aws

多租戶 SaaS 授權和 API 存取控制：實作選項和最佳實務

AWS 方案指引

AWS 方案指引: 多租戶 SaaS 授權和 API 存取控制：實作選項和最佳實務

Table of Contents

多租戶 SaaS 授權和 API 存取控制：實作選項和最佳實務

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

2024 年 5 月 (文件歷史記錄)

授權和 API 存取控制是許多軟體應用程式的挑戰，特別是多租戶軟體即服務 (SaaS) 應用程式。當您考慮必須保護的微服務 APIs 的擴散，以及來自不同租用戶、使用者特性和應用程式狀態的大量存取條件時，此複雜性顯而易見。為了有效地解決這些問題，解決方案必須對微服務、後端前端 (BFF) 層和多租用戶 SaaS 應用程式的其他元件提供的許多 APIs 強制執行存取控制。此方法必須隨附一種機制，能夠根據許多因素和屬性做出複雜的存取決策。

傳統上，API 存取控制和授權是由應用程式程式碼中的自訂邏輯處理。這種方法容易出錯且不安全，因為有權存取此程式碼的開發人員可能會意外或刻意變更授權邏輯，進而導致未經授權的存取。稽核應用程式程式碼中自訂邏輯所做的決策很困難，因為稽核人員必須將自己沉浸在自訂邏輯中，以判斷其維護任何特定標準的有效性。此外，API 存取控制通常是不必要的，因為沒有太多 APIs需要保護。應用程式設計中偏袒微服務和服務導向架構的模式轉移已增加必須使用授權和存取控制形式的 APIs 數量。此外，在多租用戶 SaaS 應用程式中維護租用戶型存取的需求，會帶來額外的授權挑戰來保留租用。本指南中概述的最佳實務提供數種好處：

- 授權邏輯可以集中化並以高階宣告性語言撰寫，並非任何程式設計語言特有。
- 授權邏輯會從應用程式程式碼中抽象化，並可作為可重複模式套用至應用程式中的所有 APIs。
- 抽象防止開發人員意外變更以授權邏輯。
- 整合到 SaaS 應用程式既一致又簡單。
- 抽象概念可避免為每個 API 端點撰寫自訂授權邏輯的需求。
- 稽核已簡化，因為稽核人員不再需要檢閱程式碼來判斷許可。
- 本指南中概述的方法支援根據組織的需求使用多個存取控制範例。
- 此授權和存取控制方法提供簡單且直接的方式，以在 SaaS 應用程式中的 API 層維持租戶資料隔離。
- 最佳實務提供一致的方法，讓租戶在授權方面加入和離職。
- 此方法提供不同的授權部署模型（混合或孤立），兩者兼具優點和缺點，如本指南所述。

目標業務成果

此方案指引說明可針對多租用戶 SaaS 應用程式實作的授權和 API 存取控制的可重複設計模式。本指南適用於開發具有複雜授權需求或嚴格 API 存取控制需求之應用程式的任何團隊。架構詳細說明政策決策點 (PDP) 或政策引擎的建立，以及在 APIs 中政策強制執行點 (PEP) 的整合。會討論建立 PDP 的兩個特定選項：搭配使用 Amazon Verified Permissions 與 Cedar SDK，以及搭配使用 Open Policy Agent (OPA) 與 Rego 政策語言。本指南也討論如何根據屬性型存取控制 (ABAC) 模型或角色型存取控制 (RBAC) 模型，或兩種模型的組合，做出存取決策。我們建議您使用本指南中提供的設計模式和概念，在多租用戶 SaaS 應用程式中通知和標準化授權和 API 存取控制的實作。本指南有助於實現下列業務成果：

- 適用於多租戶 SaaS 應用程式的標準化 API 授權架構 – 此架構區分三個元件：儲存和管理政策的政策管理點 (PAP)、評估這些政策以達到授權決策的政策決策點 (PDP)，以及強制執行該決策的政策強制執行點 (PEP)。託管的授權服務 Verified Permissions 可同時做為 PAP 和 PDP。或者，您可以使用 Cedar 或 OPA 等開放原始碼引擎自行建置 PDP。
- 從應用程式解耦授權邏輯 – 嵌入應用程式程式碼或透過臨機操作強制執行機制實作的授權邏輯，可能會受到意外或惡意變更，而導致意外的跨租用戶資料存取或其他安全漏洞。為了協助降低這些可能性，您可以使用 PAP，例如 Verified Permissions、獨立於應用程式程式碼存放授權政策，以及將強大的控管套用至這些政策的管理。政策可以用高階宣告式語言集中維護，這使得維護授權邏輯比在應用程式程式碼的多個區段中嵌入政策更為簡單。此方法也可確保一致地套用更新。
- 存取控制模型的彈性方法 – 角色型存取控制 (RBAC)、屬性型存取控制 (ABAC) 或兩種模型的組合都是存取控制的有效方法。這些模型嘗試使用不同的方法來滿足企業的授權要求。本指南會比較和對比這些模型，以協助您選擇適用於組織的模型。本指南也會討論這些模型如何套用至不同的授權政策語言，例如 OPA/Rego 和 Cedar。本指南中討論的架構可成功採用其中一個或兩個模型。
- 嚴格的 API 存取控制 – 本指南提供一種方法，可在應用程式中以最少的努力一致且普遍的方式保護 APIs。這對於通常使用大量 APIs 來促進應用程式內通訊的服務導向或微服務應用程式架構特別有用。嚴格的 API 存取控制有助於提高應用程式的安全性，並降低其遭受攻擊或利用的風險。

租戶隔離和多租戶授權

本指南參考租戶隔離和多租戶授權的概念。租用戶隔離是指您在 SaaS 系統中使用的明確機制，以確保即使每個租用戶在共用基礎設施上操作，其資源也會受到隔離。多租戶授權是指授權傳入動作，並防止在錯誤的租戶上實作這些動作。假設使用者可以進行身分驗證和授權，但仍可以存取另一個租用戶的資源。身分驗證和授權不會封鎖此存取，您需要實作租用戶隔離才能達成此目標。如需這兩個概念之間差異的更廣泛討論，請參閱 SaaS 架構基礎白皮書的租戶隔離一節。

存取控制的類型

您可以使用兩種廣泛定義的模型來實作存取控制：角色型存取控制 (RBAC) 和屬性型存取控制 (ABAC)。每個模型都有優點和缺點，本節會簡要討論這些優點和缺點。您應該使用的模型取決於您的特定使用案例。本指南中討論的架構支援兩種模型。

RBAC

角色型存取控制 (RBAC) 會根據通常符合商業邏輯的角色來決定對資源的存取。許可會視需要與角色建立關聯。例如，行銷角色會授權使用者在受限的系統中執行行銷活動。這是相對簡單的存取控制模型，因為它與易於識別的商業邏輯保持一致。

RBAC 模型在下列情況下效果較差：

- 您有唯一的使用者，其責任包含多個角色。
- 您有複雜的商業邏輯，讓角色難以定義。
- 擴展到大型需要持續管理和映射許可到新的和現有的角色。
- 授權是以動態參數為基礎。

ABAC

屬性型存取控制 (ABAC) 會根據屬性決定對資源的存取。屬性可以與使用者、資源、環境，甚至是應用程式狀態建立關聯。您的政策或規則參考屬性 和 可以使用基本布林邏輯來判斷是否允許使用者執行 動作。以下是許可的基本範例：

在付款系統中，財務部門的所有使用者都可以*`/payments`*在營業時間內在 API 端點處理付款。

財務部門的成員是使用者屬性，可決定對 的存取`/payments`。也有與 `/payments` API 端點相關聯的資源屬性，僅允許在上班時間存取。在 ABAC 中，使用者是否可以處理付款是由政策決定，政策將財務部門成員資格納入為使用者屬性，並將時間納入為 的資源屬性`/payments`。

ABAC 模型在允許動態、內容和精細的授權決策方面非常靈活。不過，一開始很難實作 ABAC 模型。定義規則和政策，以及列舉所有相關存取向量的屬性，需要大量預先投資才能實作。

RBAC-ABAC 混合方法

結合 RBAC 和 ABAC 可以提供這兩種模型的一些優點。RBAC 與商業邏輯非常一致，比 ABAC 更易於實作。若要在進行授權決策時提供額外的精細層級，您可以將 ABAC 與 RBAC 結合。此混合方法結合使用者的角色（及其指派的許可）和其他屬性來做出存取決策，藉此決定存取。使用這兩種模型可讓您輕鬆管理和指派許可，同時允許與授權決策相關的更高彈性和精細性。

存取控制模型比較

下表比較了先前討論的三種存取控制模型。此比較旨在提供資訊和高階。在特定情況下使用存取模型不一定與此表格中的比較相關。

因素	RBAC	ABAC	混合
彈性	中	高	高
簡單	高	低	中
精細程度	低	高	中
動態決策和規則	否	是	是
了解內容	否	是	有些
實作工作	低	高	中

實作 PDP

政策決策點 (PDP) 可以描述為政策或規則引擎。此元件負責套用政策或規則，並傳回是否允許特定存取的決定。PDP 可以搭配角色型存取控制 (RBAC) 和屬性型存取控制 (ABAC) 模型運作；不過，PDP 是 ABAC 的需求。PDP 允許將應用程式程式碼中的授權邏輯卸載至個別系統。這可以簡化應用程式程式碼。它還提供easy-to-use可重複界面，用於對 APIs、微服務、後端前端 (BFF) 層或任何其他應用程式元件進行授權決策。

以下各節討論實作 PDP 的三種方法。不過，這不是完整的清單。

PDP 實作方法：

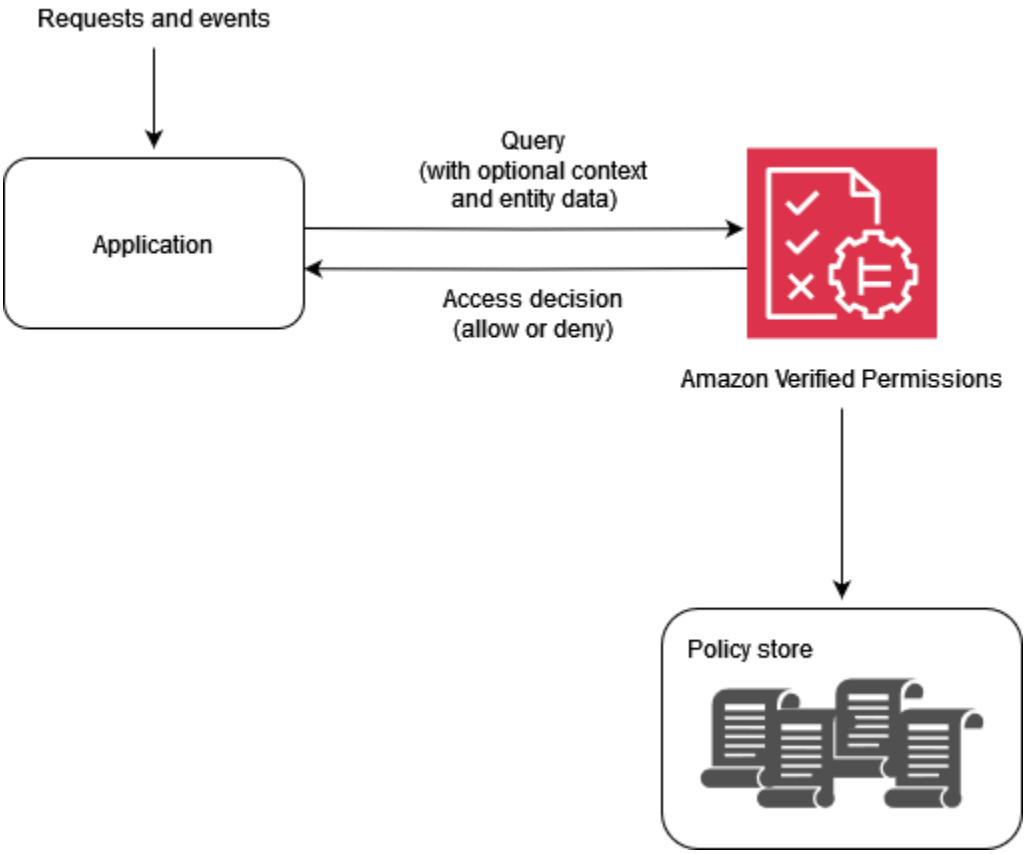
- 使用 Amazon Verified Permissions 實作 PDP
- 使用 OPA 實作 PDP
- 使用自訂政策引擎

使用 Amazon Verified Permissions 實作 PDP

Amazon Verified Permissions 是一種可擴展、精細的許可管理和授權服務，可用來實作政策決策點 (PDP)。作為政策引擎，它可以協助您的應用程式即時驗證使用者動作，並反白顯示過度特權或無效的許可。它透過外部化授權並集中管理政策，協助您的開發人員更快速地建置更安全的應用程式。透過將授權邏輯與應用程式邏輯分開，Verified Permissions 支援政策解耦。

透過使用 Verified Permissions 實作 PDP，並在應用程式中實作最低權限和持續驗證，開發人員可以使其應用程式存取符合零信任原則。此外，安全與稽核團隊可以更妥善地分析和稽核可存取應用程式中哪些資源的人員。Verified Permissions 使用 Cedar，這是一種專用且安全優先的開放原始碼政策語言，根據角色型存取控制 (RBAC) 和屬性型存取控制 (ABAC) 定義政策型存取控制，以實現更精細、內容感知的存取控制。

Verified Permissions 為 SaaS 應用程式提供一些有用的功能，例如能夠使用 Amazon Cognito、Google 和 Facebook 等多個身分提供者來啟用多租戶授權。另一個對 SaaS 應用程式特別有幫助的已驗證許可功能是支援每個租用戶的自訂角色。如果您正在設計客戶關係管理 (CRM) 系統，一個租戶可能會根據一組特定條件，依銷售機會定義存取的精細程度。另一個租用戶可能有另一個定義。Verified Permissions 中的基礎許可系統可以支援這些變化，這使得它成為 SaaS 使用案例的理想候選者。Verified Permissions 也支援撰寫適用於所有租用戶的政策，因此直接套用護欄政策以防止以 SaaS 供應商身分進行未經授權的存取。



為什麼要使用 Verified Permissions？

搭配 Amazon Cognito 等身分提供者使用 Verified Permissions，為您的應用程式提供更動態、以政策為基礎的存取管理解決方案。您可以建置應用程式，協助使用者共用資訊和協作，同時維護其資料的安全性、機密性和隱私權。Verified Permissions 為您提供精細的授權系統，以根據身分和資源的角色和屬性強制執行存取權，協助降低營運成本。您可以定義政策模型、在中央位置建立和存放政策，並以毫秒為單位評估存取請求。

在 Verified Permissions 中，您可以使用稱為 Cedar 的簡單、人類可讀宣告性語言來表達許可。無論每個團隊的應用程式使用何種程式設計語言，都可以在團隊之間共用以 Cedar 撰寫的政策。

使用 Verified Permissions 時應考量的事項

在 Verified Permissions 中，您可以建立政策並將其自動化，做為佈建的一部分。您也可以在執行時間建立政策，做為應用程式邏輯的一部分。最佳實務是，當您建立政策作為租戶加入和佈建的一部分時，您應該使用持續整合和持續部署 (CI/CD) 管道來管理、修改和追蹤政策版本。或者，應用程式可以管理、修改和追蹤政策版本；不過，應用程式邏輯本質上不會執行此功能。若要在您的應用程式中支援這些功能，您必須明確設計您的應用程式以實作此功能。

如果需要從其他來源提供外部資料以達到授權決策，則必須擷取此資料並將其提供給 Verified Permissions，作為授權請求的一部分。此服務預設不會擷取其他內容、實體和屬性。

Cedar 概觀

Cedar 是一種靈活、可擴展且可擴展的政策型存取控制語言，可協助開發人員將應用程式許可表達為政策。管理員和開發人員可以定義允許或禁止使用者對應用程式資源採取行動的政策。多個政策可以連接到單一資源。當您的應用程式使用者嘗試對資源執行動作時，您的應用程式會向 Cedar 政策引擎請求授權。Cedar 會評估適用的政策，並傳回 `ALLOW`或 `DENY`決策。Cedar 支援任何類型的主體和資源的授權規則，允許角色型存取控制 (RBAC) 和屬性型存取控制 (ABAC)，並支援透過自動化推理工具進行分析。

Cedar 可讓您將商業邏輯與授權邏輯分開。當您從應用程式的程式碼提出請求時，您可以呼叫 Cedar 的授權引擎來判斷請求是否獲得授權。如果授權 （ 決策為 `ALLOW`)，您的應用程式可以執行請求的操作。如果未授權 （ 決策為 `DENY`)，您的應用程式可能會傳回錯誤訊息。Cedar 的主要功能包括：

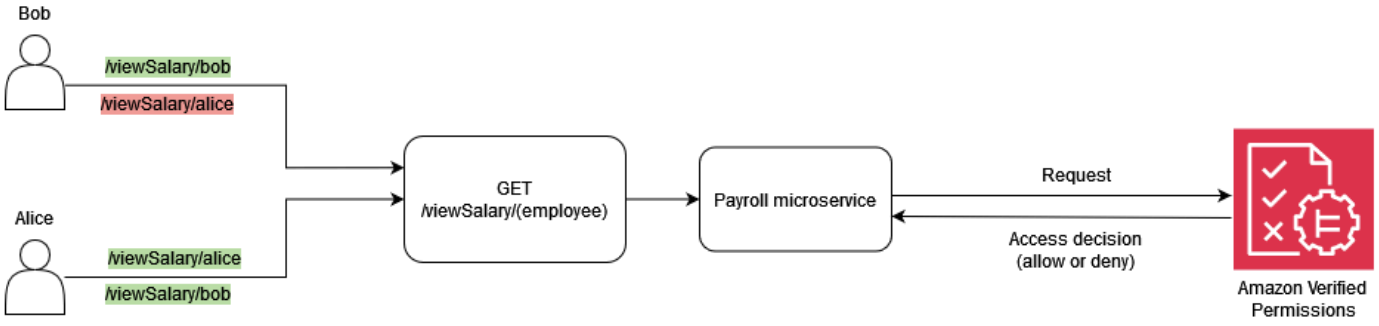
- 表達性 – Cedar 專為支援授權使用案例而打造，並且以人類可讀性為考量進行開發。
- 效能 – Cedar 支援索引政策以快速擷取，並提供快速且可擴展的即時評估與限制延遲。
- 分析 – Cedar 支援分析工具，可最佳化您的政策並驗證您的安全模型。

如需詳細資訊，請參閱 Cedar 網站。

範例 1：具有已驗證許可和 Cedar 的基本 ABAC

在此範例案例中，Amazon Verified Permissions 用於判斷哪些使用者可存取虛構薪資微服務中的資訊。本節包含 Cedar 程式碼片段，示範如何使用 Cedar 來呈現存取控制決策。這些範例並非旨在完整探索 Cedar 和 Verified Permissions 所提供的功能。如需 Cedar 的更完整概觀，請參閱 Cedar 文件。

在下圖中，我們希望強制執行與 `viewSalaryGET`方法相關聯的兩個一般業務規則：員工可以檢視自己的薪資，而員工可以檢視向其報告的任何人員的薪資。您可以使用 Verified Permissions 政策來強制執行這些業務規則。



員工可以檢視自己的薪資。

在 Cedar 中，基本建構是實體，代表委託人、動作或資源。若要提出授權請求並使用 Verified Permissions 政策開始評估，您需要提供委託人、動作、資源和實體清單。

- 主體 (principal) 是登入的使用者或角色。
- 動作 (action) 是由請求評估的操作。
- 資源 (resource) 是動作正在存取的元件。
- 實體清單 (entityList) 包含評估請求所需的所有必要實體。

為了滿足業務規則 員工可以檢視自己的薪資，您可以提供驗證許可政策，如下所示。

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

此政策會評估 Action ALLOW是否為 ，viewSalary且請求中的資源是否具有等於主體的屬性擁有者。例如，如果 Bob 是請求薪資報告的登入使用者，也是薪資報告的擁有者，則政策會評估為 ALLOW。

下列授權請求會提交至驗證許可，以供範例政策評估。在此範例中，Bob 是發出viewSalary請求的登入使用者。因此，Bob 是實體類型 的委託人Employee。Bob 嘗試執行的動作是 ，而 viewSalary 顯示viewSalary,的資源Salary-Bob是 類型Salary。為了評估 Bob 是否可以檢視Salary-Bob資源，您需要提供將 類型Employee與 值 Bob（委託人） 連結至類型為 之資源擁有者屬性Salary的實體結構。您可以在 中提供此結構entityList，其中與 相關聯的屬性Salary包含擁有者，指定包含類型Employee和值 entityIdentifier的 Bob。Verified Permissions 會將授權請求中principal提供的 與與Salary資源相關聯的 owner 屬性進行比較，以做出決策。

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

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

Verified Permissions 的授權請求會傳回下列項目做為輸出，其中 屬性`decision`為 `ALLOW`或 `DENY`。

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

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

在此情況下，由於 Bob 嘗試檢視自己的薪資，傳送至 Verified Permissions 的授權請求會評估為 ALLOW。不過，我們的目標是使用 Verified Permissions 來強制執行兩個業務規則。陳述下列項目的商業規則也應該是 true：

員工可以檢視向他們報告的任何人員的薪資。

若要滿足此業務規則，您可以提供另一個政策。下列政策會評估 動作ALLOW是否為 ，viewSalary且請求中的資源是否具有owner.manager等於委託人的屬性。例如，如果 Alice 是請求薪資報告的登入使用者，而 Alice 是報告擁有者的經理，則政策會評估為 ALLOW。

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

下列授權請求會提交至驗證許可，以供範例政策評估。在此範例中，Alice 是發出viewSalary請求的登入使用者。因此，Alice 是委託人，而實體的類型為 Employee。Alice 嘗試執行的動作是 viewSalary，而viewSalary顯示的資源類型為 Salary，值為 Salary-Bob。為了評估 Alice 是否可以檢視Salary-Bob資源，您需要提供將 類型Employee與 manager 值連結至 屬性Alice的實體結構，該結構接著必須與 類型 owner 值Salary為 的 屬性相關聯Salary-Bob。您可以在 中提供此結構entityList，其中與 相關聯的屬性Salary包含擁有者，指定包含類型Employee和值 entityIdentifier的 Bob。Verified Permissions 會先檢查 owner 屬性，該屬性會評估為 類型Employee和值 Bob。然後，Verified Permissions 會評估與 相關聯的manager屬性，Employee並將其與提供的委託人進行比較，以做出授權決策。在這種情況下，決策ALLOW是因為 principal 和 resource.owner.manager 屬性相等。

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

到目前為止，在此範例中，我們提供了與 `viewSalary`方法相關聯的兩個業務規則，員工可以檢視自己的薪資，而員工可以檢視向其報告之任何人的薪資，並將 Verified Permissions 視為政策，以獨立滿足每個業務規則的條件。您也可以使用單一已驗證許可政策來滿足這兩個業務規則的條件：

員工可以檢視自己的薪資以及向其報告的任何人的薪資。

當您使用先前的授權請求時，下列政策會評估 動作`ALLOW`是否為 ，`viewSalary`而請求中的資源是否具有等於 `owner.manager`的 屬性`principal`，或 屬性`owner`是否等於 `principal`。

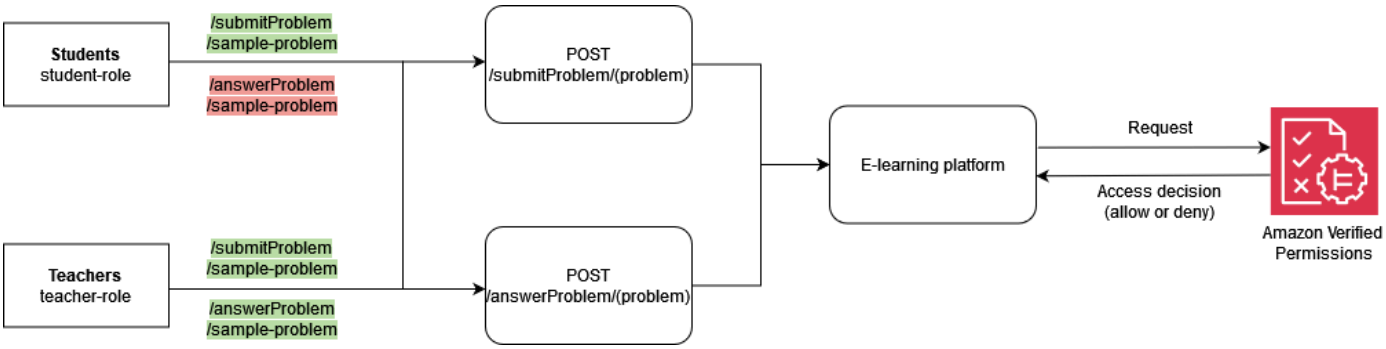
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

例如，如果 Alice 是請求薪資報告的登入使用者，而且 Alice 是擁有者的經理或報告的擁有者，則政策會評估為 `ALLOW`。

如需搭配 Cedar 政策使用邏輯運算子的詳細資訊，請參閱 Cedar 文件。

範例 2：具有已驗證許可和 Cedar 的基本 RBAC

此範例使用 Verified Permissions 和 Cedar 來示範基本 RBAC。如前所述，Cedar 的基本建構是實體。開發人員會定義自己的實體，也可以選擇性地在實體之間建立關係。下列範例包含三種類型的實體：Users、 Roles和 Problems。 Students 和 Teachers可視為 類型的實體，Role,且每個User實體都可以與零或任何 相關聯Roles。



在 Cedar 中，這些關係是透過將 Role Student 連結至 UserBob做為其父系來表示。此關聯會以邏輯方式將所有學生使用者分組在一個群組中。如需在 Cedar 中分組的詳細資訊，請參閱 Cedar 文件。

下列政策會針對連結至類型 Students之邏輯群組submitProblem,的所有委託人，評估 ALLOW動作的決策Role。

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

下列政策會針對連結至類型 Teachers之邏輯群組的所有委託人answerProblem，評估 ALLOW 動作submitProblem或 的決策Role。

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

為了使用這些政策評估請求，評估引擎需要知道授權請求中參考的委託人是否為適當群組的成員。因此，應用程式必須將相關的群組成員資格資訊傳遞給評估引擎，做為授權請求的一部分。這是透過 entities 屬性完成的，可讓您為 Cedar 評估引擎提供授權呼叫所涉及主體和資源的屬性和群組成員資格資料。在下列程式碼中，群組成員資格是透過User::"Bob"將 定義為具有名為 的父系來表示Role::"Students"。

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

```
      ]
  }
}
```

在此範例中，Bob 是發出answerProblem請求的登入使用者。因此，Bob 是委託人，而實體的類型為 User。Bob 嘗試執行的動作是 answerProblem。為了評估 Bob 是否可以執行answerProblem動作，您需要提供將實體User與 值連結的實體結構，Bob並透過將父實體列為 來指派其群組成員資格Role::"Students"。由於使用者群組中的實體只Role::"Students"允許執行動作submitProblem，因此此授權請求會評估為 DENY。

