen AWS von CAF gehören beispielsweise Identitäts- und Zugriffsmanagement, Detektivkontrollen, Infrastruktursicherheit, Datenschutz und Reaktion auf Vorfälle. Weitere Informationen zu Epics in der AWS -Migrationsstrategie finden Sie im Leitfaden zur Programm-Implementierung.

ERP

Siehe Enterprise Resource Planning.

Explorative Datenanalyse (EDA)

Der Prozess der Analyse eines Datensatzes, um seine Hauptmerkmale zu verstehen. Sie sammeln oder aggregieren Daten und führen dann erste Untersuchungen durch, um Muster zu finden, Anomalien zu erkennen und Annahmen zu überprüfen. EDA wird durchgeführt, indem zusammenfassende Statistiken berechnet und Datenvisualisierungen erstellt werden.

F

Faktentabelle

Die zentrale Tabelle in einem Sternschema. Sie speichert quantitative Daten über den Geschäftsbetrieb. In der Regel enthält eine Faktentabelle zwei Arten von Spalten: Spalten, die Kennzahlen enthalten, und Spalten, die einen Fremdschlüssel für eine Dimensionstabelle enthalten.

schnell scheitern

Eine Philosophie, die häufige und inkrementelle Tests verwendet, um den Entwicklungslebenszyklus zu verkürzen. Dies ist ein wichtiger Bestandteil eines agilen Ansatzes.

Grenze zur Fehlerisolierung

Dabei handelt es sich um eine Grenze AWS Cloud, z. B. eine Availability Zone AWS-Region, eine Steuerungsebene oder eine Datenebene, die die Auswirkungen eines Fehlers begrenzt und die

Widerstandsfähigkeit von Workloads verbessert. Weitere Informationen finden Sie unter Grenzen zur AWS Fehlerisolierung.

Feature-Zweig

Siehe Zweig.

Features

Die Eingabedaten, die Sie verwenden, um eine Vorhersage zu treffen. In einem Fertigungskontext könnten Feature beispielsweise Bilder sein, die regelmäßig von der Fertigungslinie aus aufgenommen werden.

Bedeutung der Feature

Wie wichtig ein Feature für die Vorhersagen eines Modells ist. Dies wird in der Regel als numerischer Wert ausgedrückt, der mit verschiedenen Techniken wie Shapley Additive Explanations (SHAP) und integrierten Gradienten berechnet werden kann. Weitere Informationen finden Sie unter Interpretierbarkeit von Modellen für maschinelles Lernen mit:AWS.

Featuretransformation

Daten für den ML-Prozess optimieren, einschließlich der Anreicherung von Daten mit zusätzlichen Quellen, der Skalierung von Werten oder der Extraktion mehrerer Informationssätze aus einem einzigen Datenfeld. Das ermöglicht dem ML-Modell, von den Daten profitieren. Wenn Sie beispielsweise das Datum „27.05.2021 00:15:37“ in „2021“, „Mai“, „Donnerstag“ und „15“ aufschlüsseln, können Sie dem Lernalgorithmus helfen, nuancierte Muster zu erlernen, die mit verschiedenen Datenkomponenten verknüpft sind.

FGAC

Weitere Informationen finden Sie unter detaillierter Zugriffskontrolle.

Feinkörnige Zugriffskontrolle (FGAC)

Die Verwendung mehrerer Bedingungen, um eine Zugriffsanfrage zuzulassen oder abzulehnen.

Flash-Cut-Migration

Eine Datenbankmigrationsmethode, bei der eine kontinuierliche Datenreplikation durch Erfassung von Änderungsdaten verwendet wird, um Daten in kürzester Zeit zu migrieren, anstatt einen schrittweisen Ansatz zu verwenden. Ziel ist es, Ausfallzeiten auf ein Minimum zu beschränken.

G

Geoblocking

Siehe geografische Einschränkungen.

Geografische Einschränkungen (Geoblocking)

Bei Amazon eine Option CloudFront, um zu verhindern, dass Benutzer in bestimmten Ländern auf Inhaltsverteilungen zugreifen. Sie können eine Zulassungsliste oder eine Sperrliste verwenden, um zugelassene und gesperrte Länder anzugeben. Weitere Informationen finden Sie in der Dokumentation unter Beschränkung der geografischen Verteilung Ihrer Inhalte. CloudFront

Gitflow-Workflow

Ein Ansatz, bei dem niedrigere und höhere Umgebungen unterschiedliche Zweige in einem Quellcode-Repository verwenden. Der Gitflow-Workflow gilt als veraltet, und der Trunk-basierte Workflow ist der moderne, bevorzugte Ansatz.

Greenfield-Strategie

Das Fehlen vorhandener Infrastruktur in einer neuen Umgebung. Bei der Einführung einer Neuausrichtung einer Systemarchitektur können Sie alle neuen Technologien ohne Einschränkung der Kompatibilität mit der vorhandenen Infrastruktur auswählen, auch bekannt als Brownfield. Wenn Sie die bestehende Infrastruktur erweitern, könnten Sie Brownfield- und Greenfield-Strategien mischen.

Integritätsschutz

Eine allgemeine Regel, die dabei hilft, Ressourcen, Richtlinien und die Einhaltung von Vorschriften in allen Organisationseinheiten (OUs) zu regeln. Präventiver Integritätsschutz setzt Richtlinien durch, um die Einhaltung von Standards zu gewährleisten. Sie werden mithilfe von Service-Kontrollrichtlinien und IAM-Berechtigungsgrenzen implementiert. Detektivischer Integritätsschutz erkennt Richtlinienverstöße und Compliance-Probleme und generiert Warnmeldungen zur Abhilfe. Sie werden mithilfe von AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector und benutzerdefinierten AWS Lambda Prüfungen implementiert.

H

HEKTAR

Siehe Hochverfügbarkeit.

Heterogene Datenbankmigration

Migrieren Sie Ihre Quelldatenbank in eine Zieldatenbank, die eine andere Datenbank-Engine verwendet (z. B. Oracle zu Amazon Aurora). Eine heterogene Migration ist in der Regel Teil einer Neuarchitektur, und die Konvertierung des Schemas kann eine komplexe Aufgabe sein. AWS bietet AWS SCT, welches bei Schemakonvertierungen hilft.

hohe Verfügbarkeit (HA)

Die Fähigkeit eines Workloads, im Falle von Herausforderungen oder Katastrophen kontinuierlich und ohne Eingreifen zu arbeiten. HA-Systeme sind so konzipiert, dass sie automatisch ein Failover durchführen, gleichbleibend hohe Leistung bieten und unterschiedliche Lasten und Ausfälle mit minimalen Leistungseinbußen bewältigen.

historische Modernisierung

Ein Ansatz zur Modernisierung und Aufrüstung von Betriebstechnologiesystemen (OT), um den Bedürfnissen der Fertigungsindustrie besser gerecht zu werden. Ein Historian ist eine Art von Datenbank, die verwendet wird, um Daten aus verschiedenen Quellen in einer Fabrik zu sammeln und zu speichern.

Homogene Datenbankmigration

Migrieren Sie Ihre Quelldatenbank zu einer Zieldatenbank, die dieselbe Datenbank-Engine verwendet (z. B. Microsoft SQL Server zu Amazon RDS für SQL Server). Eine homogene Migration ist in der Regel Teil eines Hostwechsels oder eines Plattformwechsels. Sie können native Datenbankserviceprogramme verwenden, um das Schema zu migrieren.

heiße Daten

Daten, auf die häufig zugegriffen wird, z. B. Echtzeitdaten oder aktuelle Translationsdaten. Für diese Daten ist in der Regel eine leistungsstarke Speicherebene oder -klasse erforderlich, um schnelle Abfrageantworten zu ermöglichen.