另一方面，如果 和 User 的類型Alice是群組Role::"Teachers" 嘗試執行answerProblem動作的一部分，授權請求會評估為 ALLOW，因為政策規定群組中的主體Role::"Teachers"可以answerProblem對所有資源執行動作。下列程式碼顯示評估為 的這類 授權請求ALLOW。

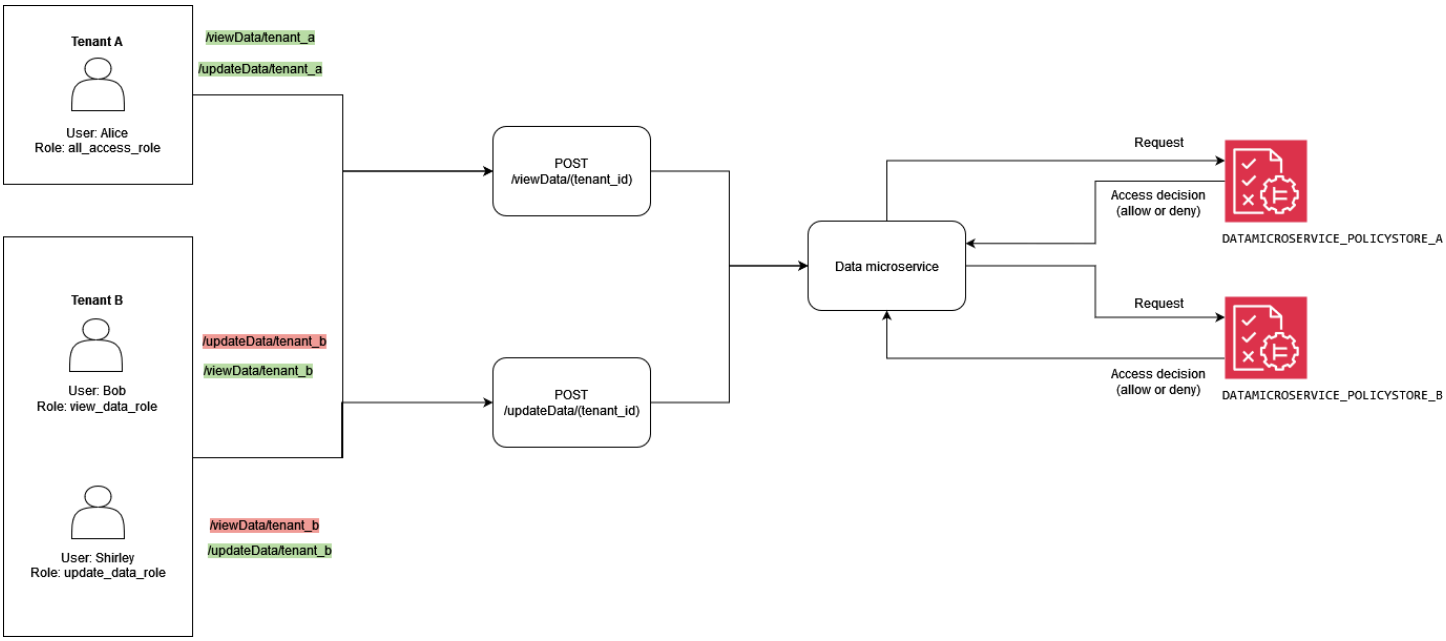
```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId":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

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

範例 3：使用 RBAC 進行多租戶存取控制

若要詳細說明先前的 RBAC 範例，您可以擴展需求以包含 SaaS 多租戶，這是 SaaS 供應商的常見需求。在多租用戶解決方案中，一律代表指定的租用戶提供資源存取。也就是說，租戶 A 的使用者無法檢視租戶 B 的資料，即使該資料在邏輯上或實際共置在系統中。下列範例說明如何使用多個 Verified Permissions 政策存放區實作租用戶隔離，以及如何使用使用者角色在租用戶內定義許可。

使用每個租用戶政策存放區設計模式是維護租用戶隔離，同時使用 Verified Permissions 實作存取控制的最佳實務。在此案例中，租戶 A 和租戶 B 使用者請求DATAMICROSERVICE_POLICYSTORE_B會分別針對個別的政策存放區 DATAMICROSERVICE_POLICYSTORE_A和 進行驗證。如需多租用戶 SaaS 應用程式之 Verified Permissions 設計考量事項的詳細資訊，請參閱 Verified Permissions 多租用戶設計考量事項一節。



下列政策位於DATAMICROSERVICE_POLICYSTORE_A政策存放區中。它會驗證委託人將是類型 allAccessRole群組的一部分Role。在這種情況下，將允許主體對與租用戶 A 相關聯的所有資源執行 viewData和 updateData動作。

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

下列政策位於DATAMICROSERVICE_POLICYSTORE_B政策存放區中。第一個政策會驗證委託人是類型 updateDataRole群組的一部分Role。假設是這種情況，它會授予主體許可，以對與租用戶 B 相關聯的資源執行 updateData動作。

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

第二個政策強制允許屬於 類型viewDataRole群組的主體對與租用戶 B 相關聯的資源Role執行 viewData動作。

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

從租用戶 A 發出的授權請求需要傳送到DATAMICROSERVICE_POLICYSTORE_A政策存放區，並由屬於該存放區的政策進行驗證。在這種情況下，它是由此範例之前討論的第一個政策進行驗證。在此授權請求中，值User為 的 類型主體Alice正在請求 執行viewData動作。委託人屬於類型 allAccessRole的群組Role。Alice 正在嘗試對SampleData資源執行 viewData動作。由於 Alice 具有 allAccessRole角色，因此此評估會產生 ALLOW決策。

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

如果您改為檢視 向租戶 B 提出的請求User Bob，您會看到類似以下授權請求的內容。請求會傳送到DATAMICROSERVICE_POLICYSTORE_B政策存放區，因為它源自租用戶 B。在此請求中，委託人Bob想要updateData對資源 執行 動作SampleData。不過， Bob 不是可存取updateData該資源上 動作的群組的一部分。因此，請求會產生DENY決策。

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

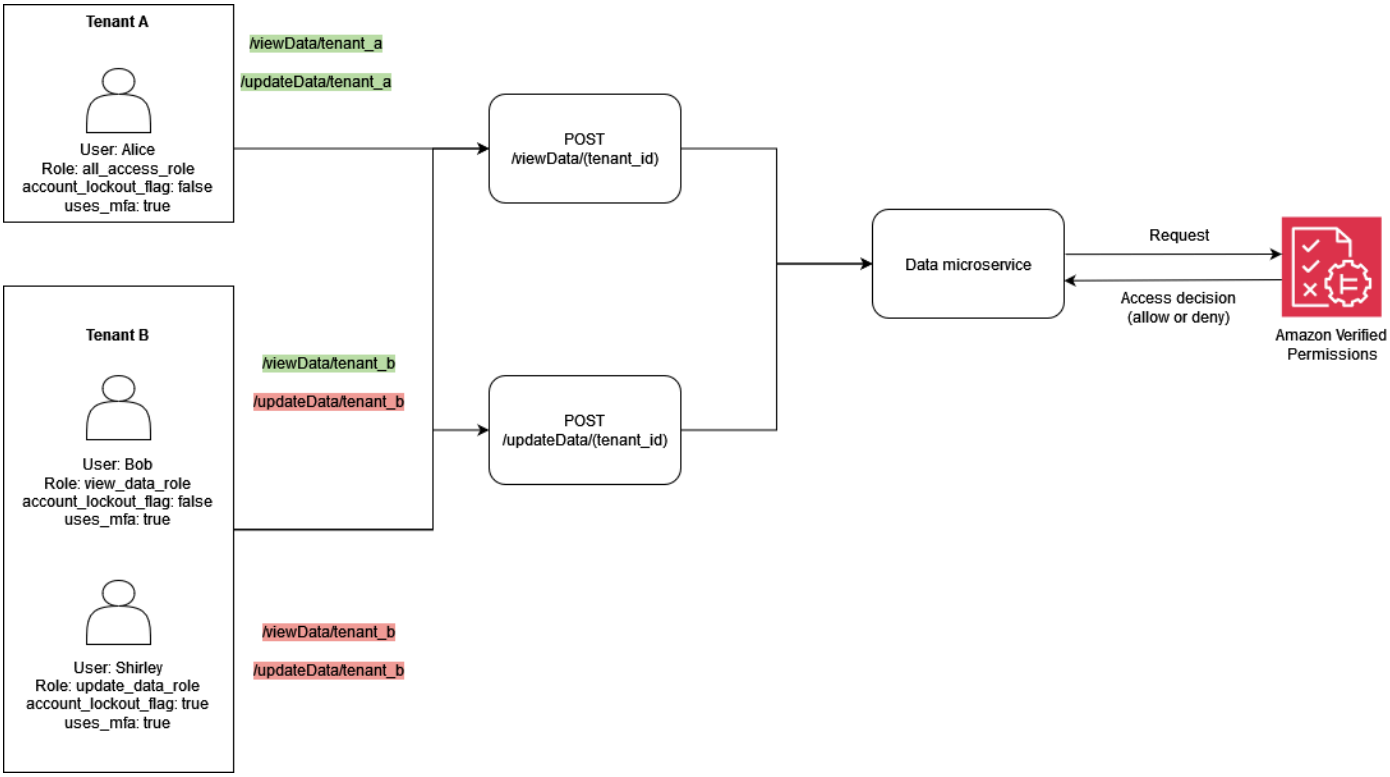
在此第三個範例中，User Alice 會嘗試對資源 執行 viewData動作SampleData。此請求會導向至DATAMICROSERVICE_POLICYSTORE_A政策存放區，因為主體Alice屬於租用戶 A。 Alice是

類型 allAccessRole 群組的一部分Role，允許她對 資源執行viewData動作。因此，請求會產生ALLOW決策。

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

範例 4：使用 RBAC 和 ABAC 進行多租戶存取控制

若要增強上一節中的 RBAC 範例，您可以將屬性新增至使用者，為多租戶存取控制建立 RBAC-ABAC 混合方法。此範例包含上一個範例中的相同角色，但會新增使用者屬性`account_lockout_flag`和內容參數 `uses_mfa`。此範例也會採用不同的方法來使用 RBAC 和 ABAC 實作多租用戶存取控制，並為每個租用戶使用一個共用政策存放區，而不是不同的政策存放區。



此範例代表多租用戶 SaaS 解決方案，您需要提供租用戶 A 和租用戶 B 的授權決策，類似於先前的範例。

若要實作使用者鎖定功能，範例會將 屬性新增至授權請求中的`account_lockout_flag`User實體主體。此旗標會鎖定使用者對系統的存取，並將鎖定使用者DENY的所有權限。`account_lockout_flag` 屬性與User實體相關聯，並對 有效，User直到跨多個工作階段主動撤銷旗標。此範例使用 when條件來評估 `account_lockout_flag`。

此範例也會新增請求和工作階段的詳細資訊。內容資訊指定已使用多重要素驗證來驗證工作階段。為了實作此驗證，範例會使用 when條件來評估 `uses_mfa`旗標做為內容欄位的一部分。如需新增內容的最佳實務的詳細資訊，請參閱 Cedar 文件。

```
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
};
```

此政策可防止存取 資源，除非資源與請求主體的 Tenant 屬性位於相同的群組中。這種維護租用戶隔離的方法稱為單一共享多租用戶政策存放區方法。如需多租用戶 SaaS 應用程式之 Verified Permissions 設計考量事項的詳細資訊，請參閱 Verified Permissions 多租用戶設計考量事項一節。

此政策也會確保委託人是 的成員allAccessRole，並將動作限制為 viewData和 updateData。此外，此政策會驗證 account_lockout_flag是 ，false且 的內容值會uses_mfa評估為 true。

同樣地，以下政策可確保委託人和資源都與相同的租用戶相關聯，以防止跨租用戶存取。此政策也會確保委託人是 的成員，viewDataRole並將動作限制為 viewData。此外，它會驗證 account_lockout_flag是 ，false且 的內容值會uses_mfa評估為 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
};
```

第三個政策類似於前一個政策。此政策要求資源成為 群組的成員，而該群組對應於由 表示的實體principal.Tenant。這可確保主體和資源都與租用戶 B 相關聯，以防止跨租用戶存取。此政策可確保委託人是 的成員，updateDataRole並將動作限制為 updateData。此外，此政策會驗證 account_lockout_flag是 ，false且 的內容值會uses_mfa評估為 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
};
```

本節稍早討論的三個政策會評估下列授權請求。在此授權請求中， 類型 User和 值為 的委託人會使用角色 Alice提出updateData請求allAccessRole。 Alice具有值為 Tenant的屬性Tenant::"TenantA"。Alice 嘗試執行的動作是 ，updateData,其將套用到的資源SampleData類型為 Data。 SampleData 具有 TenantA做為父實體。

根據政策存放區中的第一個<DATAMICROSERVICE_POLICYSTOREID>政策， Alice可以對資源執行updateData動作，假設符合政策 when子句中的條件。第一個條件需要 principal.Tenant 屬性才能評估為 TenantA。第二個條件需要主體的 屬性account_lockout_flag為 false。最終條件需要內容uses_mfa為 true。由於符合所有三個條件，請求會傳回 ALLOW決策。

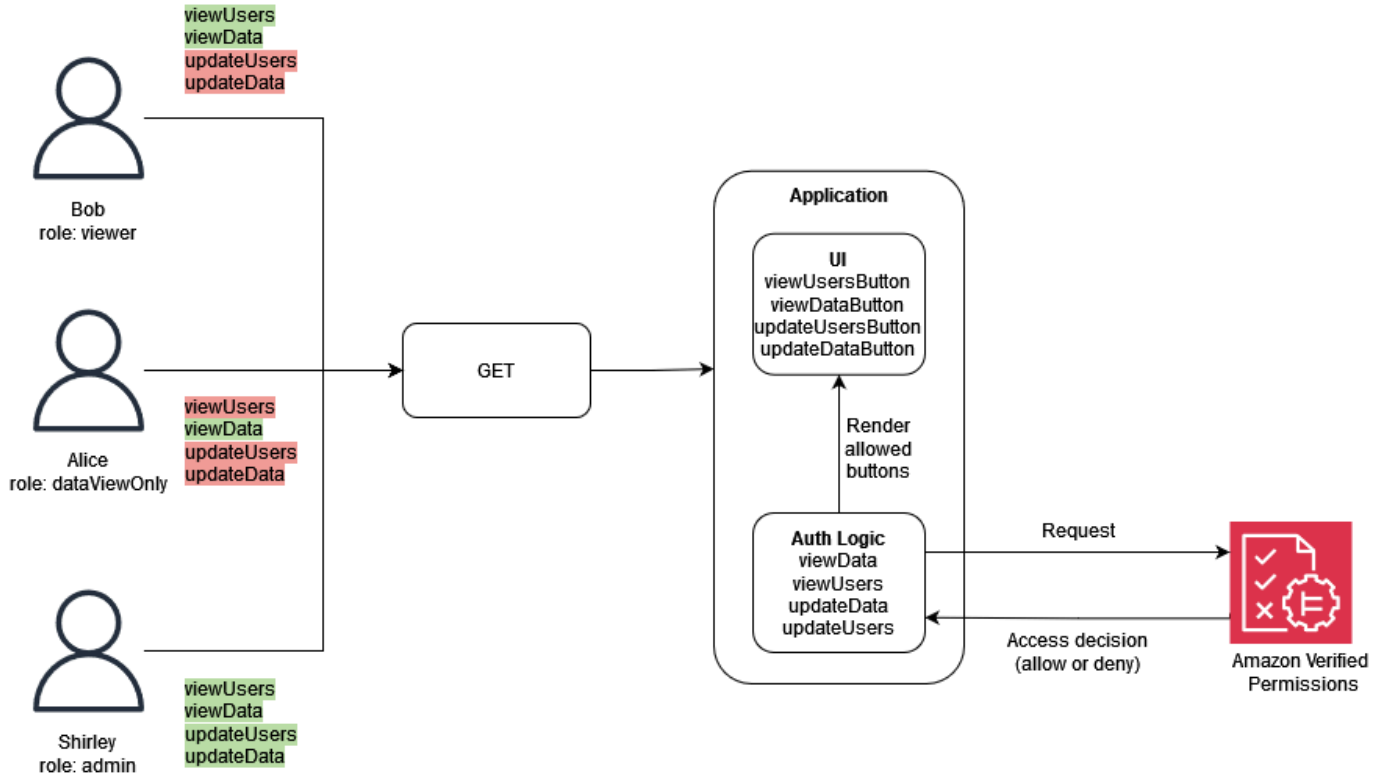
```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",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

範例 5：使用 Verified Permissions 和 Cedar 進行 UI 篩選

您也可以使用 Verified Permissions 根據授權的動作實作 UI 元素的 RBAC 篩選。這對於具有內容敏感 UI 元素的應用程式非常有用，這些元素可能會在多租用戶 SaaS 應用程式的情況下與特定使用者或租用戶相關聯。

在下列範例中，`Roleviewer`不允許 `Users` 的 執行更新。對於這些使用者，UI 不應轉譯任何更新按鈕。



在此範例中，單一頁面 Web 應用程式有四個按鈕。哪些按鈕是可見`Role`的，取決於目前登入應用程式的使用者的 。當單頁 Web 應用程式轉譯 UI 時，它會查詢 Verified Permissions，以判斷使用者獲授權執行的動作，然後根據授權決策產生按鈕。

下列政策指定值`Role`為 的 類型`viewer`可以同時檢視使用者和資料。此政策`ALLOW`的授權決策需要 `viewData`或 `viewUsers`動作，也需要資源與 類型`Data`或 相關聯`Users`。`ALLOW` 決策允許 UI 轉譯兩個按鈕： `viewDataButton`和 `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"]
};
```

下列政策指定值Role為 的 類型viewerDataOnly只能檢視資料。此政策ALLOW的授權決策需要 viewData動作，也需要資源與類型 建立關聯Data。ALLOW 決策允許 UI 轉譯按鈕 viewDataButton。

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

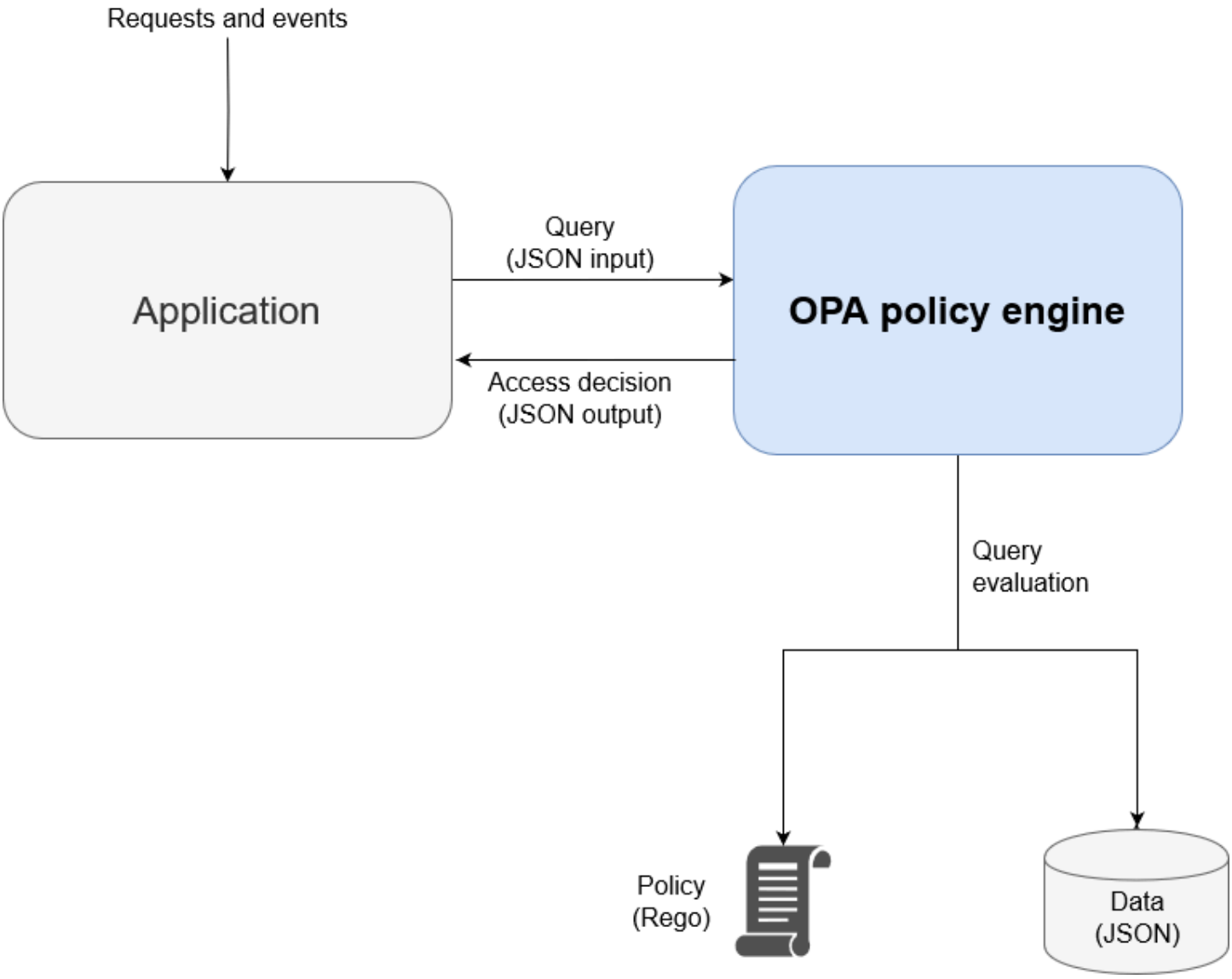
下列政策指定Role值為 的 類型admin可以編輯和檢視資料和使用者。此政策ALLOW的授權決策需要 updateData、 updateUsersviewData,或 的動作viewUsers，也需要資源與 類型Data或 相關聯Users。ALLOW 決策允許 UI 轉譯所有四個按鈕：updateDataButton、viewDataButton、updateUsersButton和 viewUsersButton。

```
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"]
};
```

使用 OPA 實作 PDP

Open Policy Agent (OPA) 是開放原始碼的一般用途政策引擎。OPA 有許多使用案例，但與 PDP 實作相關的使用案例是能夠從應用程式分離授權邏輯。這稱為政策解耦。OPA 在實作 PDP 方面很有用，原因有幾個。它使用稱為 Rego 的高階宣告性語言來草擬政策和規則。這些政策和規則與應用程式分開存在，可以呈現授權決策，而不需要任何應用程式特定的邏輯。OPA 也會公開 RESTful API，讓擷取授權決策變得簡單明瞭。若要做出授權決策，應用程式會使用 JSON 輸入查詢 OPA，而 OPA 會根據指

定的政策評估輸入，以傳回 JSON 中的存取決策。OPA 也能夠匯入可能與進行授權決策相關的外部資料。



相較於自訂政策引擎，OPA 有幾個優點：

- OPA 及其使用 Rego 進行的政策評估提供彈性的預先建置政策引擎，只需要插入政策和做出授權決策所需的任何資料。此政策評估邏輯必須在自訂政策引擎解決方案中重新建立。
- OPA 透過使用宣告性語言撰寫政策來簡化授權邏輯。您可以獨立於任何應用程式程式碼來修改和管理這些政策和規則，無需應用程式開發技能。
- OPA 公開 RESTful API，可簡化與政策強制執行點 (PEPs整合。
- OPA 提供內建支援，以驗證和解碼 JSON Web Token JWTs)。
- OPA 是公認的授權標準，這表示如果您需要協助或研究來解決特定問題，文件和範例會非常豐富。

- 採用授權標準，例如 OPA，可讓以 Rego 撰寫的政策跨團隊共用，無論團隊應用程式使用的程式設計語言為何。

OPA 不會自動提供兩件事：

- OPA 沒有用於更新和管理政策的強大控制平面。OPA 透過公開管理 API 來提供實作政策更新、監控和日誌彙總的一些基本模式，但與此 API 的整合必須由 OPA 使用者處理。根據最佳實務，您應該使用持續整合和持續部署 (CI/CD) 管道來管理、修改和追蹤政策版本，並在 OPA 中管理政策。
- 根據預設，OPA 無法從外部來源擷取資料。授權決策的外部資料來源可以是保留使用者屬性的資料庫。將外部資料提供給 OPA 的方式有一些彈性 – 在請求授權決策時，可以事先在本機快取，或從 API 動態擷取 – 但取得此資訊不是 OPA 可以代表您執行的動作。

撤銷概觀

Rego 是一種一般用途的政策語言，這表示它適用於堆疊的任何圖層和任何網域。Rego 的主要目的是接受評估的 JSON/YAML 輸入和資料，以對基礎設施資源、身分和操作做出已啟用政策的決策。Rego 可讓您撰寫有關堆疊或網域任何層的政策，而不需要變更或擴充語言。以下是 Rego 可以做出的一些決策範例：

- 是否允許或拒絕此 API 請求？
- 此應用程式的備份伺服器的主機名稱是什麼？
- 此提議基礎設施變更的風險分數是多少？
- 為了實現高可用性，此容器應該部署到哪些叢集？
- 此微服務應使用哪些路由資訊？

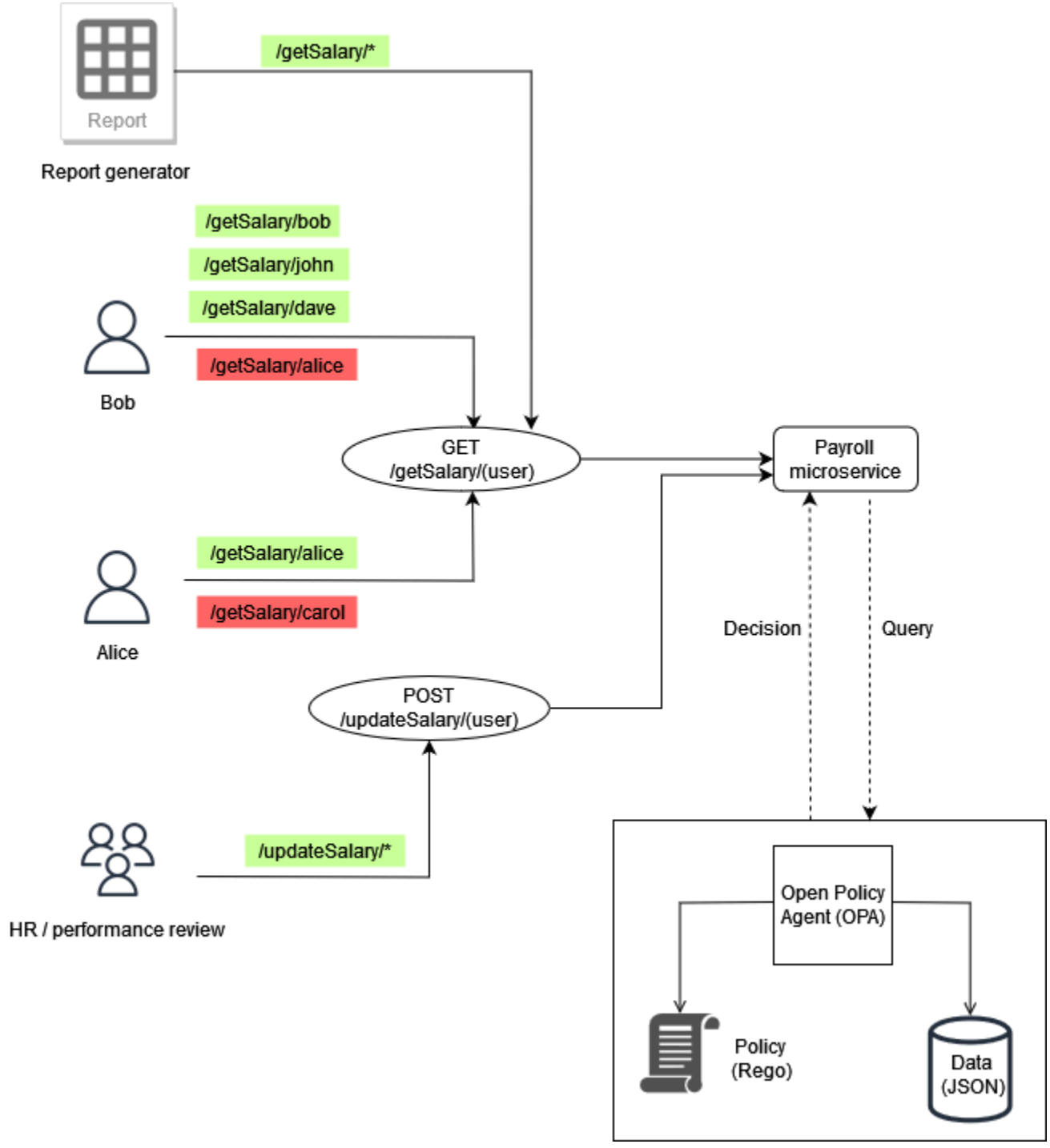
為了回答這些問題，Rego 採用如何做出這些決策的基本理念。在 Rego 中草擬政策時的兩個關鍵原則為：

- 每個資源、身分或操作都可以以 JSON 或 YAML 資料表示。
- 政策是套用至資料的邏輯。

Rego 透過定義如何評估 JSON/YAML 資料輸入的邏輯，協助軟體系統做出授權決策。C、Java、Go 和 Python 等程式設計語言是這個問題的常用解決方案，但 Rego 旨在專注於代表您系統的資料和輸入，以及使用此資訊做出政策決策的邏輯。

範例 1：具有 OPA 和 Rego 的基本 ABAC

本節說明使用 OPA 對哪些使用者允許存取虛構薪資微服務中的資訊進行存取決策的情況。提供 Rego 程式碼片段，示範如何使用 Rego 來呈現存取控制決策。這些範例既不會詳盡，也不會完整探索 Rego 和 OPA 功能。如需 Rego 的更完整概觀，建議您參閱 OPA 網站上的 Rego 文件。



基本 OPA 規則範例

在上圖中，OPA 針對薪資微服務強制執行的其中一個存取控制規則為：

員工可以讀取自己的薪資。

如果 Bob 嘗試存取薪資微服務以查看自己的薪資，薪資微服務可以將 API 呼叫重新導向至 OPA RESTful API 以做出存取決策。薪資服務會使用下列 JSON 輸入來查詢 OPA 是否有決策：

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

OPA 會根據查詢選取政策。在此情況下，以下以 Rego 撰寫的政策會評估 JSON 輸入。

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

此政策預設拒絕存取。然後，它會將查詢繫結至全域變數 來評估查詢中的輸入`input`。點運算子會與此變數搭配使用，以存取變數的值。如果規則中的表達式也是 true，則 Rego 規則會`allow`傳回 true。Rego 規則會驗證輸入`method` 中的 是否等於 GET。然後，它會驗證清單中的第一個元素`path` 是在將清單中的第二個元素指派給變數 `getSalary`之前`user`。最後，它會檢查要存取的路徑是否`/getSalary/bob` 與`user`提出請求 `input.user`符合`user`變數。規則`allow`套用 if-then 邏輯以傳回布林值，如輸出所示：

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

使用外部資料的部分規則

若要示範其他 OPA 功能，您可以將需求新增至您強制執行的存取規則。假設您想要在上圖的內容中強制執行此存取控制要求：

員工可以讀取向他們報告的任何人員的薪資。

在此範例中，OPA 可存取可匯入的外部資料，以協助做出存取決策：

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

您可以在 OPA 中建立部分規則來產生任意 JSON 回應，這會傳回一組值，而不是固定的回應。這是部分規則的範例：

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

此規則會傳回一組回報為 值的所有使用者`input.user`，在此情況下，該值為 bob。規則中的 `[_]` 建構用於反覆運算集合的值。這是規則的輸出：

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

擷取此資訊有助於判斷使用者是否為經理的直屬員工。對於某些應用程式，傳回動態 JSON 優於傳回簡單的布林值回應。

整合練習

最後一個存取需求比前兩個更複雜，因為它結合了這兩個需求中指定的條件：

員工可以讀取自己的薪資以及向其報告的任何人的薪資。

若要滿足此要求，您可以使用此 Rego 政策：

```
default allow = false

allow = true {
```

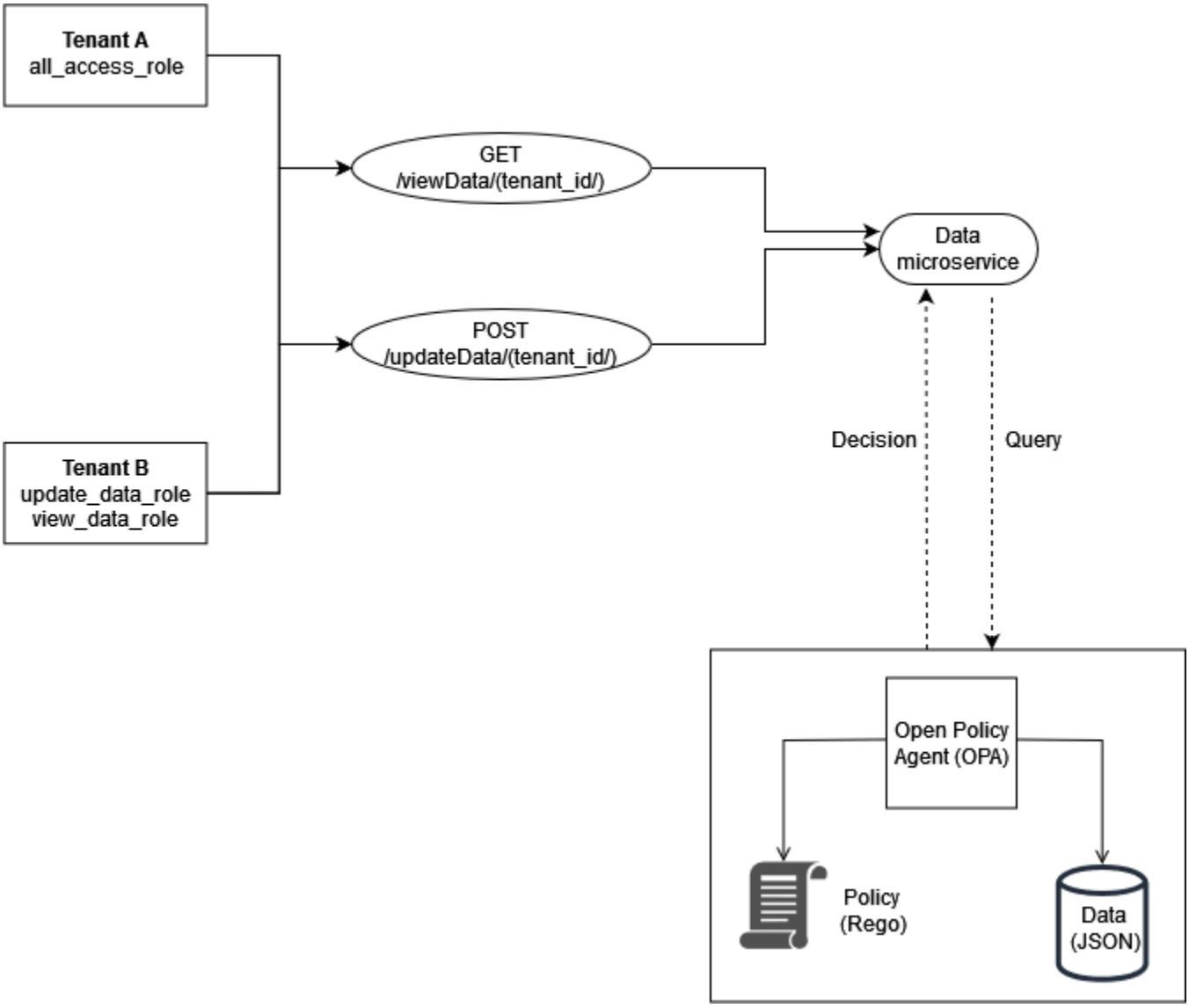
```
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

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

政策中的第一個規則允許任何嘗試查看自己薪資資訊的使用者存取 ，如前所述。在 Rego 中具有兩個名稱相同的規則，即 `allow`做為邏輯或運算子。第二個規則會擷取與 相關聯的所有直接報告清單 `input.user`（來自上圖中的資料），並將此清單指派給`managers`變數。最後，規則會驗證其名稱是否包含在`managers`變數中，`input.user`藉此檢查嘗試查看其薪資的使用者是否為 的直屬報告。

本節中的範例非常基本，不提供完整或徹底探索 Rego 和 OPA 的功能。如需詳細資訊，請檢閱 OPA 文件，請參閱 OPA GitHub README 檔案，然後在 Rego 遊樂場中實驗。

範例 2：具有 OPA 和 Rego 的多租戶存取控制和使用者定義的 RBAC

此範例使用 OPA 和 Rego 來示範如何使用租用戶使用者定義的自訂角色，在多租用戶應用程式的 API 上實作存取控制。它也會示範如何根據租用戶限制存取。此模型顯示 OPA 如何根據高階角色提供的資訊做出精細的許可決策。



租用戶的角色存放在外部資料 (RBAC 資料） 中，用於對 OPA 進行存取決策：

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

這些角色由租戶使用者定義時，應存放在外部資料來源或身分提供者 (IdP) 中，當將租戶定義的角色映射至許可和租戶本身時，可做為事實來源。

此範例使用 OPA 中的兩個政策來做出授權決策，並檢查這些政策如何強制執行租用戶隔離。這些政策使用先前定義的 RBAC 資料。

```
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")
}
```

若要顯示此規則的運作方式，請考慮具有下列輸入的 OPA 查詢：

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

此 API 呼叫的授權決策會結合 RBAC 資料、OPA 政策和 OPA 查詢輸入，如下所示：

1. 的使用者對 Tenant A進行 API 呼叫/viewData/tenant_a。
2. 資料微服務會接收呼叫並查詢allowViewData規則，傳遞 OPA 查詢輸入範例中顯示的輸入。
3. OPA 使用 OPA 政策中的查詢規則來評估提供的輸入。OPA 也會使用來自 RBAC 資料的資料來評估輸入。OPA 會執行下列動作：
   a. 驗證用於進行 API 呼叫的 方法為 GET。
   b. 驗證請求的路徑是否為 viewData。
   c. 檢查路徑tenant_id中的 是否等於與使用者input.tenant_id相關聯的 。這可確保保持租戶隔離。即使具有相同角色，另一個租戶也無法獲得進行此 API 呼叫的授權。
   d. 從角色的外部資料提取角色許可清單，並將其指派給變數 role_permissions。使用與 中的使用者相關聯的租戶定義角色來擷取此清單 input.role.
   e. 檢查role_permissions是否包含 許可 viewData.
4. OPA 會將下列決策傳回資料微服務：

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

此程序顯示 RBAC 和租戶意識如何有助於使用 OPA 做出授權決策。若要進一步說明這一點，請考慮/viewData/tenant_b使用下列查詢輸入對 進行 API 呼叫：

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

此規則會傳回與 OPA 查詢輸入相同的輸出，但適用於具有不同角色的不同租用戶。這是因為此呼叫適用於 ，/tenant_b且 RBAC 資料view_data_role中的 仍有與其相關聯的viewData許可。若要對強制執行相同類型的存取控制/updateData，您可以使用類似的 OPA 規則：

```
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")
}
```

此規則的功能與allowViewData規則相同，但會驗證不同的路徑和輸入方法。此規則仍會確保租戶隔離，並檢查租戶定義的角色是否授予 API 發起人許可。若要查看如何強制執行這項操作，請針對對 的 API 呼叫檢查下列查詢輸入/updateData/tenant_b：

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

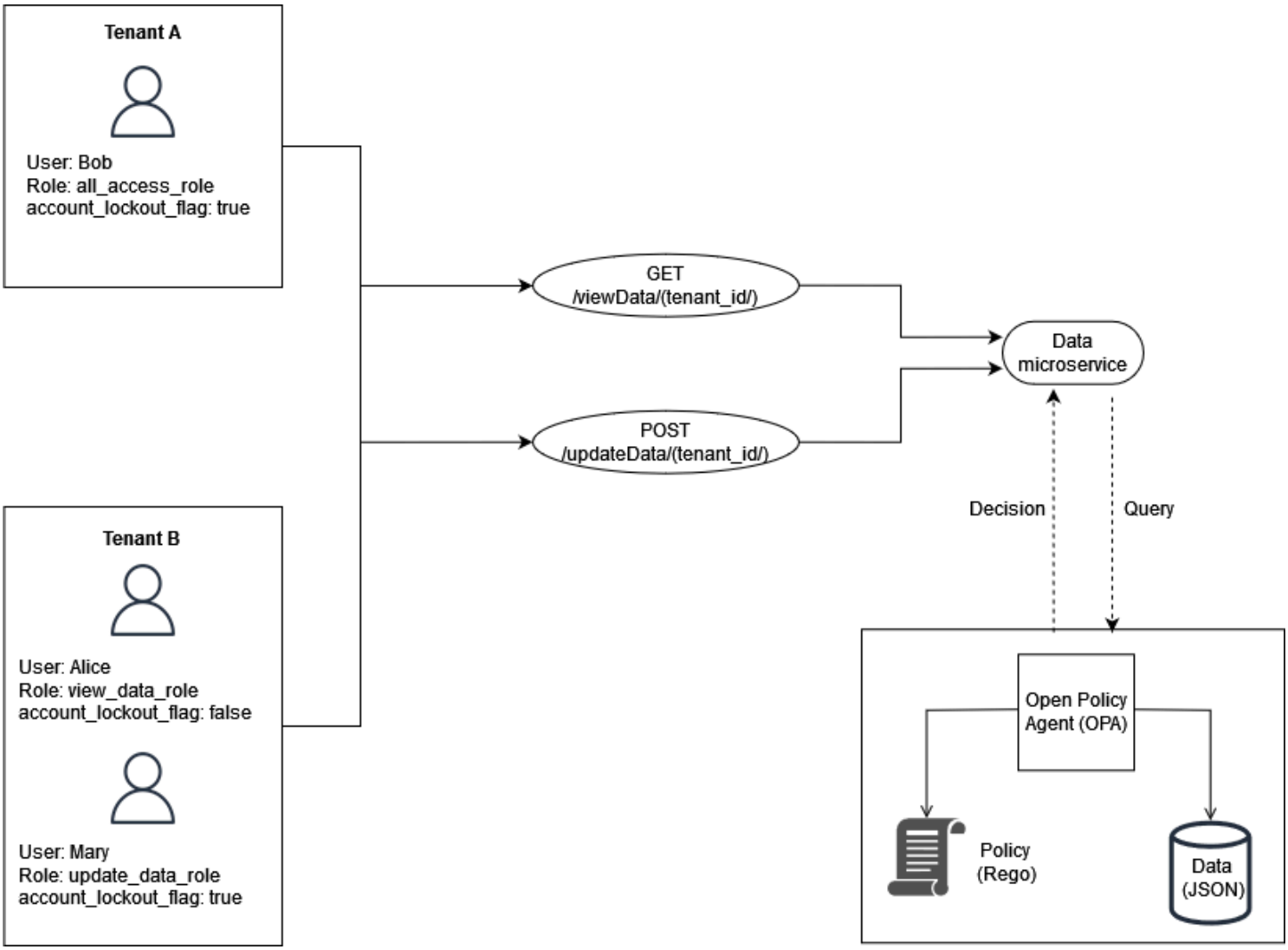
當使用 allowUpdateData規則評估時，此查詢輸入會傳回下列授權決策：

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

此呼叫將不會獲得授權。雖然 API 呼叫者與正確的 相關聯，tenant_id並且使用核准的方法呼叫 API，但 input.role是租戶定義的 view_data_role。view_data_role 沒有 updateData許可；因此，對 的呼叫/updateData未經授權。對於擁有 tenant_b的使用者，此呼叫將會成功update_data_role。

範例 3：具有 OPA 和 Rego 的 RBAC 和 ABAC 的多租戶存取控制

若要增強上一節中的 RBAC 範例，您可以將屬性新增至使用者。



此範例包含上一個範例中的相同角色，但會新增使用者屬性 account_lockout_flag。這是未與任何特定角色相關聯的使用者特定屬性。您可以使用先前用於此範例的相同 RBAC 外部資料：

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

account_lockout_flag 使用者屬性可做為使用者 Bob /viewData/tenant_a的 OPA 查詢輸入的一部分傳遞至資料服務：

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

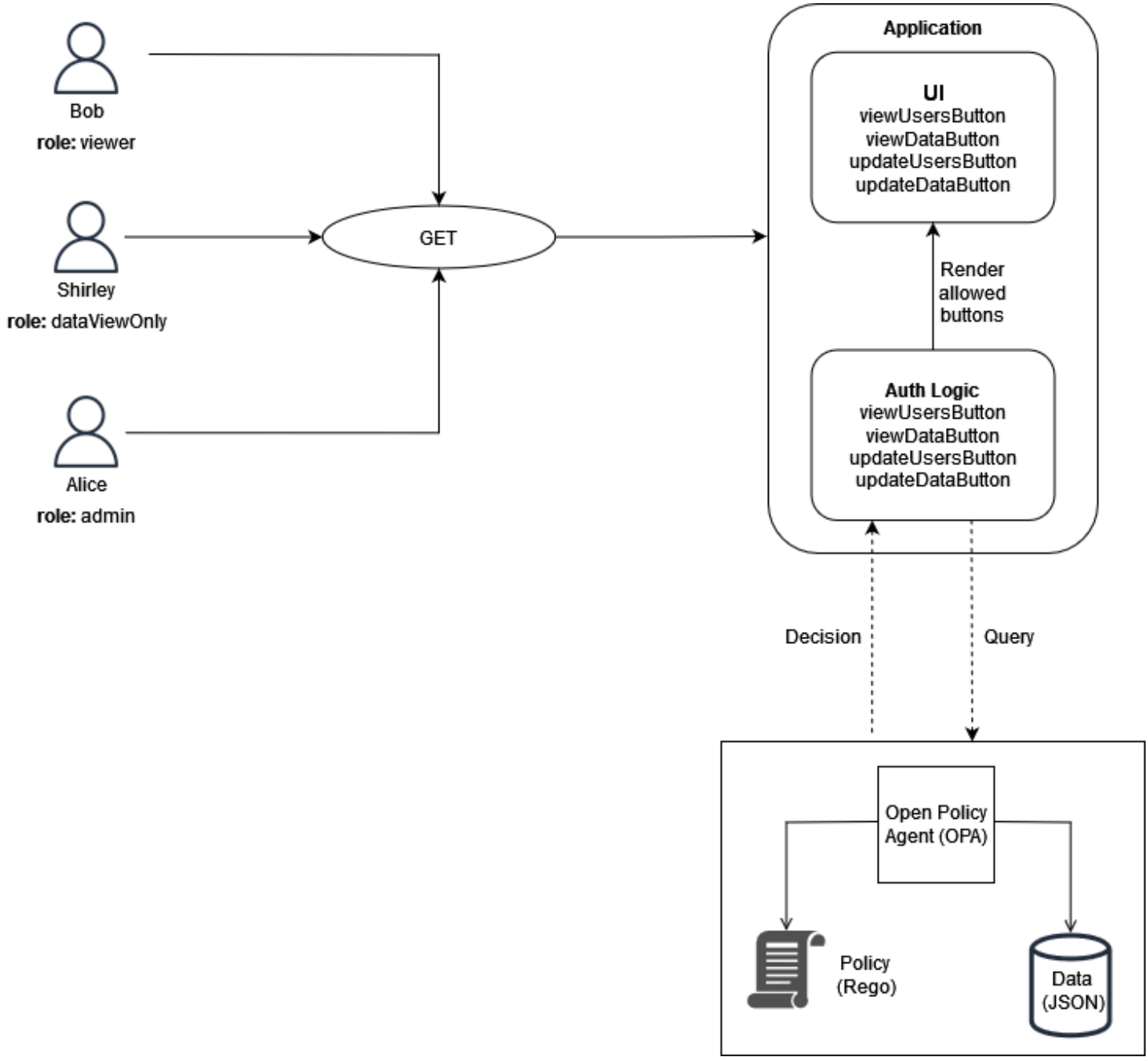
查詢存取決策的規則類似於上述範例，但包含要檢查account_lockout_flag屬性的額外行：

```
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"
}
```

此查詢會傳回 的授權決策false。這是因為 account_lockout_flag attributetrue適用於 Bob，而 Rego 規則allowViewData拒絕存取，雖然 Bob 具有正確的角色和租用戶。

範例 4：使用 OPA 和 Rego 進行 UI 篩選

OPA 和 Rego 的彈性支援篩選 UI 元素的能力。下列範例示範 OPA 部分規則如何針對 RBAC 的 UI 中應顯示的元素進行授權決策。此方法是使用 OPA 篩選 UI 元素的許多不同方法之一。



在此範例中，單一頁面 Web 應用程式有四個按鈕。假設您想要篩選 Bob、Shirley 和 Alice 的 UI，以便他們只能看到與其角色對應的按鈕。當 UI 收到來自使用者的請求時，它會查詢 OPA 部分規則，以判斷哪些按鈕應該顯示在 UI 中。當 Bob （使用角色 `viewer`) 向 UI 發出請求時，查詢會將以下內容做為輸入傳遞給 OPA：

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

OPA 使用 RBAC 結構化的外部資料來做出存取決策：

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

OPA 部分規則同時使用外部資料和輸入來產生允許的動作清單：

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

在部分規則中，OPA 會使用`input.role`指定的 做為查詢的一部分，來判斷應該顯示哪些按鈕。Bob 具有角色 `viewer`，而外部資料指定檢視器有兩個許可： `viewUsers`和 `viewData`。因此，Bob （以及具有檢視器角色的任何其他使用者） 此規則的輸出如下所示：

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

具有 `dataViewOnly`角色的 Shirley 輸出會包含許可按鈕：`viewData`。具有 `admin`角色的 Alice 輸出將包含所有這些許可。查詢 OPA 時，這些回應會傳回 UIuser_permissions。然後，應用程式可以使用此回應來隱藏或顯示 `viewUsersButton`、`updateUsersButton`、 `viewDataButton`和 `updateDataButton`。

使用自訂政策引擎

實作 PDP 的替代方法是建立自訂政策引擎。此政策引擎的目標是從應用程式分離授權邏輯。自訂政策引擎負責做出與 Verified Permissions 或 OPA 類似的授權決策，以實現政策解耦。此解決方案與使用

Verified Permissions 或 OPA 的主要區別在於，撰寫和評估政策的邏輯是針對自訂政策引擎而自訂建置的。與引擎的任何互動都必須透過 API 或其他方法公開，才能讓授權決策到達應用程式。您可以使用任何程式設計語言撰寫自訂政策引擎，或使用其他機制來評估政策，例如通用表達式語言 (CEL)。

實作 PEP

政策強制執行點 (PEP) 負責接收傳送至政策決策點 (PDP) 以供評估的授權請求。PEP 可以位於應用程式中的任何位置，其中資料和資源必須受到保護，或套用授權邏輯。與 PDPs 相比PEPs 相對簡單。PEP 僅負責請求和評估授權決策，不需要任何授權邏輯。與 PDPs不同，PEPs 無法集中在 SaaS 應用程式中。這是因為需要在整個應用程式及其存取點中實作授權和存取控制。PEPs可以套用至 APIs、微服務、後端前端 (BFF) 層，或應用程式中需要或需要存取控制的任何點。讓 PEPs 在應用程式中普及，可確保在多個點時經常獨立驗證授權。

若要實作 PEP，第一個步驟是判斷應用程式中應執行存取控制的位置。在決定應將哪些 PEPs 整合到您的應用程式中時，請考慮此原則：

如果應用程式公開 API，則該 API 應具備授權和存取控制。

這是因為在微服務導向或服務導向架構中，APIs可做為不同應用程式函數之間的分隔符號。將存取控制作為應用程式函數之間的邏輯檢查點是合理的。我們強烈建議您納入 PEPs做為存取 SaaS 應用程式中每個 API 的先決條件。您也可以在應用程式中的其他時間點整合授權。在單體應用程式中，可能需要將 PEPs整合到應用程式本身的邏輯中。沒有應該包含 PEPs 的單一位置，但請考慮使用 API 原則做為起點。

請求授權決策

PEP 必須向 PDP 請求授權決策。請求可以採用多種形式。請求授權決策最簡單且最容易存取的方法，是將授權請求或查詢傳送至 PDP (OPA 或驗證許可） 公開的 RESTful API。如果您使用的是 Verified Permissions，您也可以使用 AWS SDK 來擷取授權決策，以呼叫 IsAuthorized 方法。此模式中 PEP 的唯一函數是轉送授權請求或查詢所需的資訊。這可以像將 API 收到的請求轉送為 PDP 輸入一樣簡單。還有其他建立 PEPs 的方法。例如，您可以將 OPA PDP 在本機與以 Go 程式設計語言撰寫的應用程式整合為程式庫，而不是使用 API。

評估授權決策

PEPs 需要包含邏輯來評估授權決策的結果。當 PDPs公開為 APIs時，授權決策可能採用 JSON 格式，並由 API 呼叫傳回。PEP 必須評估此 JSON 程式碼，以判斷所採取的動作是否已獲得授權。例如，如果 PDP 旨在提供布林值允許或拒絕授權決策，則 PEP 可能只檢查此值，然後傳回 HTTP 狀態碼 200 表示允許，傳回 HTTP 狀態碼 403 表示拒絕。這種將 PEP 整合為存取 API 的先決條件的模式，是在 SaaS 應用程式中實作存取控制的輕鬆實作和高效模式。在更複雜的案例中，PEP 可能負責評估 PDP 傳回的任意 JSON 程式碼。必須寫入 PEP，以包含解釋 PDP 傳回的授權決策所需的任何邏