Hotfix

Eine dringende Lösung für ein kritisches Problem in einer Produktionsumgebung. Aufgrund seiner Dringlichkeit wird ein Hotfix normalerweise außerhalb des typischen DevOps Release-Workflows erstellt.

Hypercare-Phase

Unmittelbar nach dem Cutover, der Zeitraum, in dem ein Migrationsteam die migrierten Anwendungen in der Cloud verwaltet und überwacht, um etwaige Probleme zu beheben. In der Regel dauert dieser Zeitraum 1–4 Tage. Am Ende der Hypercare-Phase überträgt das Migrationsteam in der Regel die Verantwortung für die Anwendungen an das Cloud-Betriebsteam.

I

IaC

Sehen Sie sich Infrastruktur als Code an.

Identitätsbasierte Richtlinie

Eine Richtlinie, die einem oder mehreren IAM-Prinzipalen zugeordnet ist und deren Berechtigungen innerhalb der AWS Cloud Umgebung definiert.

Leerlaufanwendung

Eine Anwendung mit einer durchschnittlichen CPU- und Arbeitsspeicherauslastung zwischen 5 und 20 Prozent über einen Zeitraum von 90 Tagen. In einem Migrationsprojekt ist es üblich, diese Anwendungen außer Betrieb zu nehmen oder sie On-Premises beizubehalten.

IIoT

Siehe Industrielles Internet der Dinge.

unveränderliche Infrastruktur

Ein Modell, das eine neue Infrastruktur für Produktionsworkloads bereitstellt, anstatt die bestehende Infrastruktur zu aktualisieren, zu patchen oder zu modifizieren. Unveränderliche Infrastrukturen sind von Natur aus konsistenter, zuverlässiger und vorhersehbarer als veränderliche Infrastrukturen. Weitere Informationen finden Sie in der Best Practice Deploy using immutable infrastructure im AWS Well-Architected Framework.

Eingehende (ingress) VPC

In einer Architektur AWS mit mehreren Konten ist dies eine VPC, die Netzwerkverbindungen von außerhalb einer Anwendung akzeptiert, überprüft und weiterleitet. Die AWS -Referenzarchitektur für die Sicherheit empfiehlt, Ihr Netzwerkkonto mit eingehenden und ausgehenden VPCs und Inspektions-VPCs einzurichten, um die bidirektionale Schnittstelle zwischen Ihrer Anwendung und dem Internet zu schützen.

Inkrementelle Migration

Eine Cutover-Strategie, bei der Sie Ihre Anwendung in kleinen Teilen migrieren, anstatt eine einziges vollständiges Cutover durchzuführen. Beispielsweise könnten Sie zunächst nur einige Microservices oder Benutzer auf das neue System umstellen. Nachdem Sie sich vergewissert haben, dass alles ordnungsgemäß funktioniert, können Sie weitere Microservices oder Benutzer schrittweise verschieben, bis Sie Ihr Legacy-System außer Betrieb nehmen können. Diese Strategie reduziert die mit großen Migrationen verbundenen Risiken.

Industrie 4.0

Ein Begriff, der 2016 von Klaus Schwab eingeführt wurde und sich auf die Modernisierung von Fertigungsprozessen durch Fortschritte in den Bereichen Konnektivität, Echtzeitdaten, Automatisierung, Analytik und KI/ML bezieht.

Infrastruktur

Alle Ressourcen und Komponenten, die in der Umgebung einer Anwendung enthalten sind.

Infrastructure as Code (IaC)

Der Prozess der Bereitstellung und Verwaltung der Infrastruktur einer Anwendung mithilfe einer Reihe von Konfigurationsdateien. IaC soll Ihnen helfen, das Infrastrukturmanagement zu zentralisieren, Ressourcen zu standardisieren und schnell zu skalieren, sodass neue Umgebungen wiederholbar, zuverlässig und konsistent sind.

Industrielles Internet der Dinge (IIoT)

Einsatz von mit dem Internet verbundenen Sensoren und Geräten in Industriesektoren wie Fertigung, Energie, Automobilindustrie, Gesundheitswesen, Biowissenschaften und Landwirtschaft. Mehr Informationen finden Sie unter Aufbau einer digitalen Transformationsstrategie für das industrielle Internet der Dinge (IIoT).

Inspektions-VPC

In einer Architektur AWS mit mehreren Konten eine zentralisierte VPC, die Inspektionen des Netzwerkverkehrs zwischen VPCs (in derselben oder unterschiedlichen AWS-Regionen), dem Internet und lokalen Netzwerken verwaltet. Die AWS -Referenzarchitektur für die Sicherheit empfiehlt, Ihr Netzwerkkonto mit eingehenden und ausgehenden VPCs und Inspektions-VPCs einzurichten, um die bidirektionale Schnittstelle zwischen Ihrer Anwendung und dem Internet zu schützen.

Internet of Things (IoT)

Das Netzwerk verbundener physischer Objekte mit eingebetteten Sensoren oder Prozessoren, das über das Internet oder über ein lokales Kommunikationsnetzwerk mit anderen Geräten und Systemen kommuniziert. Weitere Informationen finden Sie unter Was ist IoT?

Interpretierbarkeit

Ein Merkmal eines Modells für Machine Learning, das beschreibt, inwieweit ein Mensch verstehen kann, wie die Vorhersagen des Modells von seinen Eingaben abhängen. Weitere Informationen finden Sie unter Interpretierbarkeit von Modellen für Machine Learning mit AWS.

IoT

Siehe Internet der Dinge.

IT information library (ITIL, IT-Informationsbibliothek)

Eine Reihe von bewährten Methoden für die Bereitstellung von IT-Services und die Abstimmung dieser Services auf die Geschäftsanforderungen. ITIL bietet die Grundlage für ITSM.

T service management (ITSM, IT-Servicemanagement)

Aktivitäten im Zusammenhang mit der Gestaltung, Implementierung, Verwaltung und Unterstützung von IT-Services für eine Organisation. Informationen zur Integration von Cloud-Vorgängen mit ITSM-Tools finden Sie im Leitfaden zur Betriebsintegration.

BIS

Weitere Informationen finden Sie in der IT-Informationsbibliothek.

ITSM

Siehe IT-Servicemanagement.

L

Labelbasierte Zugangskontrolle (LBAC)

Eine Implementierung der Mandatory Access Control (MAC), bei der den Benutzern und den Daten selbst jeweils explizit ein Sicherheitslabelwert zugewiesen wird. Die Schnittmenge zwischen der Benutzersicherheitsbeschriftung und der Datensicherheitsbeschriftung bestimmt, welche Zeilen und Spalten für den Benutzer sichtbar sind.

Landing Zone

Eine landing zone ist eine gut strukturierte AWS Umgebung mit mehreren Konten, die skalierbar und sicher ist. Dies ist ein Ausgangspunkt, von dem aus Ihre Organisationen Workloads und Anwendungen schnell und mit Vertrauen in ihre Sicherheits- und Infrastrukturumgebung starten und bereitstellen können. Weitere Informationen zu Landing Zones finden Sie unter Einrichtung einer sicheren und skalierbaren AWS -Umgebung mit mehreren Konten..