輯。由於 PEP 可能在您應用程式中的許多不同位置實作，我們建議您將 PEP 程式碼封裝為可重複使用的程式庫或您選擇的程式設計語言成品。如此一來，您的 PEP 就可以輕鬆地在應用程式中的任何時間點整合，只需最少的重新作業。

多租戶 SaaS 架構的設計模型

實作 API 存取控制和授權的方法有很多種。本指南著重於適用於多租戶 SaaS 架構的三種設計模型。這些設計可做為政策決策點 (PDPs) 和政策強制執行點 (PEPs) 實作的高階參考，為應用程式形成一致且普遍的授權模型。

設計模型：

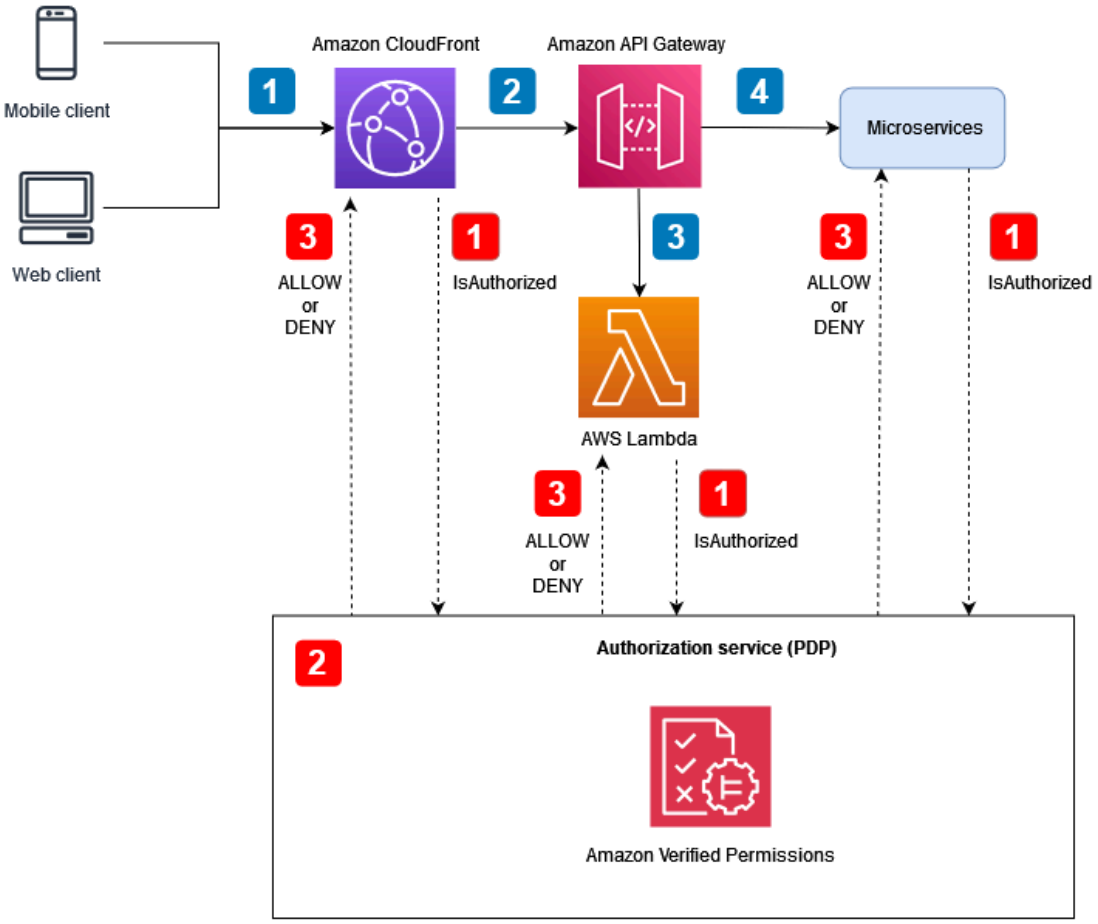
- Amazon Verified Permissions 的設計模型
- OPA 的設計模型

Amazon Verified Permissions 的設計模型

在 APIs 上使用集中式 PDP 搭配 PEPs

API 模型上具有政策強制執行點 PEPs) 的集中式政策決策點 (PDP) 遵循產業最佳實務，為 API 存取控制和授權建立有效且易於維護的系統。 APIs 此方法支援數個關鍵原則：

- 授權和 API 存取控制會套用在應用程式中的多個點。
- 授權邏輯與應用程式無關。
- 存取控制決策是集中式的。



應用程式流程（圖表中以藍色編號的標註表示）：

1. 使用 JSON Web Token (JWT) 的已驗證使用者會向 Amazon CloudFront 產生 HTTP 請求。
2. CloudFront 會將請求轉送至設定為 CloudFront 原始伺服器的 Amazon API Gateway。
3. 呼叫 API Gateway 自訂授權方來驗證 JWT。
4. Microservices 會回應請求。

授權和 API 存取控制流程（在圖表中以紅色編號的標註表示）：

1. PEP 會呼叫授權服務並傳遞請求資料，包括任何 JWTs。
2. 授權服務 (PDP) 在此情況下， 會使用請求資料做為查詢輸入，並根據查詢指定的相關政策進行評估。
3. 授權決策會傳回給 PEP 並進行評估。

此模型使用集中式 PDP 來做出授權決策。PEPs會在不同的時間點實作，以向 PDP 提出授權請求。下圖顯示如何在假設的多租用戶 SaaS 應用程式中實作此模型。

在此架構中，PEPs 會在 Amazon CloudFront 和 Amazon API Gateway 的服務端點以及每個微服務請求授權決策。授權決策是由授權服務 Amazon Verified Permissions (PDP) 做出。由於 Verified Permissions 是全受管服務，因此您不需要管理基礎基礎設施。您可以使用 RESTful API 或 AWS SDK 與 Verified Permissions 互動。

您也可以將此架構與自訂政策引擎搭配使用。不過，從 Verified Permissions 獲得的任何優勢都必須取代為自訂政策引擎提供的邏輯。

API 上具有 PEPs APIs集中式 PDP 提供簡單的選項，可建立 APIs的強大授權系統。這可簡化授權程序，並提供easy-to-use可重複界面，以針對 APIs、微服務、後端前端 (BFF) 層或其他應用程式元件做出授權決策。

使用 Cedar SDK

Amazon Verified Permissions 使用 Cedar 語言來管理自訂應用程式中的精細許可。使用 Verified Permissions，您可以將 Cedar 政策存放在中央位置、利用毫秒處理的低延遲，以及跨不同應用程式的稽核許可。您也可以選擇性地將 Cedar SDK 直接整合到您的應用程式中，以提供授權決策，而無需使用 Verified Permissions。此選項需要額外的自訂應用程式開發，才能管理和存放使用案例的政策。不過，這可能是可行的替代方案，特別是在因網際網路連線不一致而存取 Verified Permissions 是間歇性或不可行的情況下。

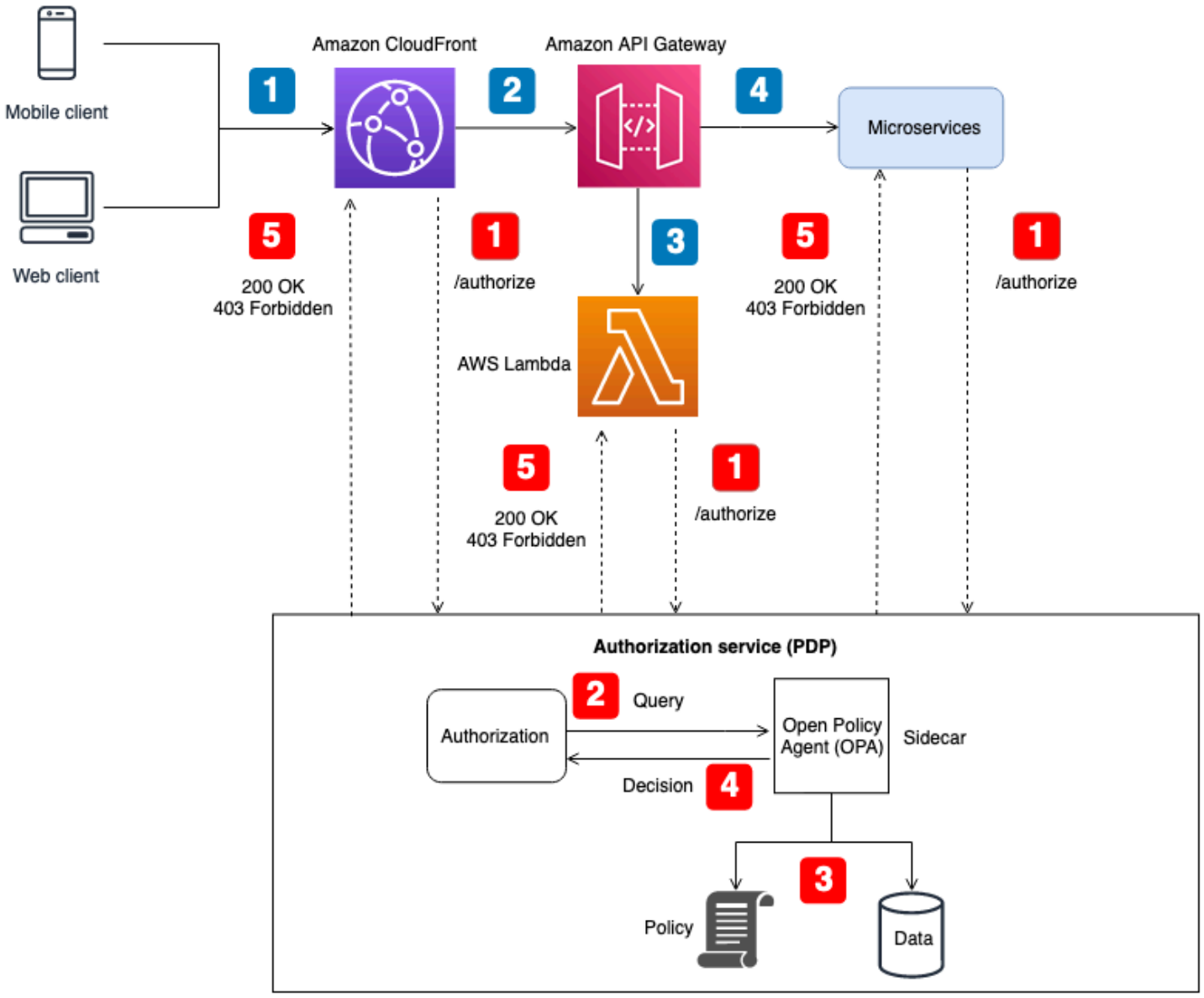
OPA 的設計模型

在 APIs 上使用集中式 PDP 搭配 PEPs

API 模型上具有政策強制執行點 PEPs) 的集中式政策決策點 (PDP) 遵循產業最佳實務，為 API 存取控制和授權建立有效且易於維護的系統。 APIs 此方法支援數個關鍵原則：

- 授權和 API 存取控制會套用在應用程式中的多個點。
- 授權邏輯與應用程式無關。
- 存取控制決策是集中式的。

此模型使用集中式 PDP 來做出授權決策。所有 APIs PEPs，以向 PDP 提出授權請求。下圖顯示如何在假設的多租用戶 SaaS 應用程式中實作此模型。



應用程式流程（圖表中以藍色編號的標註表示）：

1. 具有 JWT 的已驗證使用者會向 Amazon CloudFront 產生 HTTP 請求。
2. CloudFront 會將請求轉送至設定為 CloudFront 原始伺服器的 Amazon API Gateway。
3. 呼叫 API Gateway 自訂授權方來驗證 JWT。
4. Microservices 會回應請求。

授權和 API 存取控制流程（在圖表中以紅色編號的標註表示）：

1. PEP 會呼叫授權服務並傳遞請求資料，包括任何 JWTs。

2. 授權服務 (PDP) 會取得請求資料，並查詢以附屬身分執行的 OPA 代理程式 REST API。請求資料做為查詢的輸入。
3. OPA 會根據查詢指定的相關政策來評估輸入。如有必要，系統會匯入資料以做出授權決策。
4. OPA 會將決策傳回給授權服務。
5. 授權決策會傳回給 PEP 並進行評估。

在此架構中，PEPs 會在 Amazon CloudFront 和 Amazon API Gateway 的服務端點，以及每個微服務請求授權決策。授權決策是由具有 OPA 附屬的授權服務 (PDP) 做出。您可以將此授權服務做為容器或傳統伺服器執行個體操作。OPA 附屬項目會在本機公開其 RESTful API，因此 API 只能由授權服務存取。授權服務會公開可供 PEPs 使用的個別 API。讓授權服務做為 PEPs 和 OPA 之間的媒介，允許在 PEPs 和 OPA 之間插入可能需要的任何轉換邏輯，例如，當來自 PEP 的授權請求不符合 OPA 預期的查詢輸入時。

您也可以將此架構與自訂政策引擎搭配使用。不過，從 OPA 獲得的任何優勢都必須取代為自訂政策引擎提供的邏輯。

API 上具有 PEPs APIs集中式 PDP 提供簡單的選項，可建立 APIs的強大授權系統。它易於實作，也提供easy-to-use可重複界面，用於對 APIs、微服務、後端前端 (BFF) 層或其他應用程式元件進行授權決策。不過，這種方法可能會在您的應用程式中產生過多延遲，因為授權決策需要呼叫單獨的 API。如果網路延遲有問題，您可能會考慮分散式 PDP。

在 APIs 上使用分散式 PDP 搭配 PEPs

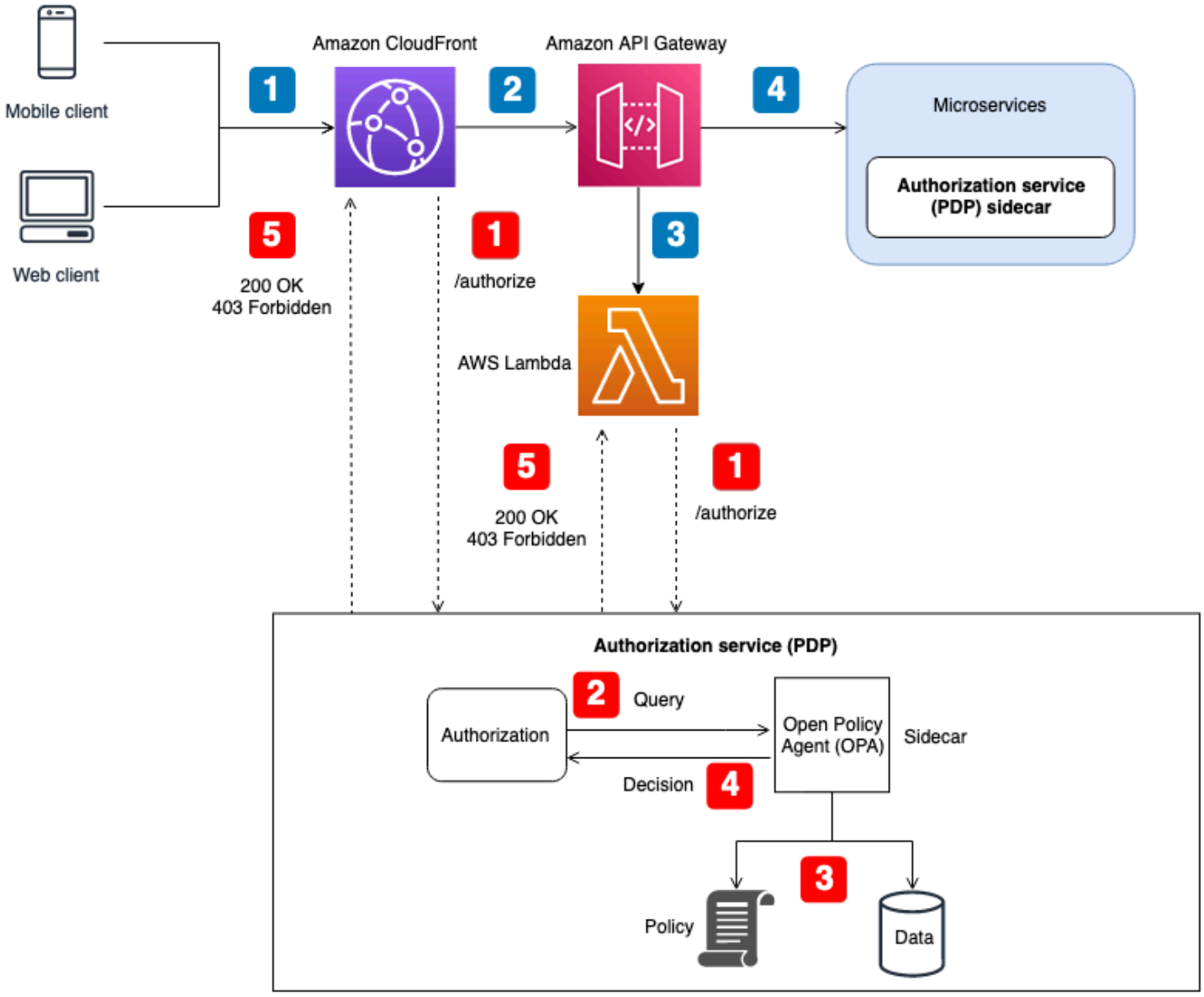
API 模型上具有政策強制執行點 PEPs) 的分散式政策決策點 (PDP) 遵循產業最佳實務，為 API 存取控制和授權建立有效的系統。 APIs 如同 API 模型上具有 PEPs 的集中式 PDP，此方法支援下列關鍵原則： APIs

- 授權和 API 存取控制會套用在應用程式中的多個點。
- 授權邏輯與應用程式無關。
- 存取控制決策是集中式的。

您可能會想知道為什麼在分佈 PDP 時，存取控制決策會集中化。雖然 PDP 可能存在於應用程式中的多個位置，但必須使用相同的授權邏輯來做出存取控制決策。所有 PDPs都會根據相同的輸入提供相同的存取控制決策。所有 APIs PEPs，以向 PDP 提出授權請求。下圖顯示如何在假設的多租戶 SaaS 應用程式中實作此分散式模型。

在此方法中，PDPs會在應用程式中的多個位置實作。對於具有可以執行 OPA 並支援 PDP 的應用程式元件，例如具有附屬或 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體的容器化服務，PDP 決策可以直接整合到應用程式元件中，而無需對集中式 PDP 服務進行 RESTful API 呼叫。這有助於減少您在集中式 PDP 模型中可能遇到的延遲，因為不是每個應用程式元件都必須進行額外的 API 呼叫，以取得授權決策。不過，對於沒有能夠直接整合 PDP 的應用程式元件，例如 Amazon CloudFront 或 Amazon API Gateway 服務，此模型仍然需要集中式 PDP。

下圖顯示如何在假設的多租用戶 SaaS 應用程式中實作此集中式 PDP 和分散式 PDP 的組合。



應用程式流程（圖表中以藍色編號的標註表示）：

1. 具有 JWT 的已驗證使用者會向 Amazon CloudFront 產生 HTTP 請求。

2. CloudFront 會將請求轉送至設定為 CloudFront 原始伺服器的 Amazon API Gateway。
3. API Gateway 自訂授權方會呼叫 來驗證 JWT。
4. Microservices 會回應請求。

授權和 API 存取控制流程（在圖表中以紅色編號的標註表示）：

1. PEP 會呼叫授權服務並傳遞請求資料，包括任何 JWTs。
2. 授權服務 (PDP) 會取得請求資料並查詢以附屬身分執行的 OPA 代理程式 REST API。請求資料做為查詢的輸入。
3. OPA 會根據查詢指定的相關政策來評估輸入。如有必要，系統會匯入資料以做出授權決策。
4. OPA 會將決策傳回給授權服務。
5. 授權決策會傳回給 PEP 並進行評估。

在此架構中，PEPs 會在 CloudFront 和 API Gateway 的服務端點，以及每個微服務請求授權決策。微服務的授權決策是由授權服務 (PDP) 所決定，該服務可做為應用程式元件的附屬操作。此模型適用於在容器或 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體上執行的微服務（或服務）。API Gateway 和 CloudFront 等服務的授權決策仍需要聯絡外部授權服務。無論如何，授權服務都會公開可供 PEPs 使用的 API。讓授權服務做為 PEPs 和 OPA 之間的媒介，允許在 PEPs 和 OPA 之間插入可能需要的任何轉換邏輯，例如，當來自 PEP 的授權請求不符合 OPA 預期的查詢輸入時。

您也可以將此架構與自訂政策引擎搭配使用。不過，從 OPA 獲得的任何優勢都必須取代為自訂政策引擎提供的邏輯。

API 上具有 PEPsAPIs分散式 PDP 提供為 APIs 建立強大授權系統的選項。它易於實作，並提供easy-to-use可重複界面，用於對 APIs、微服務、後端前端 (BFF) 層或其他應用程式元件進行授權決策。此方法也有助於減少您在集中式 PDP 模型中可能遇到的延遲。

使用分散式 PDP 做為程式庫

您也可以向 PDP 請求授權決策，該 PDP 以程式庫或套件的形式提供，以在應用程式中使用。OPA 可以用作 Go 第三方程式庫。對於其他程式設計語言，採用此模型通常表示您必須建立自訂政策引擎。

Amazon Verified Permissions 多租戶設計考量事項

當您在多租用戶 SaaS 解決方案中使用 Amazon Verified Permissions 實作授權時，需要考慮幾個設計選項。在探索這些選項之前，讓我們釐清多租用戶 SaaS 內容中隔離和授權之間的差異。隔離租用戶可防止傳入和傳出資料公開給錯誤的租用戶。授權可確保使用者具有存取租用戶的許可。

在 Verified Permissions 中，政策會存放在政策存放區中。如 Verified Permissions 文件中所述，您可以針對每個租用戶使用個別的政策存放區來隔離租用戶的政策，或針對所有租用戶使用單一政策存放區來允許租用戶共用政策。本節討論這兩種隔離策略的優點和缺點，並說明如何使用分層部署模型進行部署。如需其他內容，請參閱 Verified Permissions 文件。

雖然本節討論的 critieria 著重於 Verified Permissions，但一般概念是以隔離思維及其提供的指引為基礎。SaaS 應用程式必須一律將租戶隔離視為其設計的一部分，而此一般隔離原則延伸到在 SaaS 應用程式中包含驗證許可。本節也參考核心 SaaS 隔離模型，例如孤立 SaaS 模型和集區 SaaS 模型。如需詳細資訊，請參閱 AWS Well-Architected Framework SaaS Lens 中的核心隔離概念。

設計多租戶 SaaS 解決方案時的主要考量事項是租戶隔離和租戶加入。租用戶隔離會影響安全性、隱私權、彈性和效能。租戶加入會影響您的營運程序，因為它與營運開銷和可觀測性相關。經歷 SaaS 旅程或實作多租戶解決方案的組織，一律必須優先考慮 SaaS 應用程式處理租用的方式。雖然 SaaS 解決方案可能傾向於特定隔離模型，但在整個 SaaS 解決方案中不一定需要一致性。例如，您為應用程式的前端元件選擇的 隔離模型可能與您為微型服務或授權服務選擇的隔離模型不同。

設計考量事項：

- 租戶加入和使用者租戶註冊
- 每個租戶政策存放區
- 一個共用多租戶政策存放區
- 分層部署模型

租戶加入和使用者租戶註冊

SaaS 應用程式會觀察 SaaS 身分的概念，並遵循將使用者身分繫結至租用戶身分的一般最佳實務。繫結涉及將租戶識別符儲存為身分提供者中使用者的宣告或屬性。這會轉移將身分映射到租用戶的責任，從每個應用程式映射到使用者註冊程序。然後，每個已驗證的使用者都會擁有正確的租戶身分，做為 JSON Web Token (JWT) 的一部分。

同樣地，為授權請求選擇正確的政策存放區不應由應用程式邏輯決定。若要判斷特定授權請求應使用的政策存放區，請維護使用者與政策存放區或租用戶與政策存放區的映射。這些映射通常會維護在您

的應用程式參考的資料存放區中，例如 Amazon DynamoDB 或 Amazon Relational Database Service (Amazon RDS)。您也可以透過身分提供者 (IdP) 中的資料提供或補充這些映射。然後，租用戶、使用者和政策存放區之間的關係通常會透過 JWT 提供給使用者，其中包含授權請求所需的所有關係。

此範例顯示 JWT 如何為屬於租用戶的使用者 顯示`Alice`，`TenantA`並使用具有政策存放區 ID 的政策存放區`ps-43214321`進行授權。

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

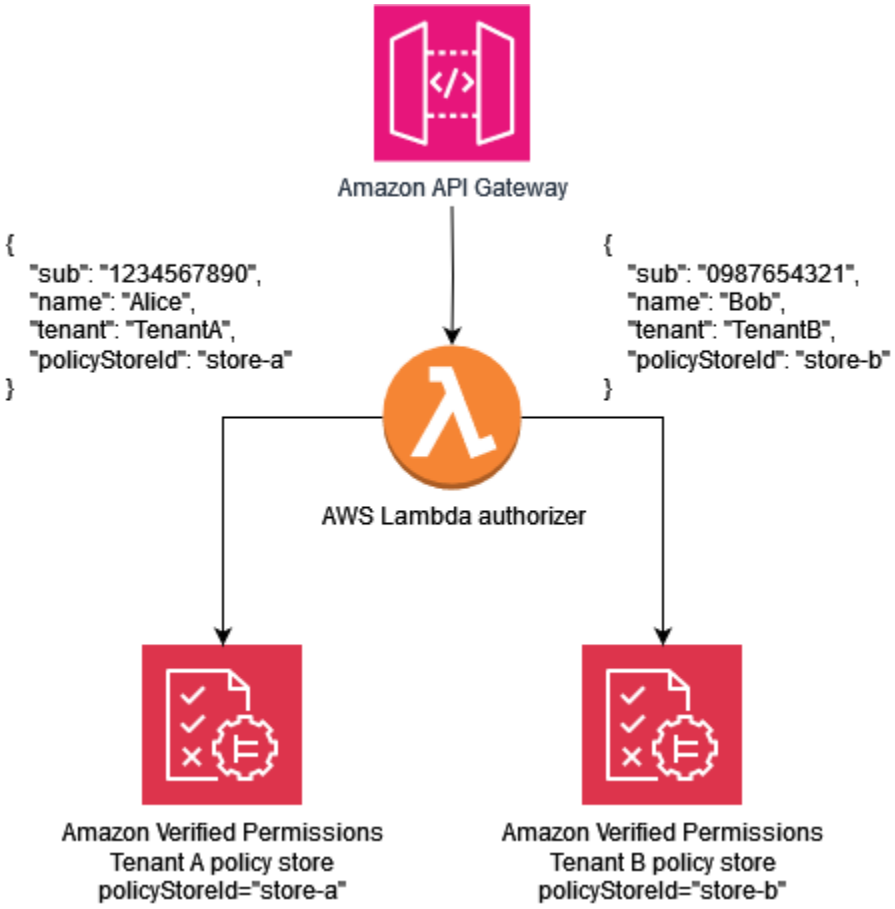
每個租戶政策存放區

Amazon Verified Permissions 中的每個租用戶政策存放區設計模型會將 SaaS 應用程式中的每個租用戶與其自己的政策存放區建立關聯。此模型類似於 SaaS 孤立隔離模型。這兩種模型都需要建立租戶特定的基礎設施，並具有類似的優點和缺點。此方法的主要優點是基礎設施強制執行租用戶隔離、支援每個租用戶的唯一授權模型、消除雜訊鄰近問題，以及減少政策更新或部署失敗的影響範圍。此方法的缺點包括更複雜的租戶加入程序、部署和操作。如果解決方案具有每個租用戶的唯一政策，則建議使用每個租用戶政策存放區。

如果您的 SaaS 應用程式需要，每個租戶政策存放區模型可以提供高度孤立的租戶隔離方法。您也可以將此模型與集區隔離搭配使用，但您的 Verified Permissions 實作不會共用更廣泛的集區隔離模型的標準優點，例如簡化的管理和操作。

在各租用戶政策存放區中，租用戶隔離是透過在使用者註冊程序期間將租用戶的政策存放區識別符映射至使用者的 SaaS 身分來實現，如前所述。此方法會將租戶的政策存放區與使用者主體緊密關聯，並提供在整個 SaaS 解決方案 中共用映射的一致方式。您可以透過將 SaaS 應用程式維護為 IdP 的一部分或在 DynamoDB 等外部資料來源中提供映射。這也可確保委託人是租用戶的一部分，並使用租用戶的政策存放區。

此範例顯示包含 `tenant` `policyStoreId`和 使用者的 JWT 如何從 API 端點傳遞到 AWS Lambda 授權方中的政策評估點，該授權方會將請求路由到正確的政策存放區。



下列範例政策說明每個租用戶的政策存放區設計範例。使用者`Alice`屬於 `TenantA`。policyStoreId `store-a`也會對應至 的租戶身分，`Alice`,並強制使用正確的政策存放區。這可確保`TenantA`使用 的政策。

> **Note**
>
> 每個租用戶的政策存放區模型會隔離租用戶的政策。授權會強制執行允許使用者對其資料執行的動作。使用這個模型的任何假設性應用程式中涉及的資源應使用其他隔離機制進行隔離，如 AWS Well-Architected Framework SaaS Lens 文件所定義。

在此政策中，`Alice`具有檢視所有 資源資料的許可。

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

若要提出授權請求並使用 Verified Permissions 政策開始評估，您需要提供對應至對應至租用戶的唯一 ID 的政策存放區 IDstore-a。

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

使用者Bob屬於租戶 B，且 policyStoreId `store-b`也會對應至 的租戶身分Bob，強制使用正確的政策存放區。這可確保使用租用戶 B 的政策。

在此政策中，Bob具有自訂所有資源資料的許可。在此範例中，customizeData可能是僅適用於租用戶 B 的動作，因此政策對於租用戶 B 是唯一的。每個租用戶的政策存放區模型本質上支援每個租用戶的自訂政策。

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

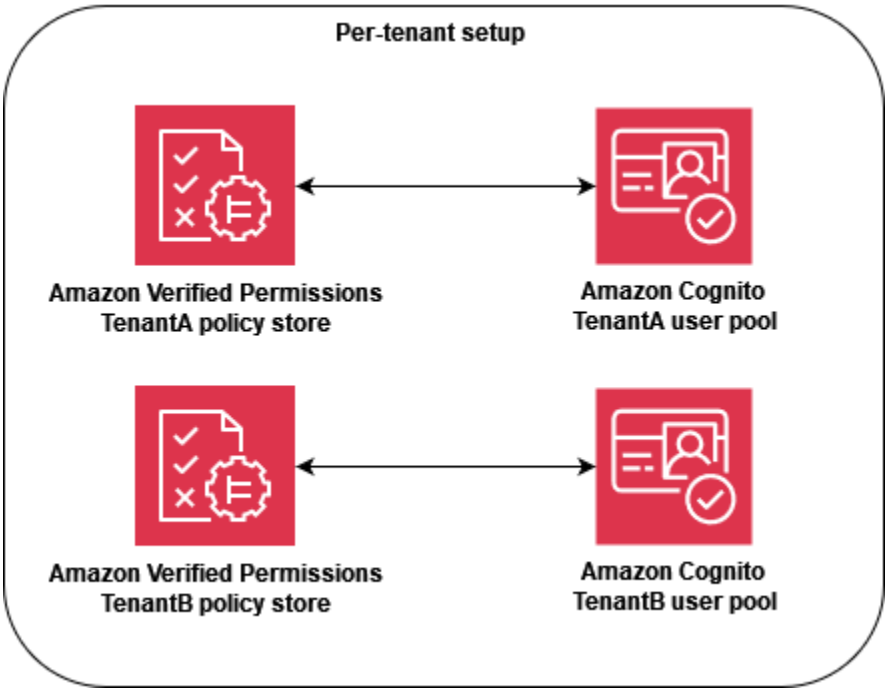
若要提出授權請求並使用 Verified Permissions 政策開始評估，您需要提供對應至對應至租用戶的唯一 ID 的政策存放區 IDstore-b。

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

使用 Verified Permissions，可以但不需要將 IdP 與政策存放區整合。此整合可讓 政策明確參考身分存放區中的委託人做為政策的委託人。如需如何將 與 Amazon Cognito 整合為 Verified Permissions IdP 的詳細資訊，請參閱 Verified Permissions 文件和 Amazon Cognito 文件。

當您將政策存放區與 IdP 整合時，每個政策存放區只能使用一個身分來源。例如，如果您選擇將 Verified Permissions 與 Amazon Cognito 整合，則必須鏡像用於租戶隔離 Verified Permissions 政策存放區和 Amazon Cognito 使用者集區的策略。政策存放區和使用者集區也必須位於相同的 中 AWS 帳戶。



在操作層級上，每個租用戶政策存放區具有稽核優勢，因為您可以為每個租用戶單獨查詢 中的記錄活動AWS CloudTrail 。不過，仍建議您將每個租用戶維度上的其他自訂指標記錄到 Amazon CloudWatch。

每個租用戶的政策存放區方法也需要密切注意兩個已驗證許可配額，以確保它們不會干擾 SaaS 解決方案的操作。這些配額是每個帳戶每個區域的政策存放區，以及每個帳戶每個區域的每秒 IsAuthorized 請求。您可以為這兩個配額請求增加。

如需如何實作每個租用戶政策存放區模型的更詳細範例，請參閱 AWS 部落格文章使用 Amazon Verified Permissions 搭配每個租用戶政策存放區的 SaaS 存取控制。

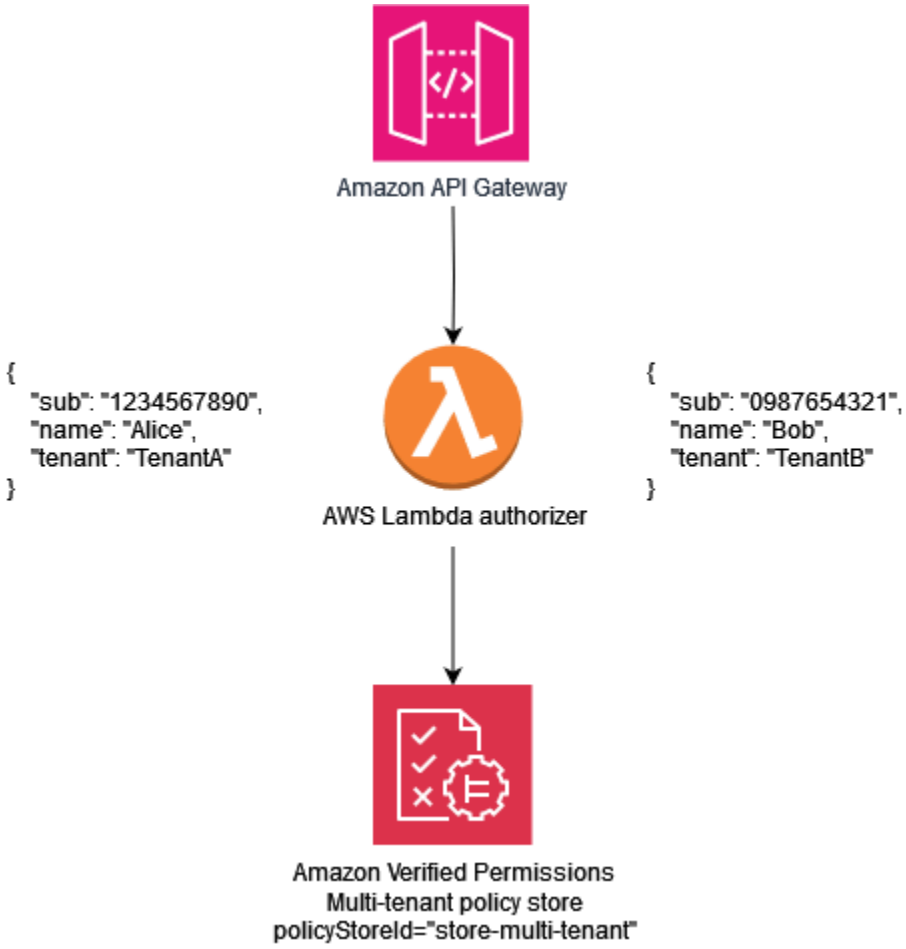
一個共用多租戶政策存放區

一個共用多租用戶政策存放區設計模型針對 SaaS 解決方案中的所有租用戶，使用 Amazon Verified Permissions 中的單一多租用戶政策存放區。此方法的主要優點是簡化管理和操作，特別是因為您不必在租戶加入期間建立額外的政策存放區。此方法的缺點包括因政策更新或部署中的任何失敗或錯誤而增加影響範圍，以及更容易受到雜訊鄰近影響。此外，如果您的解決方案需要每個租用戶的唯一政策，我們不建議使用此方法。在此情況下，請改用每個租用戶的政策存放區模型，以確保使用正確租用戶的政策。

一個共用的多租戶政策存放區方法類似於 SaaS 集區隔離模型。如果您的 SaaS 應用程式需要，它可以提供租戶隔離的集區方法。如果您的 SaaS 解決方案將孤立隔離套用至其微服務，您也可以使用此模型。當您選擇模型時，您應該評估租戶資料隔離的需求，以及 SaaS 應用程式獨立所需的 Verified Permissions 政策結構。

若要在整個 SaaS 解決方案中強制執行一致的租用戶識別符共用方式，最佳實務是在使用者註冊期間將識別符映射到使用者的 SaaS 身分，如先前所述。您可以將此映射作為 IdP 的一部分或在 DynamoDB 等外部資料來源中維護，以將此映射提供給 SaaS 應用程式。我們也建議您將共用政策存放區 ID 映射至使用者。雖然 ID 不會用作租戶隔離的一部分，但這是很好的做法，因為它有助於未來的變更。

下列範例顯示 API 端點如何為Bob屬於不同租用戶但與政策存放區 ID 共用以進行`store-multi-tenant`授權的使用者`Alice`和 傳送 JWT。由於所有租戶共用單一政策存放區，您不需要在字符或資料庫中維護政策存放區 ID。由於所有租用戶共用單一政策存放區 ID，因此您可以提供 ID 做為環境變數，供應用程式用來呼叫政策存放區。



下列範例政策說明一個共用的多租用戶政策設計範例。在此政策中，具有父系`MultiTenantApp::User`的委託人`MultiTenantApp::RoleAdmin`具有檢視所有資源資料的許可。

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

由於單一政策存放區正在使用中，Verified Permissions 政策存放區必須確保與委託人相關聯的租用屬性符合與資源相關聯的租用屬性。這可以透過在政策存放區中包含下列政策來完成，以確保資源和主體上沒有相符租用屬性的所有授權請求都會遭到拒絕。

```
forbid(
    principal,
    action,
    resource
)
```

```
unless {
    resource.Tenant == principal.Tenant
};
```

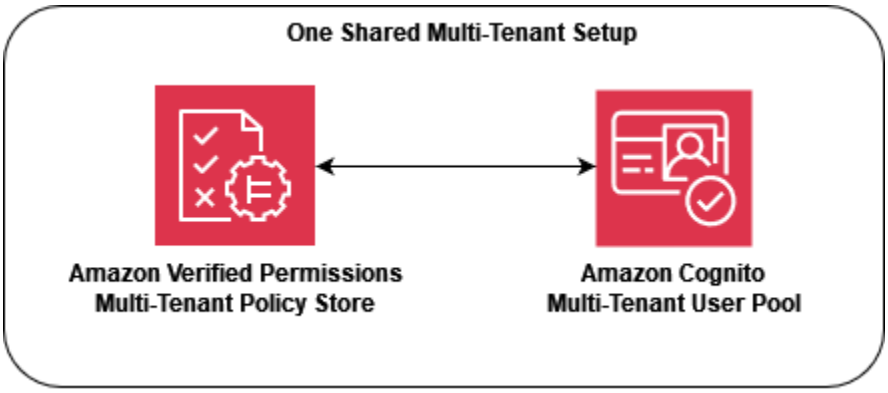
對於使用一個共用多租用戶政策存放區模型的授權請求，政策存放區 ID 是共用政策存放區的識別符。在下列請求中，UserAlice允許存取 ，因為她具有 Role的 Admin，並且與資源和主體相關聯的Tenant屬性都是 TenantA。

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

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

使用 Verified Permissions，可以但不需要將 IdP 與政策存放區整合。此整合可讓 政策明確參考身分存放區中的委託人做為政策的委託人。如需如何將 與 Amazon Cognito 整合為 Verified Permissions IdP 的詳細資訊，請參閱 Verified Permissions 文件和 Amazon Cognito 文件。

當您將政策存放區與 IdP 整合時，每個政策存放區只能使用一個身分來源。例如，如果您選擇將 Verified Permissions 與 Amazon Cognito 整合，則必須鏡像用於租戶隔離 Verified Permissions 政策存放區和 Amazon Cognito 使用者集區的策略。政策存放區和使用者集區也必須位於相同的 中 AWS 帳戶。



從操作和稽核的觀點來看，一個共用的多租用戶政策存放區模型具有缺點，因為 中記錄的活動 AWS CloudTrail需要更多涉及的查詢來篩選租用戶上的個別活動，因為每個記錄的 CloudTrail 呼叫都使用相同的政策存放區。在此案例中，將每個租用戶維度上的其他自訂指標記錄到 Amazon CloudWatch，以確保適當的可觀測性和稽核功能。

一種共用的多租用戶政策存放區方法也需要密切注意已驗證許可配額，以確保它們不會干擾 SaaS 解決方案的操作。特別是，我們建議您監控每個區域每個帳戶配額每秒的 IsAuthorized 請求，以確保不超過其限制。您可以請求提高此配額。

分層部署模型

透過建立分層部署模型，您可以隔離高優先順序的「企業層」租用戶與可能更高數量的「標準層」客戶。在此模型中，您可以針對每個層分別推出政策存放區中部署到政策的任何變更，這會隔離每個客戶層與其層之外所做的變更。在分層部署模型中，政策存放區通常會在每個層的初始基礎設施佈建中建立，而不是在租戶加入時部署。

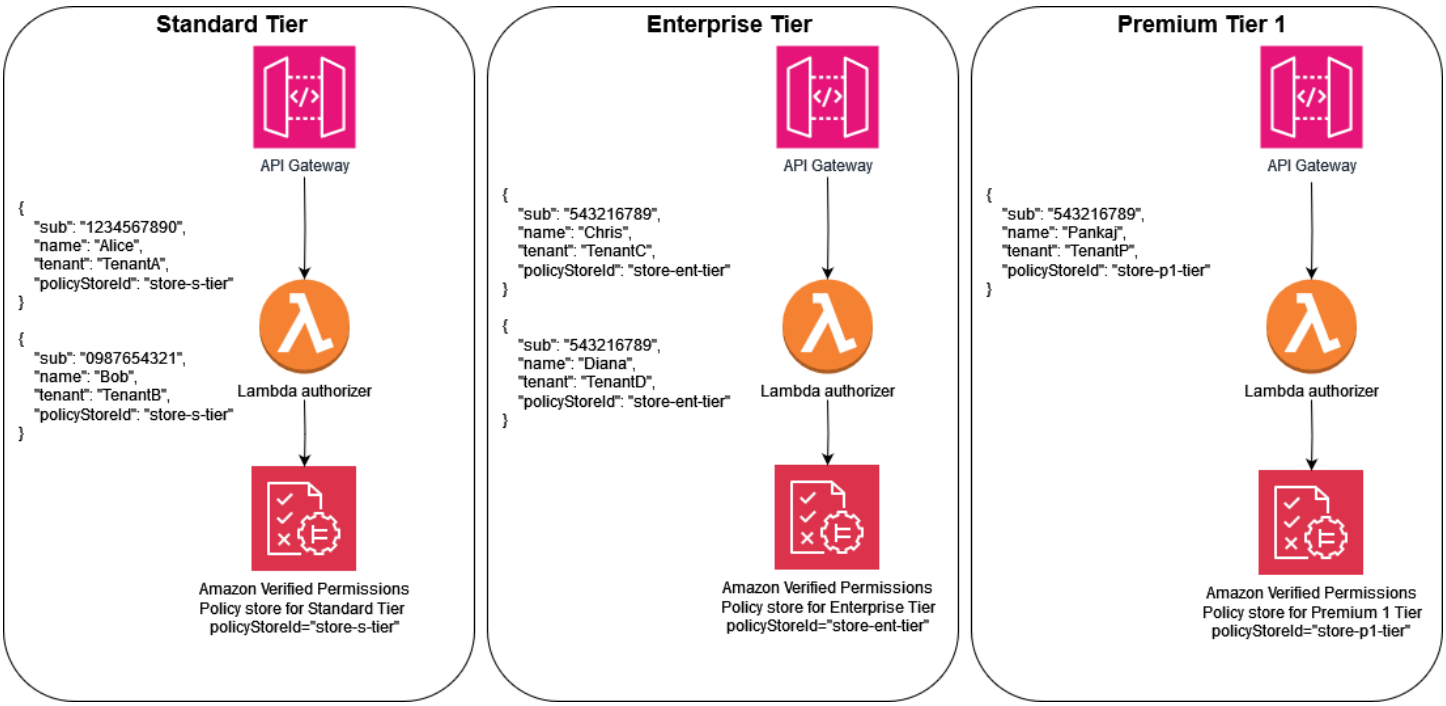
如果您的解決方案主要使用集區隔離模型，您可能需要額外的隔離或自訂。例如，您可以建立一個「高級方案」，其中每個租用戶都會取得自己的租用戶方案基礎設施，透過部署只有一個租用戶的集區執行個體來建立孤立模型。這可以採用完全分開的「Premium Tier 租用戶 A」和「Premium Tier 租用戶 B」基礎設施的形式，包括政策存放區。此方法可為最高層級的客戶產生孤立隔離模型。

在分層部署模型中，每個政策存放區都應該遵循相同的隔離模型，但會分別部署。由於使用多個政策存放區，因此您需要在整個 SaaS 解決方案中強制執行共用與租用戶相關聯之政策存放區識別符的一致方式。如同每個租用戶的政策存放區模型，在使用者註冊期間，將租用戶識別符映射到使用者的 SaaS 身分是很好的做法。

下圖顯示三個層：`Standard Tier`、`Enterprise Tier`和 `Premium Tier 1`。每個層都會分別部署在自己的基礎設施中，並在層中使用一個共用政策存放區。標準和企業方案包含多個租用戶。`TenantA`和 `TenantB` 位於 `Standard Tier`，`TenantC` 和 `TenantD` 位於企業方案。

`Premium Tier 1` 僅包含 `TenantP`，因此您可以像解決方案具有完全孤立的隔離模型一樣為高級租戶提供服務，並提供自訂政策等功能。加入新的高級方案客戶將導致建立`Premium Tier 2`基礎設施。

> **Note**
>
> 高級方案中的應用程式、部署和租戶加入與標準和企業方案相同。唯一的差別在於，進階方案加入工作流程從佈建新的方案基礎設施開始。


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

OPA 多租戶設計考量事項

開放政策代理程式 (OPA) 是一種靈活的服務，可套用至許多需要應用程式才能做出政策和授權決策的使用案例。搭配多租用戶 SaaS 應用程式使用 OPA 需要考量唯一條件，以確保租用戶隔離等關鍵 SaaS 最佳實務仍是 OPA 實作的一部分。這些條件包括 OPA 部署模式、租戶隔離和 OPA 文件模型，以及租戶加入。這些都會影響 OPA 的最佳設計，因為它與多租戶應用程式有關。

雖然本節中的討論著重於 OPA，但一般概念是以隔離思維及其提供的指引為基礎。SaaS 應用程式必須一律將租戶隔離視為其設計的一部分，而此一般隔離原則延伸到在 SaaS 應用程式中包含 OPA。如果正確使用 OPA， 可能是 SaaS 應用程式中如何強制執行隔離的關鍵部分。本節也參考核心 SaaS 隔離模型，例如孤立 SaaS 模型和集區 SaaS 模型。如需詳細資訊，請參閱 AWS Well-Architected Framework SaaS Lens 中的核心隔離概念。

設計考量事項：

- 比較集中式和分散式部署模式
- 使用 OPA 文件模型的租用戶隔離
- 租戶加入

比較集中式和分散式部署模式

您可以在集中式或分散式部署模式中部署 OPA，而多租用戶應用程式的理想方法取決於使用案例。如需這些模式的範例，請參閱本指南稍早的在 API 上使用集中式 PDP 搭配 PEPs， APIs以及在 APIs 上使用分散式 PDP 和 PEPs 一節。由於 OPA 可以部署為作業系統或容器中的協助程式，因此可以透過多種方式實作以支援多租戶應用程式。

在集中式部署模式中，OPA 會部署為容器或協助程式，其 RESTful API 可供應用程式中的其他 服務使用。當服務需要來自 OPA 的決策時，會呼叫中央 OPA RESTful API 來產生此決策。這種方法易於部署和維護，因為只有單一部署的 OPA。此方法的缺點是，它不提供任何機制來維護租用戶資料的分離。由於 OPA 只有單一部署，因此 OPA 決策中使用的所有租用戶資料，包括 OPA 參考的任何外部資料，都必須可供 OPA 使用。您可以使用此方法維持租戶資料隔離，但必須由 OPA 政策和文件結構或外部資料的存取權強制執行。集中式部署模式也需要更高的延遲，因為每個授權決策都必須對其他服務發出 RESTful API 呼叫。

在分散式部署模式中，OPA 會與多租戶應用程式的服務一起部署為容器或協助程式。它可以部署為附屬容器，也可以部署為在作業系統本機執行的協助程式。若要從 OPA 擷取決策，服務會對本機 OPA 部署進行 RESTful API 呼叫。（因為 OPA 可以部署為 Go 套件，所以您可以原生使用 Go 來擷取決策，而不是使用 RESTful API 呼叫。）與集中式部署模式不同，分散式模式需要更強大的努力來部署、

維護和更新，因為它存在於應用程式的多個區域。分散式部署模式的好處之一是能夠維持租戶資料的隔離，尤其是使用孤立 SaaS 模型的應用程式。租用戶特定的資料可以在該租用戶特有的 OPA 部署中隔離，因為分散式模型中的 OPA 會與租用戶一起部署。此外，分散式部署模式的延遲遠低於集中式部署模式，因為每個授權決策都可以在本機進行。

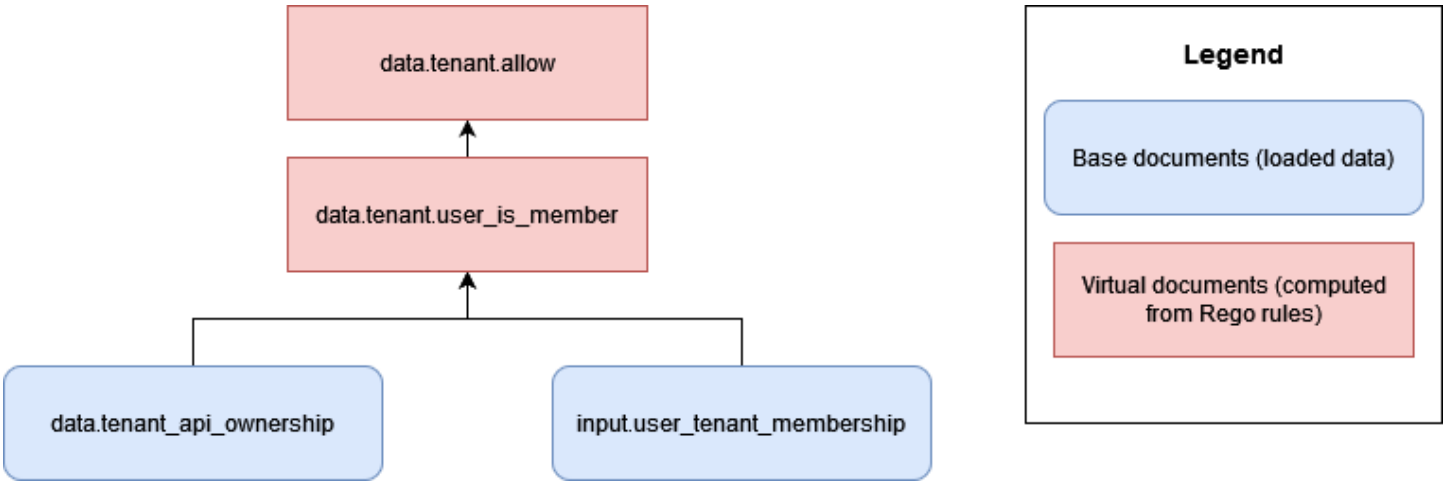
當您在多租用戶應用程式中選擇 OPA 部署模式時，請務必評估應用程式最關鍵的條件。如果您的多租用戶應用程式對延遲很敏感，分散式部署模式可以提供更好的效能，同時犧牲更複雜的部署和維護。雖然您可以透過 DevOps 和自動化來管理其中一些複雜性，但與集中式部署模式相比，它仍需要額外的努力。

如果您的多租用戶應用程式使用孤立 SaaS 模型，您可以使用分散式 OPA 部署模式來模擬孤立方法來租用戶資料隔離。這是因為當 OPA 與每個租用戶特定的應用程式服務一起執行時，您可以自訂每個 OPA 部署，只包含與該租用戶相關聯的資料。無法在集中式 OPA 部署模式中將 OPA 資料孤立。如果您使用集中式部署模式或分散式模式搭配集區 SaaS 模型，則必須在 OPA 文件模型中維護租戶資料隔離。