Große Migration

Eine Migration von 300 oder mehr Servern.

SCHWARZ

Weitere Informationen finden Sie unter Label-basierte Zugriffskontrolle.

Geringste Berechtigung

Die bewährte Sicherheitsmethode, bei der nur die für die Durchführung einer Aufgabe erforderlichen Mindestberechtigungen erteilt werden. Weitere Informationen finden Sie unter Geringste Berechtigungen anwenden in der IAM-Dokumentation.

Lift and Shift

Siehe 7 Rs.

Little-Endian-System

Ein System, welches das niedrigwertigste Byte zuerst speichert. Siehe auch Endianness.

Niedrigere Umgebungen

Siehe Umwelt.

M

Machine Learning (ML)

Eine Art künstlicher Intelligenz, die Algorithmen und Techniken zur Mustererkennung und zum Lernen verwendet. ML analysiert aufgezeichnete Daten, wie z. B. Daten aus dem Internet der Dinge (IoT), und lernt daraus, um ein statistisches Modell auf der Grundlage von Mustern zu erstellen. Weitere Informationen finden Sie unter Machine Learning.

Hauptzweig

Siehe Filiale.

Malware

Software, die entwickelt wurde, um die Computersicherheit oder den Datenschutz zu gefährden. Malware kann Computersysteme stören, vertrauliche Informationen durchsickern lassen oder sich unbefugten Zugriff verschaffen. Beispiele für Malware sind Viren, Würmer, Ransomware, Trojaner, Spyware und Keylogger.

verwaltete Dienste

AWS-Services für die die Infrastrukturebene, das Betriebssystem und die Plattformen AWS betrieben werden, und Sie greifen auf die Endgeräte zu, um Daten zu speichern und abzurufen. Amazon Simple Storage Service (Amazon S3) und Amazon DynamoDB sind Beispiele für Managed Services. Diese werden auch als abstrakte Dienste bezeichnet.

Manufacturing Execution System (MES)

Ein Softwaresystem zur Nachverfolgung, Überwachung, Dokumentation und Steuerung von Produktionsprozessen, bei denen Rohstoffe in der Fertigung zu fertigen Produkten umgewandelt werden.

MAP

Siehe Migration Acceleration Program.

Mechanismus

Ein vollständiger Prozess, bei dem Sie ein Tool erstellen, die Akzeptanz des Tools vorantreiben und anschließend die Ergebnisse überprüfen, um Anpassungen vorzunehmen. Ein Mechanismus ist ein Zyklus, der sich im Laufe seiner Tätigkeit selbst verstärkt und verbessert. Weitere Informationen finden Sie unter Aufbau von Mechanismen im AWS Well-Architected Framework.

Mitgliedskonto

Alle AWS-Konten außer dem Verwaltungskonto, die Teil einer Organisation in sind. AWS Organizations Ein Konto kann jeweils nur einer Organisation angehören.

DURCHEINANDER

Siehe Manufacturing Execution System.

Message Queuing-Telemetrietransport (MQTT)

Ein leichtes machine-to-machine (M2M) -Kommunikationsprotokoll, das auf dem Publish/Subscribe-Muster für IoT-Geräte mit beschränkten Ressourcen basiert.

Microservice

Ein kleiner, unabhängiger Service, der über klar definierte APIs kommuniziert und in der Regel kleinen, eigenständigen Teams gehört. Ein Versicherungssystem kann beispielsweise Microservices beinhalten, die Geschäftsfunktionen wie Vertrieb oder Marketing oder Subdomains wie Einkauf, Schadenersatz oder Analytik zugeordnet sind. Zu den Vorteilen von Microservices gehören Agilität, flexible Skalierung, einfache Bereitstellung, wiederverwendbarer Code und Ausfallsicherheit. Weitere Informationen finden Sie unter Integration von Microservices mithilfe serverloser Dienste. AWS

Microservices-Architekturen

Ein Ansatz zur Erstellung einer Anwendung mit unabhängigen Komponenten, die jeden Anwendungsprozess als Microservice ausführen. Diese Microservices kommunizieren über eine klar definierte Schnittstelle mithilfe einfacher APIs. Jeder Microservice in dieser Architektur kann aktualisiert, bereitgestellt und skaliert werden, um den Bedarf an bestimmten Funktionen einer Anwendung zu decken. Weitere Informationen finden Sie unter Implementierung von Microservices auf. AWS

Migration Acceleration Program (MAP)

Ein AWS Programm, das Beratung, Unterstützung, Schulungen und Services bietet, um Unternehmen dabei zu unterstützen, eine solide betriebliche Grundlage für die Umstellung auf die Cloud zu schaffen und die anfänglichen Kosten von Migrationen auszugleichen. MAP umfasst eine Migrationsmethode für die methodische Durchführung von Legacy-Migrationen sowie eine Reihe von Tools zur Automatisierung und Beschleunigung gängiger Migrationsszenarien.

Migration in großem Maßstab

Der Prozess, bei dem der Großteil des Anwendungsportfolios in Wellen in die Cloud verlagert wird, wobei in jeder Welle mehr Anwendungen schneller migriert werden. In dieser Phase werden die bewährten Verfahren und Erkenntnisse aus den früheren Phasen zur Implementierung einer Migrationsfabrik von Teams, Tools und Prozessen zur Optimierung der Migration von Workloads durch Automatisierung und agile Bereitstellung verwendet. Dies ist die dritte Phase der AWS - Migrationsstrategie.

Migrationsfabrik

Funktionsübergreifende Teams, die die Migration von Workloads durch automatisierte, agile Ansätze optimieren. Zu den Teams in der Migrationsabteilung gehören in der Regel Betriebsabläufe, Geschäftsanalysten und Eigentümer, Migrationsingenieure, Entwickler und DevOps Experten, die in Sprints arbeiten. Zwischen 20 und 50 Prozent eines Unternehmensanwendungsportfolios bestehen aus sich wiederholenden Mustern, die durch einen Fabrik-Ansatz optimiert werden können. Weitere Informationen finden Sie in Diskussion über Migrationsfabriken und den Leitfaden zur Cloud-Migration-Fabrik in diesem Inhaltssatz.

Migrationsmetadaten

Die Informationen über die Anwendung und den Server, die für den Abschluss der Migration benötigt werden. Für jedes Migrationsmuster ist ein anderer Satz von Migrationsmetadaten erforderlich. Beispiele für Migrationsmetadaten sind das Zielsubnetz, die Sicherheitsgruppe und AWS das Konto.

Migrationsmuster

Eine wiederholbare Migrationsaufgabe, in der die Migrationsstrategie, das Migrationsziel und die verwendete Migrationsanwendung oder der verwendete Migrationsservice detailliert beschrieben werden. Beispiel: Rehost-Migration zu Amazon EC2 mit AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

Ein Online-Tool, das Informationen zur Validierung des Geschäftsszenarios für die Migration auf das bereitstellt. AWS Cloud MPA bietet eine detaillierte Portfoliobewertung (richtige Servergröße, Preisgestaltung, Gesamtbetriebskostenanalyse, Migrationskostenanalyse) sowie Migrationsplanung (Anwendungsdatenanalyse und Datenerfassung, Anwendungsgruppierung, Migrationspriorisierung und Wellenplanung). Das MPA-Tool (Anmeldung erforderlich) steht allen AWS Beratern und APN-Partnerberatern kostenlos zur Verfügung.