使用 OPA 文件模型的租用戶隔離

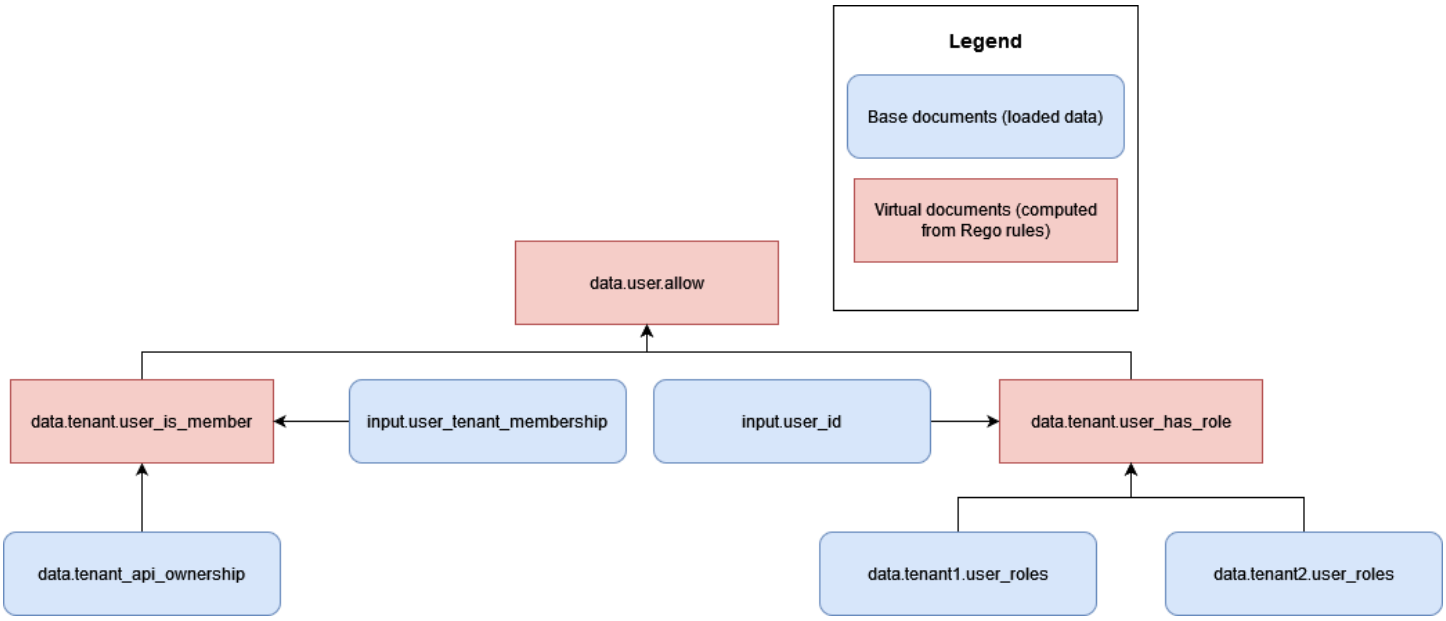
OPA 使用文件做出決策。這些文件可以包含租戶特定資料，因此您必須考慮如何維持租戶資料隔離。OPA 文件由基礎文件和虛擬文件組成。基礎文件包含來自外部世界的資料。這包括直接提供給 OPA 的資料、OPA 請求的資料，以及可能做為輸入傳遞給 OPA 的資料。虛擬文件是根據政策計算，並包含 OPA 政策和規則。如需詳細資訊，請參閱 OPA 文件。

若要在 OPA 中為多租戶應用程式設計文件模型，您必須先考慮在 OPA 中做出決策所需的基礎文件類型。如果這些基礎文件包含租用戶特定資料，您必須採取措施以確保這些資料不會意外暴露到跨租用戶存取。幸好在許多情況下，租戶特定資料不需要在 OPA 中做出決策。下列範例顯示假設的 OPA 文件模型，允許根據哪個租用戶擁有 API 來存取 API，以及使用者是否為租用戶的成員，如輸入文件中所示。



在此方法中，OPA 無法存取任何租用戶特定資料，除了有關哪些租用戶擁有 API 的資訊。在這種情況下，無需擔心 OPA 促進跨租用戶存取，因為 OPA 用於做出存取決策的唯一資訊是使用者與租用戶的關聯，以及租用戶與 APIs關聯。您可以將此類型的 OPA 文件模型套用至孤立的 SaaS 模型，因為每個租用戶都有獨立資源的所有權。

不過，在許多 RBAC 授權方法中，有可能跨租用戶公開資訊。在下列範例中，假設性 OPA 文件模型允許根據使用者是否為租用戶的成員，以及使用者是否具有存取 API 的正確角色來存取 API。



此模型帶來跨租用戶存取的風險，因為多個租用戶在 中的角色`data.tenant1.user_roles`和許可`data.tenant2.user_roles`現在必須可供 OPA 存取，才能做出授權決策。為了維持租戶隔離和角色映射的隱私權，此資料不應位於 OPA 中。RBAC 資料應位於外部資料來源中，例如資料庫。此外，OPA 不應用於將預先定義的角色映射至特定許可，因為這會使租戶難以定義自己的角色和許可。它也會讓您的授權邏輯更嚴格，且需要持續更新。如需如何將 RBAC 資料安全地納入 OPA 決策程序的指引，請參閱本指南稍後的租戶隔離和資料隱私權建議一節。

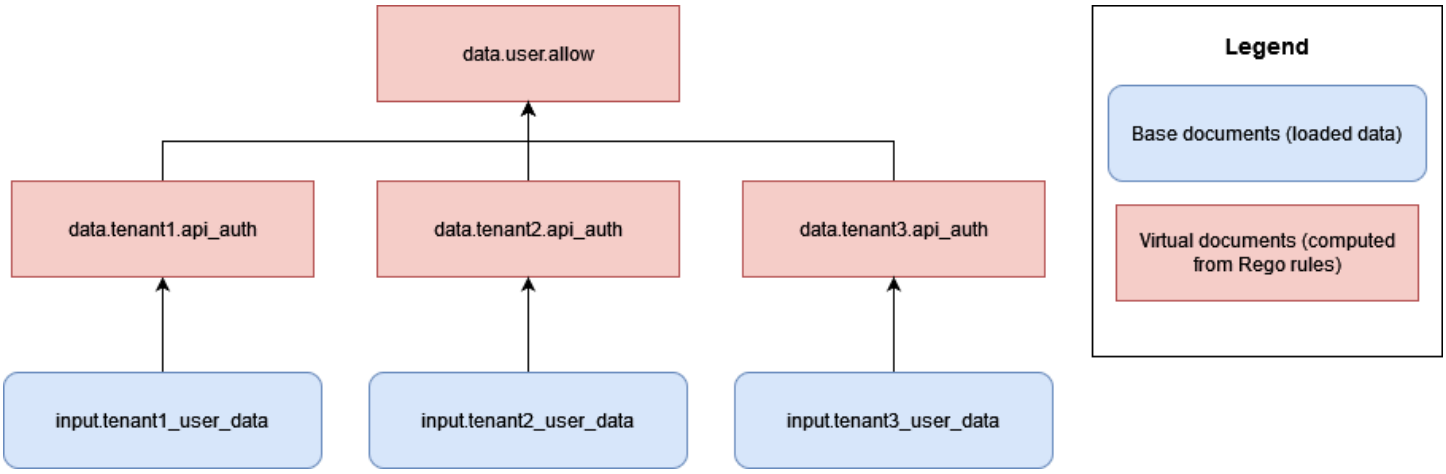
您可以透過不將任何租用戶特定資料儲存為非同步基礎文件，輕鬆地在 OPA 中維護租用戶隔離。非同步基礎文件是存放在記憶體中的資料，可在 OPA 中定期更新。其他基礎文件，例如 OPA 輸入，會同步傳遞，且只能在決策時間使用。例如，在查詢的 OPA 輸入中提供租戶特定資料不會構成違反租戶隔離，因為該資料僅在決策過程中同步可用。

租戶加入

OPA 文件的結構必須允許直接的租戶加入，而不會引入繁瑣的需求。您可以使用套件在 OPA 文件模型階層中組織虛擬文件，而且這些套件可以包含許多規則。當您為多租戶應用程式規劃 OPA 文件模型

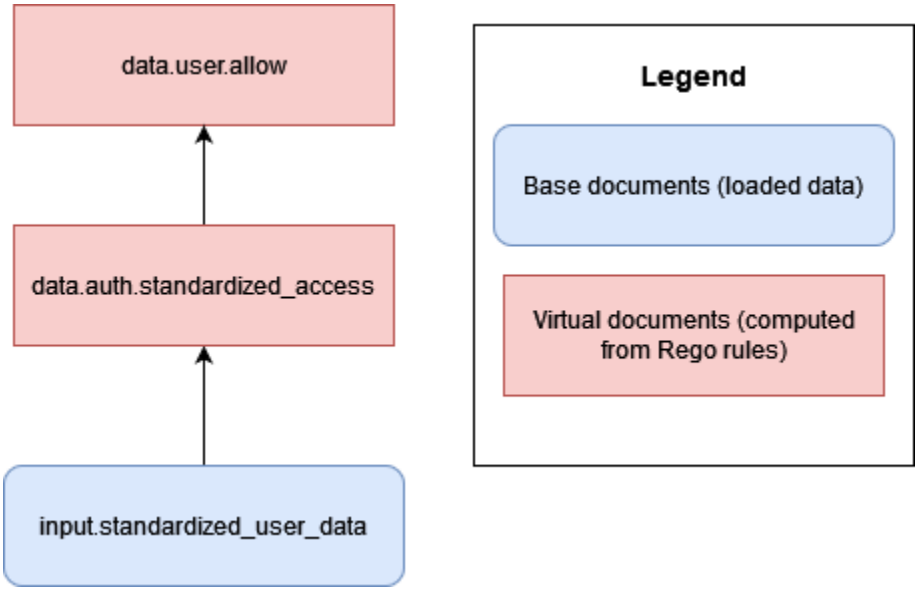
時，請先判斷 OPA 需要哪些資料才能做出決策。您可以提供資料做為輸入、預先載入 OPA，或在決策時間或定期從外部資料來源提供資料。如需搭配 OPA 使用外部資料的詳細資訊，請參閱本指南稍後在 OPA 中擷取 PDP 的外部資料一節。

在您決定在 OPA 中做出決策所需的資料後，請考慮如何實作組織為套件的 OPA 規則，以對該資料做出決策。例如，在孤立的 SaaS 模型中，每個租用戶對於授權決策的制定方式可能有獨特的要求，您可以實作租用戶特定的 OPA 規則套件。



此方法的缺點是您必須為每個租用戶新增一組新的 OPA 規則，針對您新增至 SaaS 應用程式的每個租用戶。這很繁瑣且難以擴展，但根據您的租戶需求，可能無法避免。

或者，在集區 SaaS 模型中，如果所有租用戶都根據相同的規則做出授權決策並使用相同的資料結構，您可以使用具有一般適用規則的標準 OPA 套件，以更輕鬆地加入租用戶並擴展您的 OPA 實作。



我們建議您盡可能使用一般化 OPA 規則和套件（或虛擬文件），根據每個租戶提供的標準化資料做出決策。這種方法可讓 OPA 易於擴展，因為您只會變更提供給每個租用戶的 OPA 資料，而不是 OPA 如

何透過其規則提供其決策。只有在個別租用戶需要唯一決策，或必須提供 OPA 與其他租用戶不同的資料時，才需要引進每個租用戶的rules-per-tenant模型。

DevOps、監控、記錄和擷取 PDP 的資料

在此提議的授權範例中，政策會集中在授權服務中。此集中化是刻意的，因為本指南中討論之設計模型的其中一個目標是實現政策解耦，或從應用程式中的其他元件移除授權邏輯。Amazon Verified Permissions 和 Open Policy Agent (OPA) 都提供在需要變更授權邏輯時更新政策的機制。

如果是 Verified Permissions，軟體 AWS 開發套件會提供以程式設計方式更新政策的機制（請參閱 Amazon Verified Permissions API 參考指南)。您可以使用 SDK 隨需推送新政策。此外，由於 Verified Permissions 是受管服務，因此您不需要管理、設定或維護控制平面或代理程式來執行更新。不過，我們建議您使用持續整合和持續部署 (CI/CD) 管道，來使用 AWS SDK 管理 Verified Permissions 政策存放區和政策更新的部署。

Verified Permissions 可讓您輕鬆存取可觀測性功能。它可以設定為記錄 Amazon CloudWatch 日誌群組 AWS CloudTrail、Amazon Simple Storage Service (Amazon S3) 儲存貯體或 Amazon Data Firehose 交付串流的所有存取嘗試，以便快速回應安全事件和稽核請求。此外，您可以透過 監控 Verified Permissions 服務的運作狀態 AWS Health 儀板表。由於 Verified Permissions 是受管服務，其運作狀態由 維護 AWS，您可以使用其他 AWS 受管服務來設定可觀測性功能。

在 OPA 的情況下，REST APIs提供以程式設計方式更新政策的方法。您可以設定 APIs 從已建立的位置提取新版本的政策套件，或隨需推送政策。此外，OPA 提供基本探索服務，其中新代理程式可以動態設定，並由分佈探索套件的控制平面集中管理。(OPA 的控制平面必須由 OPA 運算子設定。) 我們建議您建立強大的 CI/CD 管道來版本控制、驗證和更新政策，無論政策引擎是 Verified Permissions、OPA 或其他解決方案。

對於 OPA，控制平面也提供監控和稽核的選項。您可以將包含 OPA 授權決策的日誌匯出至遠端 HTTP 伺服器以進行日誌彙總。這些決策日誌對於稽核而言非常重要。

如果您正在考慮採用授權模型，其中存取控制決策會從您的應用程式分離，請確定您的授權服務具有有效的監控、記錄和 CI/CD 管理功能，以加入新的 PDPs或更新政策。

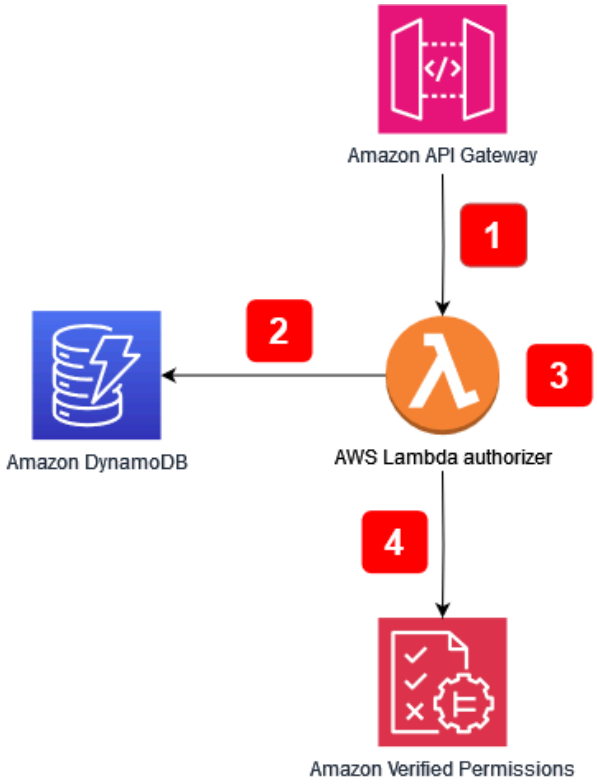
主題

- 在 Amazon Verified Permissions 中擷取 PDP 的外部資料
- 在 OPA 中擷取 PDP 的外部資料
- 租戶隔離和資料隱私權的建議

在 Amazon Verified Permissions 中擷取 PDP 的外部資料

Amazon Verified Permissions 不支援擷取 PDP 的外部資料，但可以儲存使用者提供的資料作為其結構描述的一部分。如同 OPA，如果授權決策的所有資料都可以作為授權請求的一部分提供，或作為作為請求的一部分傳遞的 JSON Web Token (JWT) 的一部分提供，則不需要額外的組態。不過，您可以在呼叫 Verified Permissions 的應用程式授權方服務中，透過授權請求將來自外部來源的其他資料提供給 Verified Permissions。例如，應用程式的授權方服務可以查詢外部來源，例如 DynamoDB 或 Amazon RDS 以取得資料，然後這些服務可以包含外部提供的資料作為授權請求的一部分。

下圖顯示如何擷取其他資料並將其納入 Verified Permissions 授權請求的範例。可能需要使用此方法擷取資料，例如 RBAC 角色映射、擷取與資源或主體相關的其他屬性，或在資料位於應用程式不同部分且無法透過身分提供者 (IdP) 權杖提供的情況下。



應用程式流程：

1. 應用程式會收到對 Amazon API Gateway 的 API 呼叫，並將呼叫轉送給 AWS Lambda 授權方。
2. Lambda 授權方會呼叫 Amazon DynamoDB，以擷取提出請求之委託人的其他資料。
3. Lambda 授權方會將其他資料納入對 Verified Permissions 提出的授權請求中。
4. Lambda 授權方向 Verified Permissions 提出授權請求，並收到授權決定。

圖表包含稱為 Lambda 授權方的 Amazon API Gateway 功能。雖然此功能可能無法用於由其他服務或應用程式提供的 APIs，但您可以使用 授權方來複寫一般模型，以擷取其他資料，以在多個使用案例中併入 Verified Permissions 授權請求。

在 OPA 中擷取 PDP 的外部資料

對於 OPA，如果授權決策所需的所有資料可以作為輸入提供，或作為作為作為查詢元件傳遞的 JSON Web Token (JWT) 的一部分提供，則不需要額外的組態。（將 JWTs和 SaaS 內容資料作為查詢輸入的一部分傳遞至 OPA 相當簡單。) OPA 可以在所謂的過載輸入方法中接受任意 JSON 輸入。如果 PDP 需要的資料超過可做為輸入或 JWT 包含的資料，OPA 會提供數個選項來擷取此資料。這些包括綁定、推送資料（複寫）和動態資料擷取。

OPA 綁定

OPA 綁定功能支援下列外部資料擷取程序：

1. 政策強制執行點 (PEP) 會請求授權決策。
2. OPA 會下載新的政策套件，包括外部資料。
3. 綁定服務會複寫來自資料來源（資料來源）的資料。

當您使用綁定功能時，OPA 會定期從集中式套件服務下載政策和資料套件。(OPA 不提供套件服務的實作和設定。) 從套件服務提取的所有政策和外部資料都會儲存在記憶體中。如果外部資料大小太大而無法存放在記憶體中，或資料變更太頻繁，此選項將無法運作。

如需綁定功能的詳細資訊，請參閱 OPA 文件。

OPA 複寫（推送資料）

OPA 複寫方法支援下列外部資料擷取程序：

1. PEP 請求授權決策。
2. 資料複寫器會將資料推送至 OPA。
3. 資料複寫器會從資料來源 (s) 複寫資料。

在此綁定方法的替代方案中，資料會推送到 ，而不是由 OPA 定期提取。(OPA 不提供複寫器的實作和設定。) 推送方法具有與綁定方法相同的資料大小限制，因為 OPA 會將所有資料存放在記憶體中。推

送選項的主要優點是您可以使用差異更新 OPA 中的資料，而不是每次都取代所有外部資料。這使得推送選項更適合經常變更的資料集。

如需複寫選項的詳細資訊，請參閱 OPA 文件。

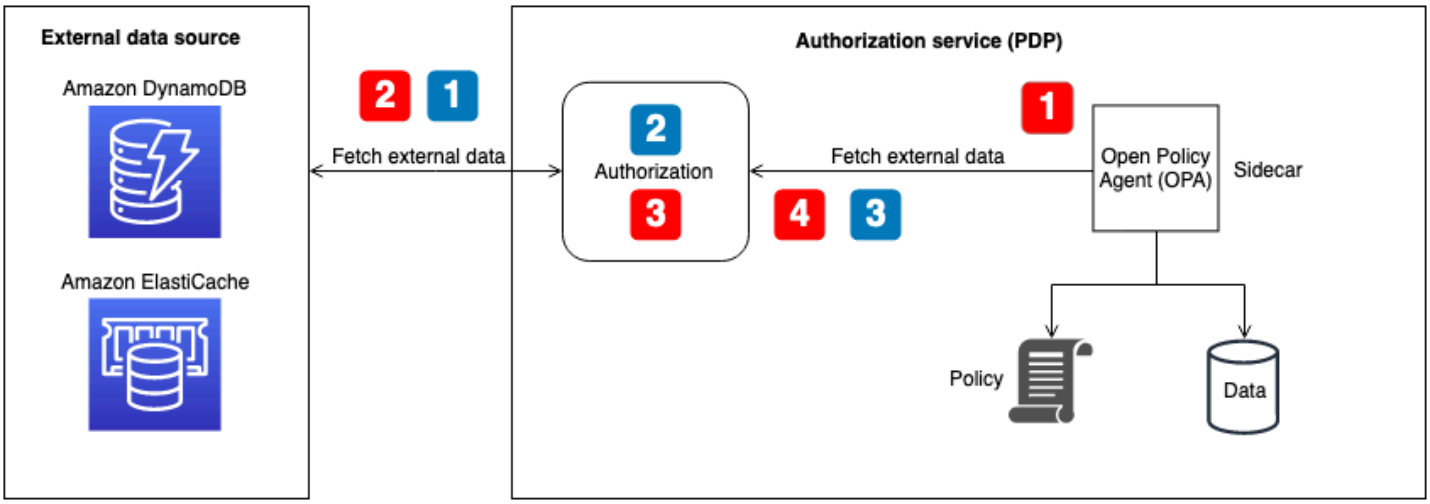
OPA 動態資料擷取

如果要擷取的外部資料太大而無法快取到 OPA 記憶體中，則在評估授權決策期間，可以從外部來源動態提取資料。當您使用此方法時，資料一律是最新的。這種方法有兩個缺點：網路延遲和可存取性。目前，OPA 只能在執行時間透過 HTTP 請求擷取資料。如果傳送至外部資料來源的呼叫無法以 HTTP 回應傳回資料，則需要自訂 API 或其他機制，才能將此資料提供給 OPA。由於 OPA 只能透過 HTTP 請求擷取資料，而且擷取資料的速度至關重要，我們建議您盡可能使用 Amazon DynamoDB AWS 服務等 來保留外部資料。

如需提取方法的詳細資訊，請參閱 OPA 文件。

使用授權服務搭配 OPA 實作

當您使用綁定、複寫或動態提取方法來擷取外部資料時，我們建議您的授權服務促進此互動。這是因為授權服務可以擷取外部資料並將其轉換為 JSON，以便 OPA 做出授權決策。下圖顯示授權服務如何搭配這三種外部資料擷取方法運作。



擷取 OPA 流程的外部資料 – 在決策時間的套件或動態資料擷取 （圖表中以紅色編號的標註表示）：

1. OPA 會呼叫授權服務的本機 API 端點，該端點設定為套件端點，或在授權決策期間用於動態資料擷取的端點。
2. 授權服務會查詢或呼叫外部資料來源來擷取外部資料。（對於套件端點，此資料也應包含 OPA 政策和規則。 套件更新會取代 OPA 快取中的所有項目，包括資料和政策。）

3. 授權服務會對傳回的資料執行任何必要的轉換，以將其轉換為預期的 JSON 輸入。
4. 資料會傳回 OPA。它會快取在用於套件組態的記憶體中，並立即用於動態授權決策。

擷取 OPA 流程的外部資料 – 複寫器（圖表中以藍色編號的標註表示）：

1. 複寫器（授權服務的一部分）會呼叫外部資料來源，並擷取要在 OPA 中更新的任何資料。這可以包含政策、規則和外部資料。此呼叫可以按照設定的節奏進行，也可以回應外部來源中的資料更新。
2. 授權服務會對傳回的資料執行任何必要的轉換，以將其轉換為預期的 JSON 輸入。
3. 授權服務會呼叫 OPA 並快取記憶體中的資料。授權服務可以選擇性地更新資料、政策和規則。

租戶隔離和資料隱私權的建議

上一節提供數種搭配 OPA 和 Amazon Verified Permissions 使用外部資料的方法，以協助做出授權決策。如果可能，我們建議您使用過載輸入方法來將 SaaS 內容資料傳遞至 OPA，以做出授權決策，而不是將資料存放在 OPA 記憶體中。此使用案例不適用於 AWS Cloud Map，因為它不支援將外部資料儲存在服務中。

在角色型存取控制 (RBAC) 或 RBAC 和屬性型存取控制 (ABAC) 混合模型中，僅由授權請求或查詢提供的資料可能不足，因為必須參考角色和許可才能做出授權決策。為了維持租戶隔離和角色映射的隱私權，此資料不應位於 OPA 中。RBAC 資料應位於資料庫等外部資料來源中，或應做為 JWT 中宣告的一部分從 IdP 傳遞。在 Verified Permissions 中，RBAC 資料可以維護為每個租用戶政策存放區模型中政策和結構描述的一部分，因為每個租用戶都有自己的邏輯分隔政策存放區。不過，在一個共用的多租用戶政策存放區模型中，角色映射資料不應位於 Verified Permissions 中，以維持租用戶隔離。

此外，OPA 和 Verified Permissions 不應用於將預先定義的角色映射至特定許可，因為這會使租戶難以定義自己的角色和許可。它也會讓您的授權邏輯更嚴格，且需要持續更新。本指南的例外是 Verified Permissions 中的每個租用戶政策存放區模型，因為此模型允許每個租用戶擁有自己的政策，可以根據每個租用戶獨立評估。

Amazon Verified Permissions

Verified Permissions 可以存放潛在私有 RBAC 資料的唯一位置是在結構描述中。這是每個租用戶政策存放區模型中可接受的，因為每個租用戶都有自己的邏輯分隔政策存放區。不過，這可能會影響一個共用多租用戶政策存放區模型中的租用戶隔離。在需要這些資料才能做出授權決策的情況下，應該從外部來源擷取資料，例如 DynamoDB 或 Amazon RDS，並納入 Verified Permissions 授權請求。

OPA

使用 OPA 安全方法來維護 RBAC 資料的隱私權和租用戶隔離，包括使用動態資料擷取或複寫來取得外部資料。這是因為您可以使用上圖中說明的授權服務，僅提供租戶特定或使用者特定外部資料，以進行授權決策。例如，您可以使用複寫器，在使用者登入時將 RBAC 資料或許可矩陣提供給 OPA 快取，並根據輸入資料 中提供的使用者來參考資料 。您可以將類似的方法與動態提取的資料搭配使用，以僅擷取進行授權決策的相關資料。此外，在動態資料擷取方法中，此資料不必快取在 OPA 中。綁定方法與維持租用戶隔離的動態擷取方法不那麼有效，因為它會更新 OPA 快取中的所有內容，而且無法處理精確的更新。綁定模型仍然是更新 OPA 政策和非RBAC 資料的好方法。

最佳實務

本節列出本指南的一些高階要點。如需各點的詳細討論，請遵循對應區段的連結。

選取適用於您應用程式的存取控制模型

本指南討論數個存取控制模型。根據您的應用程式和業務需求，您應該選取適合您的模型。考慮如何使用這些模型來滿足您的存取控制需求，以及存取控制需求可能如何演變，這需要變更您選擇的方法。

實作 PDP

政策決策點 (PDP) 的特性可以是政策或規則引擎。此元件負責套用政策或規則，並傳回是否允許特定存取的決定。PDP 允許將應用程式程式碼中的授權邏輯卸載至個別系統。這可以簡化應用程式程式碼。它還提供easy-to-use等冪界面，用於對 APIs、微服務、後端前端 (BFF) 層或任何其他應用程式元件進行授權決策。PDP 可用來在整個應用程式中一致地強制執行租用需求。

為應用程式中的每個 API 實作 PEPs

政策強制執行點 (PEP) 的實作需要判斷應用程式中應在何處執行存取控制。首先，找到應用程式中您可以納入 PEPs點。在決定在何處新增 PEPs 時，請考慮此原則：

如果應用程式公開 API，則應對該 API 進行授權和存取控制。

考慮使用 Amazon Verified Permissions 或 OPA 作為 PDP 的政策引擎

Amazon Verified Permissions 優於自訂政策引擎。它為您建置的應用程式提供可擴展、精細的許可管理和授權服務。它支援以高階宣告性開放原始碼語言 Cedar 撰寫政策。因此，相較於實作您自己的解決方案，使用 Verified Permissions 實作政策引擎所需的開發工作較少。此外，Verified Permissions 是完全受管的，因此您不需要管理基礎基礎設施。

開放政策代理程式 (OPA) 優於自訂政策引擎。OPA 及其使用 Rego 進行的政策評估提供彈性的預先建置政策引擎，支援以高階宣告性語言撰寫政策。這使得實作政策引擎所需的工作量遠低於建置您自己的解決方案。此外，OPA 很快就成為支援良好的授權標準。

實作適用於 DevOps、監控和記錄的 OPA 控制平面

由於 OPA 不提供透過來源控制更新和追蹤授權邏輯變更的方法，因此我們建議您實作控制平面來執行這些函數。這將允許更輕鬆地將更新分發給 OPA 代理程式，特別是在 OPA 在分散式系統中操作時，這將減少使用 OPA 的管理負擔。此外，控制平面可用來收集日誌以進行彙總，並監控 OPA 代理程式的狀態。

在 Verified Permissions 中設定記錄和可觀測性功能

Verified Permissions 可讓您輕鬆存取可觀測性功能。您可以設定服務來記錄 Amazon CloudWatch 日誌群組 AWS CloudTrail、S3 儲存貯體或 Amazon Data Firehose 交付串流的所有存取嘗試，以便快速回應安全事件和稽核請求。此外，您可以透過 監控服務的運作狀態 AWS Health 儀板表。由於 Verified Permissions 是受管服務，其運作狀態由 維護 AWS，您可以使用其他 AWS 受管服務來設定其可觀測性功能。

使用 CI/CD 管道在 Verified Permissions 中佈建和更新政策存放區和政策

Verified Permissions 是一種受管服務，因此您不需要管理、設定或維護控制平面或代理程式來執行更新。不過，我們仍建議您使用持續整合和持續部署 (CI/CD) 管道，透過 AWS SDK 來管理 Verified Permissions 政策存放區和政策更新的部署。當您變更 Verified Permissions 資源時，這項工作可以消除手動工作，並降低運算子錯誤的可能性。

判斷授權決策是否需要外部資料，然後選取模型以容納它

如果 PDP 可以僅根據 JSON Web Token (JWT) 中包含的資料進行授權決策，則通常不需要匯入外部資料來協助做出這些決策。如果您使用 Verified Permissions 或 OPA 做為 PDP，它也可以接受作為請求一部分傳遞的額外輸入，即使此資料不包含在 JWT 中。對於 Verified Permissions，您可以使用其他資料的內容參數。對於 OPA，您可以使用 JSON 資料作為過載輸入。如果您使用 JWT，內容或過載輸入方法通常比在另一個來源中維護外部資料更容易。如果需要更複雜的外部資料來做出授權決策，OPA 會提供數種模型來擷取外部資料，而 Verified Permissions 可以透過使用授權服務參考外部來源來補充其授權請求中的資料。

常見問答集

本節提供有關在多租用戶 SaaS 應用程式中實作 API 存取控制和授權的常見問題解答。

問：授權和身分驗證有何不同？

A.Authentication 是驗證使用者身分的程序。授權會授予使用者存取特定資源的許可。

問：SaaS 應用程式中的授權與租戶隔離有何不同？

答：租用戶隔離是指 SaaS 系統中使用的明確機制，以確保即使在共用基礎設施上操作時，每個租用戶的資源都已隔離。多租戶授權是指傳入動作的授權，並防止在錯誤的租戶上實作這些動作。假設使用者可以進行身分驗證和授權，但仍然可以存取另一個租用戶的資源。身分驗證和授權不一定會封鎖此存取，但租戶需要隔離才能達成此目標。如需這兩個概念的詳細資訊，請參閱 AWS SaaS 架構基礎白皮書中的租戶隔離討論。

問：為什麼我需要考慮 SaaS 應用程式的租戶隔離？

A.SaaS 應用程式有多個租用戶。租用戶可以是客戶組織或任何使用該 SaaS 應用程式的外部實體。根據應用程式的設計方式，這表示租戶可能會存取共用 APIs、資料庫或其他資源。請務必維持租戶隔離，也就是可嚴格控制對資源的存取，並封鎖任何存取其他租戶資源的嘗試，以防止使用者從一個租戶存取另一個租戶的私有資訊。SaaS 應用程式通常旨在確保在整個應用程式中維持租用戶隔離，並且租用戶只能存取自己的資源。

問：為什麼我需要存取控制模型？

A.存取控制模型用於建立一致的方法來判斷如何授予應用程式中資源的存取權。這可以透過將角色指派給與商業邏輯密切一致的使用者來完成，也可以根據其他內容屬性，例如一天中的時間或使用者是否符合預先定義的條件。存取控制模型構成應用程式在進行授權決策時用來判斷使用者許可的基本邏輯。

問：我的應用程式是否需要 API 存取控制？

A.是。APIs應一律驗證發起人是否具有適當的存取權。普遍 API 存取控制也可確保僅根據租用戶授予存取權，以便維持適當的隔離。

問：為什麼建議政策引擎或 PDPs進行授權？

A.政策決策點 (PDP) 允許將應用程式程式碼中的授權邏輯卸載至個別系統。這可以簡化應用程式程式碼。它還提供easy-to-use等冪界面，用於對 APIs、微服務、後端前端 (BFF) 層或任何其他應用程式元件進行授權決策。

問：什麼是 PEP？

A.政策強制執行點 (PEP) 負責接收傳送到 PDP 進行評估的授權請求。PEP 可以在必須保護資料和資源，或套用授權邏輯的應用程式中的任何位置。與 PDPs 相比PEPs 相對簡單。PEP 僅負責請求和評估授權決策，不需要將任何授權邏輯納入其中。

問：我應該如何選擇 Amazon Verified Permissions 和 OPA？

答：若要在 Verified Permissions 和 Open Policy Agent (OPA) 之間進行選擇，請謹記您的使用案例和唯一需求。Verified Permissions 提供完全受管的方式，可定義精細的許可、跨應用程式稽核許可，以及集中應用程式的政策管理系統，同時透過毫秒處理滿足您的應用程式延遲需求。OPA 是一種開放原始碼的一般用途政策引擎，也可協助您在應用程式堆疊中統一政策。若要執行 OPA，您需要在 AWS 環境中託管它，通常使用容器或 AWS Lambda 函數。

Verified Permissions 使用開放原始碼 Cedar 政策語言，而 OPA 使用 Rego。因此，熟悉其中一種語言可能會阻礙您選擇該解決方案。不過，我們建議您閱讀兩種語言的相關資訊，然後解決您嘗試解決的問題，為您的使用案例尋找最佳解決方案。

問：驗證許可和 OPA 是否有開放原始碼替代方案？

答：有一些開放原始碼系統類似於 Verified Permissions 和 Open Policy Agent (OPA)，例如 Common Expression Language (CEL)。本指南著重於 Verified Permissions，作為可擴展的許可管理和精細的授權服務，以及 OPA，其廣泛採用、記錄和適應許多不同類型的應用程式和授權要求。

問：我需要撰寫授權服務才能使用 OPA，還是可以直接與 OPA 互動？

答：您可以直接與 OPA 互動。本指南內容中的授權服務是指將授權決策請求轉譯為 OPA 查詢的服務，反之亦然。如果您的應用程式可以直接查詢並接受 OPA 回應，則不需要引入此額外的複雜性。

問：如何監控 OPA 代理器的運作時間和稽核目的？

A.OPA 提供記錄和基本運作時間監控，雖然其預設組態可能不足以進行企業部署。如需詳細資訊，請參閱本指南稍早的 DevOps、監控和記錄一節。

問：如何針對運作時間和稽核目的監控已驗證的許可？

A.Verified Permissions 是 AWS 受管服務，可透過 監控可用性 AWS Health 儀板表。此外，Verified Permissions 能夠記錄 AWS CloudTrail Amazon CloudWatch Logs、Amazon S3 和 Amazon Data Firehose。

問：我可以使用哪些作業系統 AWS 和服務來執行 OPA？

答：您可以在 macOS、Windows 和 Linux 上執行 OPA。您可以在 Amazon Elastic Compute Cloud (Amazon EC2) 代理程式以及容器化服務上設定 OPA 代理程式，例如 Amazon Elastic Container Service (Amazon ECS) 和 Amazon Elastic Kubernetes Service (Amazon EKS)。

問：我可以使用哪些作業系統和服務 AWS來執行 Verified Permissions？

A.Verified Permissions 是 AWS 受管服務，由 操作 AWS。使用 Verified Permissions 不需要額外的組態、安裝或託管，但向服務提出授權請求的功能除外。

問：我可以在 上執行 OPA AWS Lambda嗎？

答：您可以在 Lambda 上執行 OPA 做為 Go 程式庫。如需有關如何為 API Gateway Lambda 授權方執行此操作的詳細資訊，請參閱 AWS 部落格文章使用開放政策代理程式建立自訂 Lambda 授權方。

問：我應該如何決定分散式 PDP 和集中式 PDP 方法？

答：這取決於您的應用程式。這很可能會根據分散式和集中式 PDP 模型之間的延遲差異來決定。我們建議您建置概念驗證，並測試應用程式的效能，以驗證您的解決方案。

問：除了 APIs 之外，我可以使用 OPA 處理使用案例嗎？

A.是。OPA 文件提供 Kubernetes、Envoy、Docker、Kafka、SSH 和 sudo 以及 Terraform 的範例。此外，OPA 可以使用 Rego 部分規則將任意 JSON 回應傳回至查詢。根據查詢，OPA 可用於使用 JSON 回應來回答許多問題。

問：除了 API 之外，我可以針對使用案例使用 Verified Permissions APIs 嗎？

A.是。Verified Permissions 可為收到的任何授權請求提供 `ALLOW`或 `DENY`回應。Verified Permissions 可為需要授權決策的任何應用程式或服務提供授權回應。

問：我可以使用 IAM 政策語言在已驗證許可中建立政策嗎？

答：否。您必須使用 Cedar 政策語言來撰寫政策。Cedar 旨在支援客戶應用程式資源的許可管理，而 AWS Identity and Access Management (IAM) 政策語言則發展為支援 AWS 資源的存取控制。

後續步驟

透過採用標準化、與語言無關的方法來制定授權決策，可以解決多租用戶 SaaS 應用程式的授權和 API 存取控制的複雜性。這些方法包含政策決策點 PDPs) 和政策強制執行點 (PDPs)，以靈活且普遍的方式強制執行存取。多種存取控制方法 [例如角色型存取控制 (RBAC)、屬性型存取控制 (ABAC) 或兩者的組合] 可以整合到一個一致的存取控制策略。從應用程式中移除授權邏輯，可消除在應用程式程式碼中包含臨時解決方案來解決存取控制的額外負荷。本指南中討論的實作和最佳實務旨在通知和標準化多租用戶 SaaS 應用程式中授權和 API 存取控制的實作方法。您可以使用本指引作為收集資訊並為您的應用程式設計強大的存取控制和授權系統的第一步。後續步驟：

- 檢閱您的授權和租用戶隔離需求，並為您的應用程式選取存取控制模型。
- 使用 Amazon Verified Permissions 或 Open Policy Agent (OPA)，或編寫您自己的自訂政策引擎，來建置測試的概念驗證。
- 識別應用程式中應實作 PEP 的 API 和位置。

資源

參考

- Amazon Verified Permissions 文件 (AWS 文件）
- 如何使用 Amazon Verified Permissions 進行授權 (AWS 部落格文章）
- 使用 Amazon Verified Permissions 實作 ASP.NET Core Apps 的自訂授權政策提供者 (AWS 部落格文章）
- 使用 Amazon Verified Permissions 使用 PBAC 管理角色和權限 (AWS 部落格文章）
- 使用 Amazon Verified Permissions 搭配每個租用戶政策存放區的 SaaS 存取控制 (AWS 部落格文章）
- OPA 官方文件
- 企業為何必須接受最近畢業的 CNCF 專案 – 開放政策代理程式 (Forbes 文章作者：Janakiram MSV，2021 年 2 月 8 日）
- 使用開放政策代理程式建立自訂 Lambda 授權方 (AWS 部落格文章）
- 透過 AWS Open Policy Agent 將政策做為程式碼來實現 (AWS 部落格文章）
- AWS 將政策做為程式碼的雲端管理和合規 (AWS 部落格文章）
- 在 Amazon EKS 上使用開放政策代理程式 (AWS 部落格文章）
- 使用 Open Policy Agent、Amazon EventBridge 和 （部落格文章） 的 Amazon ECS 合規做為程式碼 AWS LambdaAWS
- Kubernetes 的政策型對策 – 第 1 部分 (AWS 部落格文章）
- 使用 API Gateway Lambda 授權方 (AWS 文件）

工具

- Cedar 遊樂場 （用於在瀏覽器中測試 Cedar)
- Cedar Github 儲存庫
- 雪松語言參考
- Rego 遊樂場 （用於在瀏覽器中測試 Rego)
- OPA GitHub 儲存庫

合作夥伴

- Identity and Access Management 合作夥伴
- 應用程式安全合作夥伴
- 雲端管理合作夥伴
- 安全與合規合作夥伴
- 安全操作和自動化合作夥伴
- 安全工程合作夥伴

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
新增 Amazon Verified Permissions 的詳細資訊和範例	新增使用 Amazon Verified Permissions 實作 PDP 的詳細討論、範例和程式碼。新章節包括： • 使用 Amazon Verified Permissions 實作 PDP • Amazon Verified Permissions 的設計模型 • Amazon Verified Permissions 多租戶設計考量事項 • 在 Amazon Verified Permissions 中擷取 PDP 的外部資料	2024 年 5 月 28 日
釐清資訊	釐清 API 設計模型上具有 PEPs 的分散式 PDP APIs。	2024 年 1 月 10 日
新增新 AWS 服務的簡短資訊	新增 Amazon Verified Permissions 的相關資訊，其提供與 OPA 相同的功能和優點。	2023 年 5 月 22 日
—	初次出版	2021 年 8 月 17 日

AWS 規範性指引詞彙表

以下是 AWS Prescriptive Guidance 提供的策略、指南和模式中常用的術語。若要建議項目，請使用詞彙表末尾的提供意見回饋連結。

數字

7 R

將應用程式移至雲端的七種常見遷移策略。這些策略以 Gartner 在 2011 年確定的 5 R 為基礎，包括以下內容：

- 重構/重新架構 – 充分利用雲端原生功能來移動應用程式並修改其架構，以提高敏捷性、效能和可擴展性。這通常涉及移植作業系統和資料庫。範例：將您的現場部署 Oracle 資料庫 遷移至 Amazon Aurora PostgreSQL 相容版本。
- 平台轉換 (隨即重塑) – 將應用程式移至雲端，並引入一定程度的優化以利用雲端功能。範例：將您的現場部署 Oracle 資料庫 遷移至 中的 Amazon Relational Database Service (Amazon RDS) for Oracle AWS 雲端。
- 重新購買 (捨棄再購買) – 切換至不同的產品，通常從傳統授權移至 SaaS 模型。範例：將您的客戶關係管理 (CRM) 系統 遷移至 Salesforce.com。
- 主機轉換 (隨即轉移) – 將應用程式移至雲端，而不進行任何變更以利用雲端功能。範例：將您的現場部署 Oracle 資料庫 遷移至 中 EC2 執行個體上的 Oracle AWS 雲端。
- 重新放置 (虛擬機器監視器等級隨即轉移) – 將基礎設施移至雲端，無需購買新硬體、重寫應用程式或修改現有操作。您可以將伺服器從內部部署平台遷移到相同平台的雲端服務。範例：將 Microsoft Hyper-V應用程式 遷移至 AWS。
- 保留 (重新檢視) – 將應用程式保留在來源環境中。其中可能包括需要重要重構的應用程式，且您希望將該工作延遲到以後，以及您想要保留的舊版應用程式，因為沒有業務理由來進行遷移。
- 淘汰 – 解除委任或移除來源環境中不再需要的應用程式。

A

A2A Agent-to-Agent)

支援任務委派和狀態轉移的agent-to-agent協同合作的狀態通訊協定。

ABAC

請參閱屬性型存取控制。

抽象服務

請參閱 受管服務。

ACID

請參閱原子性、一致性、隔離性、耐久性。

主動-主動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步 (透過使用雙向複寫工具或雙重寫入操作)，且兩個資料庫都在遷移期間處理來自連接應用程式的交易。此方法支援小型、受控制批次的遷移，而不需要一次性切換。它更靈活，但比主動-被動遷移需要更多工作。

主動-被動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步，但只有來源資料庫會在資料複寫至目標資料庫時處理來自連線應用程式的交易。目標資料庫在遷移期間不接受任何交易。

客服人員

一種 AI 系統，可以使用工具自動推理、規劃和採取行動來實現目標。

客服人員操作

在生產環境中大規模建置、測試、部署和執行 AI 代理器的操作實務。

彙總函數

在一組資料列上運作的 SQL 函數，會計算群組的單一傳回值。彙總函數的範例包括 SUM和 MAX。

AI

請參閱人工智慧。

AIOps

請參閱人工智慧操作。

匿名化

永久刪除資料集中個人資訊的程序。匿名化有助於保護個人隱私權。匿名資料不再被視為個人資料。

反模式

經常用於經常性問題的解決方案，其中解決方案具有反生產力、無效或比替代解決方案更有效。

應用程式控制

一種安全方法，僅允許使用核准的應用程式，以協助保護系統免受惡意軟體攻擊。

應用程式組合

有關組織使用的每個應用程式的詳細資訊的集合，包括建置和維護應用程式的成本及其商業價值。此資訊是產品組合探索和分析程序的關鍵，有助於識別要遷移、現代化和優化的應用程式並排定其優先順序。

人工智慧 (AI)

電腦科學領域，致力於使用運算技術來執行通常與人類相關的認知功能，例如學習、解決問題和識別模式。如需詳細資訊，請參閱什麼是人工智慧？

人工智慧操作 (AIOps)

使用機器學習技術解決操作問題、減少操作事件和人工干預以及提高服務品質的程序。如需有關如何在 AWS 遷移策略中使用 AIOps 的詳細資訊，請參閱操作整合指南。

非對稱加密

一種加密演算法，它使用一對金鑰：一個用於加密的公有金鑰和一個用於解密的私有金鑰。您可以共用公有金鑰，因為它不用於解密，但對私有金鑰存取應受到高度限制。

原子性、一致性、隔離性、耐久性 (ACID)

一組軟體屬性，即使在出現錯誤、電源故障或其他問題的情況下，也能確保資料庫的資料有效性和操作可靠性。

屬性型存取控制 (ABAC)

根據使用者屬性 (例如部門、工作職責和團隊名稱) 建立精細許可的實務。如需詳細資訊，請參閱《 AWS Identity and Access Management (IAM) 文件》中的 ABAC for AWS。

授權資料來源

存放主要版本資料的位置，被視為最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以處理或修改資料，例如匿名、修訂或假名化資料。

可用區域

中的不同位置 AWS 區域 ，可隔離其他可用區域中的故障，並提供相同區域中其他可用區域的低成本、低延遲網路連線。

AWS 雲端採用架構 (AWS CAF)

的指導方針和最佳實務架構 AWS ，可協助組織制定高效且有效的計劃，以成功地移至雲端。 AWS CAF 將指導方針組織到六個重點領域：業務、人員、治理、平台、安全和營運。業務、人員和控管層面著重於業務技能和程序；平台、安全和操作層面著重於技術技能和程序。例如，人員層面針對處理人力資源 (HR)、人員配備功能和人員管理的利害關係人。因此， AWS CAF 為人員開發、訓練和通訊提供指引，協助組織做好成功採用雲端的準備。如需詳細資訊，請參閱 AWS CAF 網站和 AWS CAF 白皮書。

AWS 工作負載資格架構 (AWS WQF)

一種工具，可評估資料庫遷移工作負載、建議遷移策略，並提供工作預估值。 AWS WQF 隨附於 AWS Schema Conversion Tool (AWS SCT)。它會分析資料庫結構描述和程式碼物件、應用程式程式碼、相依性和效能特性，並提供評估報告。

B

錯誤的機器人

旨在中斷或傷害個人或組織的機器人。

BCP

請參閱業務持續性規劃。

行為圖

資源行為的統一互動式檢視，以及一段時間後的互動。您可以將行為圖與 Amazon Detective 搭配使用來檢查失敗的登入嘗試、可疑的 API 呼叫和類似動作。如需詳細資訊，請參閱偵測文件中的行為圖中的資料。

大端序系統

首先儲存最高有效位元組的系統。另請參閱 Endianness。

二進制分類

預測二進制結果的過程 (兩個可能的類別之一)。例如，ML 模型可能需要預測諸如「此電子郵件是否是垃圾郵件？」等問題 或「產品是書還是汽車？」

Bloom 篩選條件

一種機率性、記憶體高效的資料結構，用於測試元素是否為集的成員。

藍/綠部署

一種部署策略，您可以在其中建立兩個不同但相同的環境。您可以在一個環境（藍色）中執行目前的應用程式版本，並在另一個環境（綠色）中執行新的應用程式版本。此策略可協助您快速復原，並將影響降至最低。

機器人

透過網際網路執行自動化任務並模擬人類活動或互動的軟體應用程式。有些機器人有用或有益，例如在網際網路上編製資訊索引的 Web 爬蟲程式。某些其他機器人稱為惡意機器人，旨在中斷或傷害個人或組織。

殭屍網路

受到惡意軟體感染且受單一方控制之機器人的網路，稱為機器人繼承器或機器人運算子。殭屍網路是擴展機器人及其影響的最佳已知機制。

分支

程式碼儲存庫包含的區域。儲存庫中建立的第一個分支是主要分支。您可以從現有分支建立新分支，然後在新分支中開發功能或修正錯誤。您建立用來建立功能的分支通常稱為功能分支。當準備好發佈功能時，可以將功能分支合併回主要分支。如需詳細資訊，請參閱關於分支 (GitHub 文件)。

碎片存取

在特殊情況下，並透過核准的程序，讓使用者快速取得他們通常無權存取 AWS 帳戶 之 的存取權。如需詳細資訊，請參閱 Well-Architected 指南中的 AWS 實作打破玻璃程序指標。

棕地策略

環境中的現有基礎設施。對系統架構採用棕地策略時，可以根據目前系統和基礎設施的限制來設計架構。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

緩衝快取

儲存最常存取資料的記憶體區域。

業務能力

業務如何創造價值 (例如，銷售、客戶服務或營銷)。業務能力可驅動微服務架構和開發決策。如需詳細資訊，請參閱在 AWS上執行容器化微服務白皮書的圍繞業務能力進行組織部分。

業務連續性規劃 (BCP)

一種解決破壞性事件 (如大規模遷移) 對營運的潛在影響並使業務能夠快速恢復營運的計畫。

C

CAF

請參閱AWS 雲端採用架構。

Canary 部署

版本對最終使用者的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前的版本。

CCoE

請參閱 Cloud Center of Excellence。

CDC

請參閱變更資料擷取。

變更資料擷取 (CDC)

追蹤對資料來源 (例如資料庫表格) 的變更並記錄有關變更的中繼資料的程序。您可以將 CDC 用於各種用途，例如稽核或複寫目標系統中的變更以保持同步。

混沌工程

故意引入故障或破壞性事件，以測試系統的彈性。您可以使用 AWS Fault Injection Service (AWS FIS) 執行實驗，為您的 AWS 工作負載帶來壓力，並評估其回應。

CI/CD

請參閱持續整合和持續交付。

分類

有助於產生預測的分類程序。用於分類問題的 ML 模型可預測離散值。離散值永遠彼此不同。例如，模型可能需要評估影像中是否有汽車。

公民開發人員

不使用專業技術技能，而使用無程式碼/低程式碼平台建立 AI 應用程式的商業使用者。

用戶端加密

在目標 AWS 服務 接收資料之前，在本機加密資料。

雲端卓越中心 (CCoE)

一個多學科團隊，可推動整個組織的雲端採用工作，包括開發雲端最佳實務、調動資源、制定遷移時間表以及領導組織進行大規模轉型。如需詳細資訊，請參閱 AWS 雲端 企業策略部落格上的 CCoE 文章。

雲端運算

通常用於遠端資料儲存和 IoT 裝置管理的雲端技術。雲端運算通常連接到邊緣運算技術。

雲端操作模型

在 IT 組織中，用於建置、成熟和最佳化一或多個雲端環境的操作模型。如需詳細資訊，請參閱建置您的雲端操作模型。

採用雲端階段

組織在遷移至 時通常會經歷的四個階段 AWS 雲端：

- 專案 – 執行一些與雲端相關的專案以進行概念驗證和學習用途
- 基礎 – 進行基礎投資以擴展雲端採用 (例如，建立登陸區域、定義 CCoE、建立營運模型)
- 遷移 – 遷移個別應用程式
- 重塑 – 優化產品和服務，並在雲端中創新

這些階段由 Stephen Orban 在部落格文章 The Journey Toward Cloud-First 和 Enterprise Strategy 部落格上的採用階段中定義。 AWS 雲端 如需有關它們如何與 AWS 遷移策略相關的詳細資訊，請參閱遷移整備指南。

CMDB

請參閱組態管理資料庫。

程式碼儲存庫

透過版本控制程序來儲存及更新原始程式碼和其他資產 (例如文件、範例和指令碼) 的位置。常見的雲端儲存庫包括 GitHub或 Bitbucket Cloud。程式碼的每個版本都稱為分支。在微服務結構中，每個儲存庫都專用於單個功能。單一 CI/CD 管道可以使用多個儲存庫。