Migration Readiness Assessment (MRA)

Der Prozess, bei dem mithilfe des AWS CAF Erkenntnisse über den Cloud-Bereitschaftsstatus eines Unternehmens gewonnen, Stärken und Schwächen identifiziert und ein Aktionsplan zur Schließung festgestellter Lücken erstellt wird. Weitere Informationen finden Sie im Benutzerhandbuch für Migration Readiness. MRA ist die erste Phase der AWS - Migrationsstrategie.

Migrationsstrategie

Der Ansatz, der verwendet wurde, um einen Workload auf den AWS Cloud zu migrieren. Weitere Informationen finden Sie im Eintrag 7 Rs in diesem Glossar und unter Mobilisieren Sie Ihr Unternehmen, um umfangreiche Migrationen zu beschleunigen.

ML

Siehe maschinelles Lernen.

Modernisierung

Umwandlung einer veralteten (veralteten oder monolithischen) Anwendung und ihrer Infrastruktur in ein agiles, elastisches und hochverfügbares System in der Cloud, um Kosten zu senken, die Effizienz zu steigern und Innovationen zu nutzen. Weitere Informationen finden Sie unter Strategie zur Modernisierung von Anwendungen in der AWS Cloud.

Bewertung der Modernisierungsfähigkeit

Eine Bewertung, anhand derer festgestellt werden kann, ob die Anwendungen einer Organisation für die Modernisierung bereit sind, Vorteile, Risiken und Abhängigkeiten identifiziert und ermittelt wird, wie gut die Organisation den zukünftigen Status dieser Anwendungen unterstützen kann. Das Ergebnis der Bewertung ist eine Vorlage der Zielarchitektur, eine Roadmap, in der die Entwicklungsphasen und Meilensteine des Modernisierungsprozesses detailliert beschrieben werden, sowie ein Aktionsplan zur Behebung festgestellter Lücken. Weitere Informationen finden Sie unter Evaluierung der Modernisierungsbereitschaft von Anwendungen in der AWS Cloud.

Monolithische Anwendungen (Monolithen)

Anwendungen, die als ein einziger Service mit eng gekoppelten Prozessen ausgeführt werden. Monolithische Anwendungen haben verschiedene Nachteile. Wenn ein Anwendungs-Feature stark nachgefragt wird, muss die gesamte Architektur skaliert werden. Das Hinzufügen oder Verbessern der Feature einer monolithischen Anwendung wird ebenfalls komplexer, wenn die Codebasis wächst. Um diese Probleme zu beheben, können Sie eine Microservices-Architektur verwenden. Weitere Informationen finden Sie unter Zerlegen von Monolithen in Microservices.

MPA

Siehe Bewertung des Migrationsportfolios.

MQTT

Siehe Message Queuing-Telemetrietransport.

Mehrklassen-Klassifizierung

Ein Prozess, der dabei hilft, Vorhersagen für mehrere Klassen zu generieren (wobei eines von mehr als zwei Ergebnissen vorhergesagt wird). Ein ML-Modell könnte beispielsweise fragen: „Ist dieses Produkt ein Buch, ein Auto oder ein Telefon?“ oder „Welche Kategorie von Produkten ist für diesen Kunden am interessantesten?“

veränderbare Infrastruktur

Ein Modell, das die bestehende Infrastruktur für Produktionsworkloads aktualisiert und modifiziert. Für eine verbesserte Konsistenz, Zuverlässigkeit und Vorhersagbarkeit empfiehlt das AWS Well-Architected Framework die Verwendung einer unveränderlichen Infrastruktur als bewährte Methode.

O

OAC

Weitere Informationen finden Sie unter Origin Access Control.

OAI

Siehe Zugriffsidentität von Origin.

COM

Siehe organisatorisches Change-Management.

Offline-Migration

Eine Migrationsmethode, bei der der Quell-Workload während des Migrationsprozesses heruntergefahren wird. Diese Methode ist mit längeren Ausfallzeiten verbunden und wird in der Regel für kleine, unkritische Workloads verwendet.

OI

Siehe Betriebsintegration.

OLA

Siehe Vereinbarung auf operativer Ebene.

Online-Migration

Eine Migrationsmethode, bei der der Quell-Workload auf das Zielsystem kopiert wird, ohne offline genommen zu werden. Anwendungen, die mit dem Workload verbunden sind, können während der Migration weiterhin funktionieren. Diese Methode beinhaltet keine bis minimale Ausfallzeit und wird in der Regel für kritische Produktionsworkloads verwendet.

OPC-UA

Siehe Open Process Communications — Unified Architecture.

Offene Prozesskommunikation — Einheitliche Architektur (OPC-UA)

Ein machine-to-machine (M2M) -Kommunikationsprotokoll für die industrielle Automatisierung. OPC-UA bietet einen Interoperabilitätsstandard mit Datenverschlüsselungs-, Authentifizierungs- und Autorisierungsschemata.

Vereinbarung auf Betriebsebene (OLA)

Eine Vereinbarung, in der klargestellt wird, welche funktionalen IT-Gruppen sich gegenseitig versprechen zu liefern, um ein Service Level Agreement (SLA) zu unterstützen.

Überprüfung der Betriebsbereitschaft (ORR)

Eine Checkliste mit Fragen und zugehörigen bewährten Methoden, die Ihnen helfen, Vorfälle und mögliche Ausfälle zu verstehen, zu bewerten, zu verhindern oder deren Umfang zu reduzieren. Weitere Informationen finden Sie unter Operational Readiness Reviews (ORR) im AWS Well-Architected Framework.

Betriebstechnologie (OT)

Hardware- und Softwaresysteme, die mit der physischen Umgebung zusammenarbeiten, um industrielle Abläufe, Ausrüstung und Infrastruktur zu steuern. In der Fertigung ist die Integration von OT- und Informationstechnologie (IT) -Systemen ein zentraler Schwerpunkt der Industrie 4.0-Transformationen.

Betriebsintegration (OI)

Der Prozess der Modernisierung von Abläufen in der Cloud, der Bereitschaftsplanung, Automatisierung und Integration umfasst. Weitere Informationen finden Sie im Leitfaden zur Betriebsintegration.

Organisationspfad

Ein Pfad, der von erstellt wird und in AWS CloudTrail dem alle Ereignisse für alle AWS-Konten in einer Organisation protokolliert werden. AWS Organizations Diese Spur wird in jedem AWS-Konto , der Teil der Organisation ist, erstellt und verfolgt die Aktivität in jedem Konto. Weitere Informationen finden Sie in der CloudTrail Dokumentation unter Erstellen eines Pfads für eine Organisation.

Organisatorisches Veränderungsmanagement (OCM)

Ein Framework für das Management wichtiger, disruptiver Geschäftstransformationen aus Sicht der Mitarbeiter, der Kultur und der Führung. OCM hilft Organisationen dabei, sich auf neue Systeme und Strategien vorzubereiten und auf diese umzustellen, indem es die Akzeptanz von Veränderungen beschleunigt, Übergangsprobleme angeht und kulturelle und organisatorische Veränderungen vorantreibt. In der AWS Migrationsstrategie wird dieses Framework aufgrund der Geschwindigkeit des Wandels, der bei Projekten zur Cloud-Einführung erforderlich ist, als Mitarbeiterbeschleunigung bezeichnet. Weitere Informationen finden Sie im OCM-Handbuch.

Ursprungszugriffskontrolle (OAC)

In CloudFront, eine erweiterte Option zur Zugriffsbeschränkung, um Ihre Amazon Simple Storage Service (Amazon S3) -Inhalte zu sichern. OAC unterstützt alle S3-Buckets insgesamt AWS-Regionen, serverseitige Verschlüsselung mit AWS KMS (SSE-KMS) sowie dynamische `PUT` und `DELETE` Anfragen an den S3-Bucket.

Ursprungszugriffsidentität (OAI)

In CloudFront, eine Option zur Zugriffsbeschränkung, um Ihre Amazon S3 S3-Inhalte zu sichern. Wenn Sie OAI verwenden, CloudFront erstellt es einen Principal, mit dem sich Amazon S3 authentifizieren kann. Authentifizierte Principals können nur über eine bestimmte Distribution auf Inhalte in einem S3-Bucket zugreifen. CloudFront Siehe auch OAC, das eine detailliertere und verbesserte Zugriffskontrolle bietet.

ODER

Siehe Überprüfung der Betriebsbereitschaft.

NICHT

Siehe Betriebstechnologie.

Ausgehende (egress) VPC

In einer Architektur AWS mit mehreren Konten eine VPC, die Netzwerkverbindungen verarbeitet, die von einer Anwendung aus initiiert werden. Die AWS -Referenzarchitektur für die Sicherheit empfiehlt, Ihr Netzwerkkonto mit eingehenden und ausgehenden VPCs und Inspektions-VPCs einzurichten, um die bidirektionale Schnittstelle zwischen Ihrer Anwendung und dem Internet zu schützen.

P

Berechtigungsgrenze

Eine IAM-Verwaltungsrichtlinie, die den IAM-Prinzipalen zugeordnet ist, um die maximalen Berechtigungen festzulegen, die der Benutzer oder die Rolle haben kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitys in der IAM-Dokumentation.

persönlich identifizierbare Informationen (PII)

Informationen, die, wenn sie direkt betrachtet oder mit anderen verwandten Daten kombiniert werden, verwendet werden können, um vernünftige Rückschlüsse auf die Identität einer Person zu ziehen. Beispiele für personenbezogene Daten sind Namen, Adressen und Kontaktinformationen.

Personenbezogene Daten

Siehe persönlich identifizierbare Informationen.

Playbook

Eine Reihe vordefinierter Schritte, die die mit Migrationen verbundenen Aufgaben erfassen, z. B. die Bereitstellung zentraler Betriebsfunktionen in der Cloud. Ein Playbook kann die Form von Skripten, automatisierten Runbooks oder einer Zusammenfassung der Prozesse oder Schritte annehmen, die für den Betrieb Ihrer modernisierten Umgebung erforderlich sind.

PLC

Siehe programmierbare Logiksteuerung.

PLM

Siehe Produktlebenszyklusmanagement.

policy

Ein Objekt, das Berechtigungen definieren (siehe identitätsbasierte Richtlinie), Zugriffsbedingungen spezifizieren (siehe ressourcenbasierte Richtlinie) oder die maximalen Berechtigungen für alle Konten in einer Organisation definieren kann AWS Organizations (siehe Dienststeuerungsrichtlinie).

Polyglotte Beharrlichkeit

Unabhängige Auswahl der Datenspeichertechnologie eines Microservices auf der Grundlage von Datenzugriffsmustern und anderen Anforderungen. Wenn Ihre Microservices über dieselbe Datenspeichertechnologie verfügen, kann dies zu Implementierungsproblemen oder zu Leistungseinbußen führen. Microservices lassen sich leichter implementieren und erzielen eine bessere Leistung und Skalierbarkeit, wenn sie den Datenspeicher verwenden, der ihren Anforderungen am besten entspricht. Weitere Informationen finden Sie unter Datenpersistenz in Microservices aktivieren.

Portfoliobewertung

Ein Prozess, bei dem das Anwendungsportfolio ermittelt, analysiert und priorisiert wird, um die Migration zu planen. Weitere Informationen finden Sie in Bewerten der Migrationsbereitschaft.

predicate

Eine Abfragebedingung, die `true` oder zurückgibt`false`, was üblicherweise in einer Klausel vorkommt. `WHERE`

Prädikat Pushdown

Eine Technik zur Optimierung von Datenbankabfragen, bei der die Daten in der Abfrage vor der Übertragung gefiltert werden. Dadurch wird die Datenmenge reduziert, die aus der relationalen Datenbank abgerufen und verarbeitet werden muss, und die Abfrageleistung wird verbessert.

Präventive Kontrolle

Eine Sicherheitskontrolle, die verhindern soll, dass ein Ereignis eintritt. Diese Kontrollen stellen eine erste Verteidigungslinie dar, um unbefugten Zugriff oder unerwünschte Änderungen an Ihrem Netzwerk zu verhindern. Weitere Informationen finden Sie unter Präventive Kontrolle in Implementierung von Sicherheitskontrollen in AWS.

Prinzipal

Eine Entität AWS , die Aktionen ausführen und auf Ressourcen zugreifen kann. Bei dieser Entität handelt es sich in der Regel um einen Root-Benutzer für eine AWS-Konto, eine IAM-Rolle oder

einen Benutzer. Weitere Informationen finden Sie unter Prinzipal in Rollenbegriffe und -konzepte in der IAM-Dokumentation.

Datenschutz durch Design

Ein Ansatz in der Systemtechnik, der den Datenschutz während des gesamten Engineering-Prozesses berücksichtigt.

Privat gehostete Zonen

Ein Container, der Informationen darüber enthält, wie Amazon Route 53 auf DNS-Abfragen für eine Domain und ihre Subdomains innerhalb einer oder mehrerer VPCs reagieren soll. Weitere Informationen finden Sie unter Arbeiten mit privat gehosteten Zonen in der Route-53-Dokumentation.

proaktive Steuerung

Eine Sicherheitskontrolle, die den Einsatz nicht richtlinienkonformer Ressourcen verhindern soll. Mit diesen Steuerelementen werden Ressourcen gescannt, bevor sie bereitgestellt werden. Wenn die Ressource nicht mit der Steuerung konform ist, wird sie nicht bereitgestellt. Weitere Informationen finden Sie im Referenzhandbuch zu Kontrollen in der AWS Control Tower Dokumentation und unter Proaktive Kontrollen unter Implementierung von Sicherheitskontrollen am AWS.

Produktlebenszyklusmanagement (PLM)

Das Management von Daten und Prozessen für ein Produkt während seines gesamten Lebenszyklus, vom Design, der Entwicklung und Markteinführung über Wachstum und Reife bis hin zur Markteinführung und Markteinführung.

Produktionsumgebung

Siehe Umgebung.

Speicherprogrammierbare Steuerung (SPS)

In der Fertigung ein äußerst zuverlässiger, anpassungsfähiger Computer, der Maschinen überwacht und Fertigungsprozesse automatisiert.

Pseudonymisierung

Der Prozess, bei dem persönliche Identifikatoren in einem Datensatz durch Platzhalterwerte ersetzt werden. Pseudonymisierung kann zum Schutz der Privatsphäre beitragen. Pseudonymisierte Daten gelten weiterhin als personenbezogene Daten.

veröffentlichen/abonnieren (pub/sub)

Ein Muster, das asynchrone Kommunikation zwischen Microservices ermöglicht, um die Skalierbarkeit und Reaktionsfähigkeit zu verbessern. In einem auf Microservices basierenden MES kann ein Microservice beispielsweise Ereignismeldungen in einem Kanal veröffentlichen, den andere Microservices abonnieren können. Das System kann neue Microservices hinzufügen, ohne den Veröffentlichungsservice zu ändern.

Q

Abfrageplan

Eine Reihe von Schritten, wie Anweisungen, die für den Zugriff auf die Daten in einem relationalen SQL-Datenbanksystem verwendet werden.

Abfrageplanregression

Wenn ein Datenbankserviceoptimierer einen weniger optimalen Plan wählt als vor einer bestimmten Änderung der Datenbankumgebung. Dies kann durch Änderungen an Statistiken, Beschränkungen, Umgebungseinstellungen, Abfrageparameter-Bindungen und Aktualisierungen der Datenbank-Engine verursacht werden.

R

RACI-Matrix

Siehe verantwortlich, rechenschaftspflichtig, konsultiert, informiert (RACI).

Ransomware

Eine bösartige Software, die entwickelt wurde, um den Zugriff auf ein Computersystem oder Daten zu blockieren, bis eine Zahlung erfolgt ist.

RASCI-Matrix

Siehe verantwortlich, rechenschaftspflichtig, konsultiert, informiert (RACI).

RCAC

Siehe Zugriffskontrolle für Zeilen und Spalten.

Read Replica

Eine Kopie einer Datenbank, die nur für Lesezwecke verwendet wird. Sie können Abfragen an das Lesereplikat weiterleiten, um die Belastung auf Ihrer Primärdatenbank zu reduzieren.

neu strukturieren

Siehe 7 Rs.

Recovery Point Objective (RPO)

Die maximal zulässige Zeitspanne seit dem letzten Datenwiederherstellungspunkt. Dies bestimmt, was als akzeptabler Datenverlust zwischen dem letzten Wiederherstellungspunkt und der Betriebsunterbrechung angesehen wird.

Wiederherstellungszeitziel (RTO)

Die maximal zulässige Verzögerung zwischen der Betriebsunterbrechung und der Wiederherstellung des Dienstes.

Refaktorierung

Siehe 7 Rs.

Region

Eine Sammlung von AWS Ressourcen in einem geografischen Gebiet. Jeder AWS-Region ist isoliert und unabhängig von den anderen, um Fehlertoleranz, Stabilität und Belastbarkeit zu gewährleisten. Weitere Informationen finden Sie unter Geben Sie an, was AWS-Regionen Ihr Konto verwenden kann.

Regression

Eine ML-Technik, die einen numerischen Wert vorhersagt. Zum Beispiel, um das Problem „Zu welchem Preis wird dieses Haus verkauft werden?“ zu lösen Ein ML-Modell könnte ein lineares Regressionsmodell verwenden, um den Verkaufspreis eines Hauses auf der Grundlage bekannter Fakten über das Haus (z. B. die Quadratmeterzahl) vorherzusagen.

rehosten

Siehe 7 Rs.

Veröffentlichung

In einem Bereitstellungsprozess der Akt der Förderung von Änderungen an einer Produktionsumgebung.

umziehen

Siehe 7 Rs.

neue Plattform

Siehe 7 Rs.

Rückkauf

Siehe 7 Rs.

Ausfallsicherheit

Die Fähigkeit einer Anwendung, Störungen zu widerstehen oder sich von ihnen zu erholen. Hochverfügbarkeit und Notfallwiederherstellung sind häufig Überlegungen bei der Planung der Ausfallsicherheit in der. AWS Cloud Weitere Informationen finden Sie unter AWS Cloud Resilienz.

Ressourcenbasierte Richtlinie

Eine mit einer Ressource verknüpfte Richtlinie, z. B. ein Amazon-S3-Bucket, ein Endpunkt oder ein Verschlüsselungsschlüssel. Diese Art von Richtlinie legt fest, welchen Prinzipalen der Zugriff gewährt wird, welche Aktionen unterstützt werden und welche anderen Bedingungen erfüllt sein müssen.

RACI-Matrix (verantwortlich, rechenschaftspflichtig, konsultiert, informiert)

Eine Matrix, die die Rollen und Verantwortlichkeiten aller an Migrationsaktivitäten und Cloud-Operationen beteiligten Parteien definiert. Der Matrixname leitet sich von den in der Matrix definierten Zuständigkeitstypen ab: verantwortlich (R), rechenschaftspflichtig (A), konsultiert (C) und informiert (I). Der Unterstützungstyp (S) ist optional. Wenn Sie Unterstützung einbeziehen, wird die Matrix als RASCI-Matrix bezeichnet, und wenn Sie sie ausschließen, wird sie als RACI-Matrix bezeichnet.

Reaktive Kontrolle

Eine Sicherheitskontrolle, die darauf ausgelegt ist, die Behebung unerwünschter Ereignisse oder Abweichungen von Ihren Sicherheitsstandards voranzutreiben. Weitere Informationen finden Sie unter Reaktive Kontrolle in Implementieren von Sicherheitskontrollen in AWS.

Beibehaltung

Siehe 7 Rs.

zurückziehen

Siehe 7 Rs.

Drehung

Der Vorgang, bei dem ein Geheimnis regelmäßig aktualisiert wird, um es einem Angreifer zu erschweren, auf die Anmeldeinformationen zuzugreifen.

Zugriffskontrolle für Zeilen und Spalten (RCAC)

Die Verwendung einfacher, flexibler SQL-Ausdrücke mit definierten Zugriffsregeln. RCAC besteht aus Zeilenberechtigungen und Spaltenmasken.

RPO

Siehe Recovery Point Objective.

RTO

Siehe Ziel der Wiederherstellungszeit.

Runbook

Eine Reihe manueller oder automatisierter Verfahren, die zur Ausführung einer bestimmten Aufgabe erforderlich sind. Diese sind in der Regel darauf ausgelegt, sich wiederholende Operationen oder Verfahren mit hohen Fehlerquoten zu rationalisieren.

S

SAML 2.0

Ein offener Standard, den viele Identitätsanbieter (IdPs) verwenden. Diese Funktion ermöglicht föderiertes Single Sign-On (SSO), sodass sich Benutzer bei den API-Vorgängen anmelden AWS Management Console oder die AWS API-Operationen aufrufen können, ohne dass Sie einen Benutzer in IAM für alle in Ihrer Organisation erstellen müssen. Weitere Informationen zum SAML-2.0.-basierten Verbund finden Sie unter Über den SAML-2.0-basierten Verbund in der IAM-Dokumentation.

SCADA

Siehe Aufsichtskontrolle und Datenerfassung.

SCP

Siehe Richtlinie zur Dienstkontrolle.

Secret

Interne AWS Secrets Manager, vertrauliche oder eingeschränkte Informationen, wie z. B. ein Passwort oder Benutzeranmeldeinformationen, die Sie in verschlüsselter Form speichern. Es besteht aus dem geheimen Wert und seinen Metadaten. Der geheime Wert kann binär, eine einzelne Zeichenfolge oder mehrere Zeichenketten sein. Weitere Informationen finden Sie unter Was ist in einem Secrets Manager Manager-Geheimnis? in der Secrets Manager Manager-Dokumentation.

Sicherheitskontrolle

Ein technischer oder administrativer Integritätsschutz, der die Fähigkeit eines Bedrohungsakteurs, eine Schwachstelle auszunutzen, verhindert, erkennt oder einschränkt. Es gibt vier Haupttypen von Sicherheitskontrollen: präventiv, detektiv, reaktionsschnell und proaktiv.

Härtung der Sicherheit

Der Prozess, bei dem die Angriffsfläche reduziert wird, um sie widerstandsfähiger gegen Angriffe zu machen. Dies kann Aktionen wie das Entfernen von Ressourcen, die nicht mehr benötigt werden, die Implementierung der bewährten Sicherheitsmethode der Gewährung geringster Berechtigungen oder die Deaktivierung unnötiger Feature in Konfigurationsdateien umfassen.

System zur Verwaltung von Sicherheitsinformationen und Ereignissen (security information and event management – SIEM)

Tools und Services, die Systeme für das Sicherheitsinformationsmanagement (SIM) und das Management von Sicherheitsereignissen (SEM) kombinieren. Ein SIEM-System sammelt, überwacht und analysiert Daten von Servern, Netzwerken, Geräten und anderen Quellen, um Bedrohungen und Sicherheitsverletzungen zu erkennen und Warnmeldungen zu generieren.

Automatisierung von Sicherheitsreaktionen

Eine vordefinierte und programmierte Aktion, die darauf ausgelegt ist, automatisch auf ein Sicherheitsereignis zu reagieren oder es zu beheben. Diese Automatisierungen dienen als detektive oder reaktionsschnelle Sicherheitskontrollen, die Sie bei der Implementierung bewährter AWS Sicherheitsmethoden unterstützen. Beispiele für automatisierte Antwortaktionen sind das Ändern einer VPC-Sicherheitsgruppe, das Patchen einer Amazon EC2 EC2-Instance oder das Rotieren von Anmeldeinformationen.

Serverseitige Verschlüsselung

Verschlüsselung von Daten am Zielort durch denjenigen AWS-Service , der sie empfängt.

Service-Kontrollrichtlinie (SCP)

Eine Richtlinie, die eine zentrale Kontrolle über die Berechtigungen für alle Konten in einer Organisation in AWS Organizations ermöglicht. SCPs definieren Integritätsschutz oder legen Grenzwerte für Aktionen fest, die ein Administrator an Benutzer oder Rollen delegieren kann. Sie können SCPs als Zulassungs- oder Ablehnungslisten verwenden, um festzulegen, welche Services oder Aktionen zulässig oder verboten sind. Weitere Informationen finden Sie in der AWS Organizations Dokumentation unter Richtlinien zur Dienststeuerung.

Service-Endpunkt

Die URL des Einstiegspunkts für einen AWS-Service. Sie können den Endpunkt verwenden, um programmgesteuert eine Verbindung zum Zielservice herzustellen. Weitere Informationen finden Sie unter AWS-Service -Endpunkte in der Allgemeine AWS-Referenz.

Service Level Agreement (SLA)

Eine Vereinbarung, in der klargestellt wird, was ein IT-Team seinen Kunden zu bieten verspricht, z. B. in Bezug auf Verfügbarkeit und Leistung der Services.

Service-Level-Indikator (SLI)

Eine Messung eines Leistungsaspekts eines Dienstes, z. B. seiner Fehlerrate, Verfügbarkeit oder Durchsatz.

Service-Level-Ziel (SLO)

Eine Zielkennzahl, die den Zustand eines Dienstes darstellt, gemessen anhand eines Service-Level-Indikators.

Modell der geteilten Verantwortung

Ein Modell, das die Verantwortung beschreibt, mit der Sie gemeinsam AWS für Cloud-Sicherheit und Compliance verantwortlich sind. AWS ist für die Sicherheit der Cloud verantwortlich, wohingegen Sie für die Sicherheit in der Cloud verantwortlich sind. Weitere Informationen finden Sie unter Modell der geteilten Verantwortung.

SIEM

Siehe Sicherheitsinformations- und Event-Management-System.

Single Point of Failure (SPOF)

Ein Fehler in einer einzelnen, kritischen Komponente einer Anwendung, der das System stören kann.

SLA

Siehe Service Level Agreement.

SLI

Siehe Service-Level-Indikator.

ALSO

Siehe Service-Level-Ziel.

split-and-seed Modell

Ein Muster für die Skalierung und Beschleunigung von Modernisierungsprojekten. Sobald neue Features und Produktversionen definiert werden, teilt sich das Kernteam auf, um neue Produktteams zu bilden. Dies trägt zur Skalierung der Fähigkeiten und Services Ihrer Organisation bei, verbessert die Produktivität der Entwickler und unterstützt schnelle Innovationen. Weitere Informationen finden Sie unter Schrittweiser Ansatz zur Modernisierung von Anwendungen in der AWS Cloud.

SPOTTEN

Siehe Single Point of Failure.

Sternschema

Eine Datenbank-Organisationsstruktur, die eine große Faktentabelle zum Speichern von Transaktions- oder Messdaten und eine oder mehrere kleinere dimensionale Tabellen zum Speichern von Datenattributen verwendet. Diese Struktur ist für die Verwendung in einem Data Warehouse oder für Business Intelligence-Zwecke konzipiert.

Strangler-Fig-Muster

Ein Ansatz zur Modernisierung monolithischer Systeme, bei dem die Systemfunktionen schrittweise umgeschrieben und ersetzt werden, bis das Legacy-System außer Betrieb genommen werden kann. Dieses Muster verwendet die Analogie einer Feigenrebe, die zu einem etablierten Baum heranwächst und schließlich ihren Wirt überwindet und ersetzt. Das Muster wurde eingeführt von Martin Fowler als Möglichkeit, Risiken beim Umschreiben

monolithischer Systeme zu managen. Ein Beispiel für die Anwendung dieses Musters finden Sie unter Schrittweises Modernisieren älterer Microsoft ASP.NET (ASMX)-Webservices mithilfe von Containern und Amazon API Gateway.

Subnetz

Ein Bereich von IP-Adressen in Ihrer VPC. Ein Subnetz muss sich in einer einzigen Availability Zone befinden.

Aufsichtskontrolle und Datenerfassung (SCADA)

In der Fertigung ein System, das Hardware und Software zur Überwachung von Sachanlagen und Produktionsabläufen verwendet.

Symmetrische Verschlüsselung

Ein Verschlüsselungsalgorithmus, der denselben Schlüssel zum Verschlüsseln und Entschlüsseln der Daten verwendet.

synthetisches Testen

Testen eines Systems auf eine Weise, die Benutzerinteraktionen simuliert, um potenzielle Probleme zu erkennen oder die Leistung zu überwachen. Sie können Amazon CloudWatch Synthetics verwenden, um diese Tests zu erstellen.

T

tags

Schlüssel-Wert-Paare, die als Metadaten für die Organisation Ihrer Ressourcen dienen. AWS Mit Tags können Sie Ressourcen verwalten, identifizieren, organisieren, suchen und filtern. Weitere Informationen finden Sie unter Markieren Ihrer AWS -Ressourcen.

Zielvariable

Der Wert, den Sie in überwachtem ML vorhersagen möchten. Dies wird auch als Ergebnisvariable bezeichnet. In einer Fertigungsumgebung könnte die Zielvariable beispielsweise ein Produktfehler sein.

Aufgabenliste

Ein Tool, das verwendet wird, um den Fortschritt anhand eines Runbooks zu verfolgen. Eine Aufgabenliste enthält eine Übersicht über das Runbook und eine Liste mit allgemeinen Aufgaben,

die erledigt werden müssen. Für jede allgemeine Aufgabe werden der geschätzte Zeitaufwand, der Eigentümer und der Fortschritt angegeben.

Testumgebungen

Siehe Umgebung.

Training

Daten für Ihr ML-Modell bereitstellen, aus denen es lernen kann. Die Trainingsdaten müssen die richtige Antwort enthalten. Der Lernalgorithmus findet Muster in den Trainingsdaten, die die Attribute der Input-Daten dem Ziel (die Antwort, die Sie voraussagen möchten) zuordnen. Es gibt ein ML-Modell aus, das diese Muster erfasst. Sie können dann das ML-Modell verwenden, um Voraussagen für neue Daten zu erhalten, bei denen Sie das Ziel nicht kennen.

Transit-Gateway

Ein Transit-Gateway ist ein Netzwerk-Transit-Hub, mit dem Sie Ihre VPCs und On-Premises-Netzwerke miteinander verbinden können. Weitere Informationen finden Sie in der AWS Transit Gateway Dokumentation unter Was ist ein Transit-Gateway.

Stammbasierter Workflow

Ein Ansatz, bei dem Entwickler Feature lokal in einem Feature-Zweig erstellen und testen und diese Änderungen dann im Hauptzweig zusammenführen. Der Hauptzweig wird dann sequentiell für die Entwicklungs-, Vorproduktions- und Produktionsumgebungen erstellt.

Vertrauenswürdiger Zugriff

Gewährung von Berechtigungen für einen Dienst, den Sie angeben, um Aufgaben in Ihrer Organisation AWS Organizations und in deren Konten in Ihrem Namen auszuführen. Der vertrauenswürdige Service erstellt in jedem Konto eine mit dem Service verknüpfte Rolle, wenn diese Rolle benötigt wird, um Verwaltungsaufgaben für Sie auszuführen. Weitere Informationen finden Sie in der AWS Organizations Dokumentation unter Verwendung AWS Organizations mit anderen AWS Diensten.

Optimieren

Aspekte Ihres Trainingsprozesses ändern, um die Genauigkeit des ML-Modells zu verbessern. Sie können das ML-Modell z. B. trainieren, indem Sie einen Beschriftungssatz generieren, Beschriftungen hinzufügen und diese Schritte dann mehrmals unter verschiedenen Einstellungen wiederholen, um das Modell zu optimieren.

Zwei-Pizzen-Team

Ein kleines DevOps Team, das Sie mit zwei Pizzen ernähren können. Eine Teamgröße von zwei Pizzen gewährleistet die bestmögliche Gelegenheit zur Zusammenarbeit bei der Softwareentwicklung.

U

Unsicherheit

Ein Konzept, das sich auf ungenaue, unvollständige oder unbekannte Informationen bezieht, die die Zuverlässigkeit von prädiktiven ML-Modellen untergraben können. Es gibt zwei Arten von Unsicherheit: Epistemische Unsicherheit wird durch begrenzte, unvollständige Daten verursacht, wohingegen aleatorische Unsicherheit durch Rauschen und Randomisierung verursacht wird, die in den Daten liegt. Weitere Informationen finden Sie im Leitfaden Quantifizieren der Unsicherheit in Deep-Learning-Systemen.

undifferenzierte Aufgaben

Diese Arbeit wird auch als Schwerstarbeit bezeichnet. Dabei handelt es sich um Arbeiten, die zwar für die Erstellung und den Betrieb einer Anwendung erforderlich sind, aber dem Endbenutzer keinen direkten Mehrwert bieten oder keinen Wettbewerbsvorteil bieten. Beispiele für undifferenzierte Aufgaben sind Beschaffung, Wartung und Kapazitätsplanung.

höhere Umgebungen

Siehe Umgebung.

V

Vacuuming

Ein Vorgang zur Datenbankwartung, bei dem die Datenbank nach inkrementellen Aktualisierungen bereinigt wird, um Speicherplatz zurückzugewinnen und die Leistung zu verbessern.

Versionskontrolle

Prozesse und Tools zur Nachverfolgung von Änderungen, z. B. Änderungen am Quellcode in einem Repository.

VPC-Peering

Eine Verbindung zwischen zwei VPCs, mit der Sie den Datenverkehr mithilfe von privaten IP-Adressen weiterleiten können. Weitere Informationen finden Sie unter Was ist VPC-Peering? in der Amazon-VPC-Dokumentation.

Schwachstelle

Ein Software- oder Hardwarefehler, der die Sicherheit des Systems gefährdet.

W

Warmer Cache

Ein Puffer-Cache, der aktuelle, relevante Daten enthält, auf die häufig zugegriffen wird. Die Datenbank-Instance kann aus dem Puffer-Cache lesen, was schneller ist als das Lesen aus dem Hauptspeicher oder von der Festplatte.

warme Daten

Daten, auf die selten zugegriffen wird. Bei der Abfrage dieser Art von Daten sind mäßig langsame Abfragen in der Regel akzeptabel.

Fensterfunktion

Eine SQL-Funktion, die eine Berechnung für eine Gruppe von Zeilen durchführt, die sich in irgendeiner Weise auf den aktuellen Datensatz beziehen. Fensterfunktionen sind nützlich für die Verarbeitung von Aufgaben wie die Berechnung eines gleitenden Durchschnitts oder für den Zugriff auf den Wert von Zeilen auf der Grundlage der relativen Position der aktuellen Zeile.

Workload

Ein Workload ist eine Sammlung von Ressourcen und Code, die einen Unternehmenswert bietet, wie z. B. eine kundenorientierte Anwendung oder ein Backend-Prozess.

Workstream

Funktionsgruppen in einem Migrationsprojekt, die für eine bestimmte Reihe von Aufgaben verantwortlich sind. Jeder Workstream ist unabhängig, unterstützt aber die anderen Workstreams im Projekt. Der Portfolio-Workstream ist beispielsweise für die Priorisierung von Anwendungen, die Wellenplanung und die Erfassung von Migrationsmetadaten verantwortlich. Der Portfolio-Workstream liefert diese Komponenten an den Migrations-Workstream, der dann die Server und Anwendungen migriert.

WURM

Sehen Sie einmal schreiben, viele lesen.

WQF

Weitere Informationen finden Sie unter AWS Workload Qualification Framework.

einmal schreiben, viele lesen (WORM)

Ein Speichermodell, das Daten ein einziges Mal schreibt und verhindert, dass die Daten gelöscht oder geändert werden. Autorisierte Benutzer können die Daten so oft wie nötig lesen, aber sie können sie nicht ändern. Diese Datenspeicherinfrastruktur gilt als unveränderlich.

Z

Zero-Day-Exploit

Ein Angriff, in der Regel Malware, der eine Zero-Day-Sicherheitslücke ausnutzt.

Zero-Day-Sicherheitslücke

Ein unfehlbarer Fehler oder eine Sicherheitslücke in einem Produktionssystem. Bedrohungsakteure können diese Art von Sicherheitslücke nutzen, um das System anzugreifen. Entwickler werden aufgrund des Angriffs häufig auf die Sicherheitsanfälligkeit aufmerksam.

Zombie-Anwendung

Eine Anwendung, deren durchschnittliche CPU- und Arbeitsspeichernutzung unter 5 Prozent liegt. In einem Migrationsprojekt ist es üblich, diese Anwendungen außer Betrieb zu nehmen.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.