冷快取

一種緩衝快取，它是空的、未填充的，或者包含過時或不相關的資料。這會影響效能，因為資料庫執行個體必須從主記憶體或磁碟讀取，這比從緩衝快取讀取更慢。

冷資料

很少存取且通常是歷史資料的資料。查詢這類資料時，通常可接受慢查詢。將此資料移至效能較低且成本較低的儲存層或類別，可以降低成本。

電腦視覺 (CV)

AI 欄位???，使用機器學習從數位影像和影片等視覺化格式分析和擷取資訊。例如，Amazon SageMaker AI 提供 CV 的影像處理演算法。

組態偏離

對於工作負載，組態會從預期狀態變更。這可能會導致工作負載不合規，而且通常是漸進和無意的。

組態管理資料庫 (CMDB)

儲存和管理有關資料庫及其 IT 環境的資訊的儲存庫，同時包括硬體和軟體元件及其組態。您通常在遷移的產品組合探索和分析階段使用 CMDB 中的資料。

一致性套件

您可以組合的 AWS Config 規則和修補動作集合，以自訂您的合規和安全檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和 區域中或整個組織的單一實體。如需詳細資訊，請參閱 AWS Config 文件中的一致性套件。

持續整合和持續交付 (CI/CD)

自動化軟體發行程序的來源、建置、測試、暫存和生產階段的程序。CI/CD 通常被描述為管道。CI/CD 可協助您將程序自動化、提升生產力、改善程式碼品質以及加快交付速度。如需詳細資訊，請參閱持續交付的優點。CD 也可表示持續部署。如需詳細資訊，請參閱持續交付與持續部署。

CV

請參閱電腦視覺。

D

靜態資料

網路中靜止的資料，例如儲存中的資料。

資料分類

根據重要性和敏感性來識別和分類網路資料的程序。它是所有網路安全風險管理策略的關鍵組成部分，因為它可以協助您確定適當的資料保護和保留控制。資料分類是 AWS Well-Architected Framework 中安全支柱的元件。如需詳細資訊，請參閱資料分類。

資料偏離

生產資料與用於訓練 ML 模型的資料之間有意義的變化，或輸入資料隨時間有意義的變更。資料偏離可以降低 ML 模型預測的整體品質、準確性和公平性。

傳輸中的資料

在您的網路中主動移動的資料，例如在網路資源之間移動。

資料網格

架構架構，提供分散式、分散式資料擁有權與集中式管理。

資料最小化

僅收集和處理嚴格必要資料的原則。在 中實作資料最小化 AWS 雲端 可以降低隱私權風險、成本和分析碳足跡。

資料周邊

AWS 環境中的一組預防性防護機制，可協助確保只有信任的身分才能從預期的網路存取信任的資源。如需詳細資訊，請參閱在 上建置資料周邊 AWS。

資料預先處理

將原始資料轉換成 ML 模型可輕鬆剖析的格式。預處理資料可能意味著移除某些欄或列，並解決遺失、不一致或重複的值。

資料來源

在整個生命週期中追蹤資料的原始伺服器和歷史記錄的程序，例如資料的產生、傳輸和儲存方式。

資料主體

正在收集和處理其資料的個人。

資料倉儲

支援商業智慧的資料管理系統，例如 分析。資料倉儲通常包含大量歷史資料，通常用於查詢和分析。

資料庫定義語言 (DDL)

用於建立或修改資料庫中資料表和物件之結構的陳述式或命令。

資料庫處理語言 (DML)

用於修改 (插入、更新和刪除) 資料庫中資訊的陳述式或命令。

DDL

請參閱資料庫定義語言。

深度整體

結合多個深度學習模型進行預測。可以使用深度整體來獲得更準確的預測或估計預測中的不確定性。

深度學習

一個機器學習子領域，它使用多層人工神經網路來識別感興趣的輸入資料與目標變數之間的對應關係。

深度防禦

這是一種資訊安全方法，其中一系列的安全機制和控制項會在整個電腦網路中精心分層，以保護網路和其中資料的機密性、完整性和可用性。當您在 上採用此策略時 AWS，您可以在 AWS Organizations 結構的不同層新增多個控制項，以協助保護資源。例如，defense-in-depth方法可能會結合多重要素驗證、網路分割和加密。

委派的管理員

在 中 AWS Organizations，相容的服務可以註冊 AWS 成員帳戶來管理組織的帳戶，並管理該服務的許可。此帳戶稱為該服務的委派管理員。如需詳細資訊和相容服務清單，請參閱 AWS Organizations 文件中的可搭配 AWS Organizations運作的服務。

deployment

在目標環境中提供應用程式、新功能或程式碼修正的程序。部署涉及在程式碼庫中實作變更，然後在應用程式環境中建置和執行該程式碼庫。

開發環境

請參閱 環境。

偵測性控制

一種安全控制，用於在事件發生後偵測、記錄和提醒。這些控制是第二道防線，提醒您注意繞過現有預防性控制的安全事件。如需詳細資訊，請參閱在 AWS上實作安全控制中的偵測性控制。

開發值串流映射 (DVSM)

一種程序，用於識別對軟體開發生命週期中的速度和品質造成負面影響的限制並排定優先順序。DVSM 擴展了最初專為精簡製造實務設計的價值串流映射程序。它著重於透過軟體開發程序建立和移動價值所需的步驟和團隊。

數位分身

虛擬呈現真實世界的系統，例如建築物、工廠、工業設備或生產線。數位分身支援預測性維護、遠端監控和生產最佳化。

維度資料表

在星星結構描述中，較小的資料表包含有關事實資料表中量化資料的資料屬性。維度資料表屬性通常是文字欄位或離散數字，其行為類似於文字。這些屬性通常用於查詢限制、篩選和結果集標記。

災難

防止工作負載或系統在其主要部署位置實現其業務目標的事件。這些事件可能是自然災難、技術故障或人為動作的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您用來將災難造成的停機時間和資料遺失降至最低的策略和程序。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的 上工作負載災難復原 AWS：雲端中的復原。

DML

請參閱資料庫處理語言。

領域驅動的設計

一種開發複雜軟體系統的方法，它會將其元件與每個元件所服務的不斷發展的領域或核心業務目標相關聯。Eric Evans 在其著作 Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003) 中介紹了這一概念。如需有關如何將領域驅動的設計與 strangler fig 模式搭配使用的資訊，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

DR

請參閱災難復原。

偏離偵測

追蹤與基準組態的偏差。例如，您可以使用 AWS CloudFormation 來偵測系統資源中的偏離，也可以使用 AWS Control Tower 來偵測登陸區域中可能影響控管要求合規性的變更。 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html

DVSM

請參閱開發值串流映射。

E

EDA

請參閱探索性資料分析。

EDI

請參閱電子資料交換。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與雲端運算相比，邊緣運算可以減少通訊延遲並改善回應時間。

電子資料交換 (EDI)

在組織之間自動交換商業文件。如需詳細資訊，請參閱什麼是電子資料交換。

加密

將人類可讀取的純文字資料轉換為加密文字的運算程序。

加密金鑰

由加密演算法產生的隨機位元的加密字串。金鑰長度可能有所不同，每個金鑰的設計都是不可預測且唯一的。

端序

位元組在電腦記憶體中的儲存順序。大端序系統首先儲存最高有效位元組。小端序系統首先儲存最低有效位元組。

端點

請參閱 服務端點。

端點服務

您可以在虛擬私有雲端 (VPC) 中託管以與其他使用者共用的服務。您可以使用 建立端點服務，AWS PrivateLink 並將許可授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 委託人。這些帳戶或主體可以透過建立介面 VPC 端點私下連接至您的端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud (Amazon VPC) 文件中的建立端點服務。

企業資源規劃 (ERP)

一種系統，可自動化和管理企業的關鍵業務流程 （例如會計、MES 和專案管理）。

信封加密

使用另一個加密金鑰對某個加密金鑰進行加密的程序。如需詳細資訊，請參閱 AWS Key Management Service (AWS KMS) 文件中的信封加密。

環境

執行中應用程式的執行個體。以下是雲端運算中常見的環境類型：

- 開發環境 – 執行中應用程式的執行個體，只有負責維護應用程式的核心團隊才能使用。開發環境用來測試變更，然後再將開發環境提升到較高的環境。此類型的環境有時稱為測試環境。
- 較低的環境 – 應用程式的所有開發環境，例如用於初始建置和測試的開發環境。
- 生產環境 – 最終使用者可以存取的執行中應用程式的執行個體。在 CI/CD 管道中，生產環境是最後一個部署環境。
- 較高的環境 – 核心開發團隊以外的使用者可存取的所有環境。這可能包括生產環境、生產前環境以及用於使用者接受度測試的環境。

epic

在敏捷方法中，有助於組織工作並排定工作優先順序的功能類別。epic 提供要求和實作任務的高層級描述。例如， AWS CAF 安全概念包括身分和存取管理、偵測控制、基礎設施安全、資料保護和事件回應。如需有關 AWS 遷移策略中的 Epic 的詳細資訊，請參閱計畫實作指南。

ERP

請參閱企業資源規劃。

探索性資料分析 (EDA)

分析資料集以了解其主要特性的過程。您收集或彙總資料，然後執行初步調查以尋找模式、偵測異常並檢查假設。透過計算摘要統計並建立資料可視化來執行 EDA。

F

事實資料表

星狀結構描述中的中央資料表。它存放有關業務操作的量化資料。一般而言，事實資料表包含兩種類型的資料欄：包含度量的資料，以及包含維度資料表外部索引鍵的資料欄。

快速失敗

一種使用頻繁和增量測試來縮短開發生命週期的理念。這是敏捷方法的關鍵部分。

故障隔離界限

在 中 AWS 雲端，像是可用區域 AWS 區域、控制平面或資料平面等界限會限制故障的影響，並有助於改善工作負載的彈性。如需詳細資訊，請參閱AWS 故障隔離界限。

功能分支

請參閱分支。

特徵

用來進行預測的輸入資料。例如，在製造環境中，特徵可能是定期從製造生產線擷取的影像。

功能重要性

特徵對於模型的預測有多重要。這通常表示為可以透過各種技術來計算的數值得分，例如 Shapley Additive Explanations (SHAP) 和積分梯度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

特徵轉換

優化 ML 程序的資料，包括使用其他來源豐富資料、調整值、或從單一資料欄位擷取多組資訊。這可讓 ML 模型從資料中受益。例如，如果將「2021-05-27 00:15:37」日期劃分為「2021」、「五月」、「週四」和「15」，則可以協助學習演算法學習與不同資料元件相關聯的細微模式。

少量擷取提示

在要求 LLM 執行類似的任務之前，提供少量示範任務和所需輸出的範例。此技術是內容內學習的應用程式，其中模型會從內嵌在提示中的範例 (快照) 中學習。對於需要特定格式、推理或網域知識的任務，少量的提示非常有效。另請參閱零鏡頭提示。

FGAC

請參閱精細存取控制。

精細存取控制 (FGAC)

使用多個條件來允許或拒絕存取請求。

閃切遷移

一種資料庫遷移方法，透過變更資料擷取使用連續資料複寫，以盡可能在最短的時間內遷移資料，而不是使用分階段方法。目標是將停機時間降至最低。

FM

請參閱基礎模型。

基礎模型 (FM)

大型深度學習神經網路，已針對廣義和未標記資料的大量資料集進行訓練。FMs 能夠執行各種一般任務，例如了解語言、產生文字和影像，以及自然語言的交談。如需詳細資訊，請參閱什麼是基礎模型。

FM 閘道

控制和標準化基礎模型存取的集中式中介裝置。也稱為 LLM 閘道。

G

生成式 AI

已針對大量資料進行訓練的 AI 模型子集，可使用簡單的文字提示建立新的內容和成品，例如影像、影片、文字和音訊。如需詳細資訊，請參閱什麼是生成式 AI。

地理封鎖

請參閱地理限制。

地理限制 (地理封鎖)

Amazon CloudFront 中的選項，可防止特定國家/地區的使用者存取內容分發。您可以使用允許清單或封鎖清單來指定核准和禁止的國家/地區。如需詳細資訊，請參閱 CloudFront 文件中的限制內容的地理分佈。

Gitflow 工作流程

這是一種方法，其中較低和較高環境在原始碼儲存庫中使用不同分支。Gitflow 工作流程被視為舊版，而以幹線為基礎的工作流程是現代、偏好的方法。

黃金影像

系統或軟體的快照，做為部署該系統或軟體新執行個體的範本。例如，在製造中，黃金映像可用於在多個裝置上佈建軟體，並有助於提高裝置製造操作的速度、可擴展性和生產力。

綠地策略

新環境中缺乏現有基礎設施。對系統架構採用綠地策略時，可以選擇所有新技術，而不會限制與現有基礎設施的相容性，也稱為棕地。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

防護機制

有助於跨組織單位 (OU) 來管控資源、政策和合規的高層級規則。預防性防護機制會強制執行政策，以確保符合合規標準。透過使用服務控制政策和 IAM 許可界限來將其實作。偵測性防護機制可偵測政策違規和合規問題，並產生提醒以便修正。它們是透過使用 AWS Config AWS Security Hub CSPM、Amazon GuardDuty、、Amazon Inspector AWS Trusted Advisor和自訂 AWS Lambda 檢查來實作。

護欄 (AI)

安全機制可篩選、驗證和限制代理程式輸入和輸出，以協助確保負責任且安全的 AI 行為。

H

HA

請參閱高可用性。

異質資料庫遷移

將來源資料庫遷移至使用不同資料庫引擎的目標資料庫 (例如，Oracle 至 Amazon Aurora)。異質遷移通常是重新架構工作的一部分，而轉換結構描述可能是一項複雜任務。AWS 提供有助於結構描述轉換的 AWS SCT。

高可用性 (HA)

在遇到挑戰或災難時，工作負載能夠在不介入的情況下持續運作。HA 系統的設計目的是自動容錯移轉、持續提供高品質的效能，並處理不同的負載和故障，並將效能影響降至最低。

歷史現代化

一種方法，用於現代化和升級操作技術 (OT) 系統，以更好地滿足製造業的需求。歷史資料是一種資料庫，用於從工廠中的各種來源收集和存放資料。

保留資料

從用於訓練機器學習模型的資料集中保留的部分歷史標記資料。您可以使用保留資料，透過比較模型預測與保留資料來評估模型效能。

human-in-the-loop (HitL)

一種工作流程模式，其中代理程式執行會在關鍵決策點暫停進行人工審核和核准。

異質資料庫遷移

將您的來源資料庫遷移至共用相同資料庫引擎的目標資料庫 (例如，Microsoft SQL Server 至 Amazon RDS for SQL Server)。同質遷移通常是主機轉換或平台轉換工作的一部分。您可以使用原生資料庫公用程式來遷移結構描述。

熱資料

經常存取的資料，例如即時資料或最近的轉譯資料。此資料通常需要高效能儲存層或類別，才能提供快速的查詢回應。

修補程序

緊急修正生產環境中的關鍵問題。由於其緊迫性，通常會在典型 DevOps 發行工作流程之外執行修補程式。

超級護理期間

在切換後，遷移團隊在雲端管理和監控遷移的應用程式以解決任何問題的時段。通常，此期間的長度為 1-4 天。在超級護理期間結束時，遷移團隊通常會將應用程式的責任轉移給雲端營運團隊。

I

IaC

請參閱基礎設施即程式碼。

身分型政策

連接至一或多個 IAM 主體的政策，可定義其在 AWS 雲端 環境中的許可。

閒置應用程式

90 天期間 CPU 和記憶體平均使用率在 5% 至 20% 之間的應用程式。在遷移專案中，通常會淘汰這些應用程式或將其保留在內部部署。

IIoT

請參閱工業物聯網。

不可變的基礎設施

為生產工作負載部署新基礎設施的模型，而不是更新、修補或修改現有的基礎設施。不可變基礎設施本質上比可變基礎設施更一致、可靠且可預測。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的使用不可變基礎設施部署最佳實務。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，接受、檢查和路由來自應用程式外部之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

增量遷移

一種切換策略，您可以在其中將應用程式分成小部分遷移，而不是執行單一、完整的切換。例如，您最初可能只將一些微服務或使用者移至新系統。確認所有項目都正常運作之後，您可以逐步移動其他微服務或使用者，直到可以解除委任舊式系統。此策略可降低與大型遷移關聯的風險。

工業 4.0

2016 年 Klaus Schwab 推出的術語，透過連線能力、即時資料、自動化、分析和 AI/ML 的進展，指製造程序的現代化。

基礎設施

應用程式環境中包含的所有資源和資產。

基礎設施即程式碼 (IaC)

透過一組組態檔案來佈建和管理應用程式基礎設施的程序。IaC 旨在協助您集中管理基礎設施，標準化資源並快速擴展，以便新環境可重複、可靠且一致。

工業物聯網 (IIoT)

在製造業、能源、汽車、醫療保健、生命科學和農業等產業領域使用網際網路連線的感測器和裝置。如需詳細資訊，請參閱建立工業物聯網 (IIoT) 數位轉型策略。

檢查 VPC

在 AWS 多帳戶架構中，集中式 VPC，可管理 VPCs 之間 （在相同或不同的 中 AWS 區域)、網際網路和內部部署網路之間的網路流量檢查。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

物聯網 (IoT)

具有內嵌式感測器或處理器的相連實體物體網路，其透過網際網路或本地通訊網路與其他裝置和系統進行通訊。如需詳細資訊，請參閱什麼是 IoT？

可解釋性

機器學習模型的一個特徵，描述了人類能夠理解模型的預測如何依賴於其輸入的程度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

IoT

請參閱物聯網。

IT 資訊庫 (ITIL)

一組用於交付 IT 服務並使這些服務與業務需求保持一致的最佳實務。ITIL 為 ITSM 提供了基礎。

IT 服務管理 (ITSM)

與組織的設計、實作、管理和支援 IT 服務關聯的活動。如需有關將雲端操作與 ITSM 工具整合的資訊，請參閱操作整合指南。

ITIL

請參閱 IT 資訊庫。

ITSM

請參閱 IT 服務管理。

L

標籤型存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中使用者和資料本身都會獲得明確指派的安全標籤值。使用者安全標籤和資料安全標籤之間的交集會決定使用者可以看到哪些資料列和資料欄。

登陸區域

登陸區域是架構良好的多帳戶 AWS 環境，可擴展且安全。這是一個起點，您的組織可以從此起點快速啟動和部署工作負載與應用程式，並對其安全和基礎設施環境充滿信心。如需有關登陸區域的詳細資訊，請參閱設定安全且可擴展的多帳戶 AWS 環境。

大型語言模型 (LLM)

預先訓練大量資料的深度學習 AI 模型。LLM 可以執行多個任務，例如回答問題、摘要文件、將文字翻譯成其他語言，以及完成句子。如需詳細資訊，請參閱什麼是 LLMs。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱標籤型存取控制。

最低權限

授予執行任務所需之最低許可的安全最佳實務。如需詳細資訊，請參閱 IAM 文件中的套用最低權限許可。

隨即轉移

請參閱 7 個 R。

小端序系統

首先儲存最低有效位元組的系統。另請參閱 Endianness。

LLM

請參閱大型語言模型。

較低的環境

請參閱 環境。

M

機器學習 (ML)

一種使用演算法和技術進行模式識別和學習的人工智慧。機器學習會進行分析並從記錄的資料 (例如物聯網 (IoT) 資料) 中學習，以根據模式產生統計模型。如需詳細資訊，請參閱機器學習。

主要分支

請參閱分支。

惡意軟體

旨在危及電腦安全或隱私權的軟體。惡意軟體可能會中斷電腦系統、洩露敏感資訊，或取得未經授權的存取。惡意軟體的範例包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄器。

受管服務

AWS 服務 會 AWS 操作基礎設施層、作業系統和平台，而您會存取端點來存放和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也稱為抽象服務。

製造執行系統 (MES)

一種軟體系統，用於追蹤、監控、記錄和控制生產程序，將原物料轉換為現場成品。

MAP

請參閱遷移加速計劃。

MCP

請參閱模型內容通訊協定。

模型內容通訊協定 (MCP)

適用於代理程式對工具通訊的無狀態通訊協定。

MCP 伺服器

透過模型內容通訊協定公開一或多個工具的服務。

機制

建立工具、推動工具採用，然後檢查結果以進行調整的完整程序。機制是在操作時強化和改善自身的循環。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的建置機制。

成員帳戶

屬於組織一部分的管理帳戶 AWS 帳戶 以外的所有 AWS Organizations。帳戶一次只能是一個組織的成員。

製造執行系統

請參閱製造執行系統。

訊息佇列遙測傳輸 (MQTT)

根據發佈/訂閱模式的輕量型machine-to-machine(M2M) 通訊協定，適用於資源受限的 IoT 裝置。

微服務

一種小型的獨立服務，它可透過定義明確的 API 進行通訊，通常由小型獨立團隊擁有。例如，保險系統可能包含對應至業務能力 (例如銷售或行銷) 或子領域 (例如購買、索賠或分析) 的微服務。微服務的優點包括靈活性、彈性擴展、輕鬆部署、可重複使用的程式碼和適應力。如需詳細資訊，請參閱使用無 AWS 伺服器服務整合微服務。

微服務架構

一種使用獨立元件來建置應用程式的方法，這些元件會以微服務形式執行每個應用程式程序。這些微服務會使用輕量型 API，透過明確定義的介面進行通訊。此架構中的每個微服務都可以進行更新、部署和擴展，以滿足應用程式特定功能的需求。如需詳細資訊，請參閱在 上實作微服務 AWS。

Migration Acceleration Program (MAP)

此 AWS 計畫提供諮詢支援、訓練和服務，以協助組織建立強大的營運基礎，以移至雲端，並協助抵銷遷移的初始成本。MAP 包括用於有條不紊地執行舊式遷移的遷移方法以及一組用於自動化和加速常見遷移案例的工具。

大規模遷移

將大部分應用程式組合依波次移至雲端的程序，在每個波次中，都會以更快的速度移動更多應用程式。此階段使用從早期階段學到的最佳實務和經驗教訓來實作團隊、工具和流程的遷移工廠，以透過自動化和敏捷交付簡化工作負載的遷移。這是 AWS 遷移策略的第三階段。

遷移工廠

可透過自動化、敏捷的方法簡化工作負載遷移的跨職能團隊。遷移工廠團隊通常包括營運、業務分析師和擁有者、遷移工程師、開發人員以及從事 Sprint 工作的 DevOps 專業人員。20% 至 50% 之間的企業應用程式組合包含可透過工廠方法優化的重複模式。如需詳細資訊，請參閱此內容集中的遷移工廠的討論和雲端遷移工廠指南。

遷移中繼資料

有關完成遷移所需的應用程式和伺服器的資訊。每種遷移模式都需要一組不同的遷移中繼資料。遷移中繼資料的範例包括目標子網路、安全群組和 AWS 帳戶。

遷移模式

可重複的遷移任務，詳細描述遷移策略、遷移目的地以及所使用的遷移應用程式或服務。範例：使用 AWS Application Migration Service 重新託管遷移至 Amazon EC2。

遷移組合評定 (MPA)

線上工具，提供驗證商業案例以遷移至 的資訊 AWS 雲端。MPA 提供詳細的組合評定 (伺服器適當規模、定價、總體擁有成本比較、遷移成本分析) 以及遷移規劃 (應用程式資料分析和資料收集、應用程式分組、遷移優先順序，以及波次規劃)。MPA 工具 （需要登入） 可供所有 AWS 顧問和 APN 合作夥伴顧問免費使用。

遷移準備程度評定 (MRA)

使用 AWS CAF 取得組織雲端整備狀態的洞見、識別優缺點，以及建立行動計劃以消除已識別差距的程序。如需詳細資訊，請參閱遷移準備程度指南。MRA 是 AWS 遷移策略的第一階段。

遷移策略

用來將工作負載遷移至 的方法 AWS 雲端。如需詳細資訊，請參閱本詞彙表中的 7 個 Rs 項目，並請參閱動員您的組織以加速大規模遷移。

機器學習 (ML)

請參閱機器學習。

現代化

將過時的 (舊版或單一) 應用程式及其基礎架構轉換為雲端中靈活、富有彈性且高度可用的系統，以降低成本、提高效率並充分利用創新。如需詳細資訊，請參閱《》中的現代化應用程式的策略 AWS 雲端。

現代化準備程度評定

這項評估可協助判斷組織應用程式的現代化準備程度；識別優點、風險和相依性；並確定組織能夠在多大程度上支援這些應用程式的未來狀態。評定的結果就是目標架構的藍圖、詳細說明現代化程序的開發階段和里程碑的路線圖、以及解決已發現的差距之行動計畫。如需詳細資訊，請參閱《》中的評估應用程式的現代化準備 AWS 雲端程度。

單一應用程式 (單一)

透過緊密結合的程序作為單一服務執行的應用程式。單一應用程式有幾個缺點。如果一個應用程式功能遇到需求激增，則必須擴展整個架構。當程式碼庫增長時，新增或改進單一應用程式的功能也會變得更加複雜。若要解決這些問題，可以使用微服務架構。如需詳細資訊，請參閱將單一體系分解為微服務。

MPA

請參閱遷移產品組合評估。

MQTT

請參閱訊息佇列遙測傳輸。

多類別分類

一個有助於產生多類別預測的過程 (預測兩個以上的結果之一)。例如，機器學習模型可能會詢問「此產品是書籍、汽車還是電話？」 或者「這個客戶對哪種產品類別最感興趣？」

可變基礎設施

更新和修改生產工作負載現有基礎設施的模型。為了提高一致性、可靠性和可預測性， AWS Well-Architected Framework 建議使用不可變的基礎設施作為最佳實務。

O

OAC

請參閱原始存取控制。

OAI

請參閱原始存取身分。

OCM

請參閱組織變更管理。

離線遷移

一種遷移方法，可在遷移過程中刪除來源工作負載。此方法涉及延長停機時間，通常用於小型非關鍵工作負載。

OI

請參閱 操作整合。

OLA

請參閱操作層級協議。

線上遷移

一種遷移方法，無需離線即可將來源工作負載複製到目標系統。連接至工作負載的應用程式可在遷移期間繼續運作。此方法涉及零至最短停機時間，通常用於關鍵的生產工作負載。

OPC-UA

請參閱開放程序通訊 - 統一架構。

開放程序通訊 - 統一架構 (OPC-UA)

用於工業自動化machine-to-machine(M2M) 通訊協定。OPC-UA 提供資料加密、身分驗證和授權機制的互通性標準。

操作水準協議 (OLA)

一份協議，闡明 IT 職能群組承諾向彼此提供的內容，以支援服務水準協議 (SLA)。

操作整備審查 (ORR)

問題及相關最佳實務的檢查清單，可協助您了解、評估、預防或減少事件和可能失敗的範圍。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的操作整備審查 (ORR)。

操作技術 (OT)

使用實體環境控制工業操作、設備和基礎設施的硬體和軟體系統。在製造中，整合 OT 和資訊技術 (IT) 系統是工業 4.0 轉型的關鍵重點。

操作整合 (OI)

在雲端中將操作現代化的程序，其中包括準備程度規劃、自動化和整合。如需詳細資訊，請參閱操作整合指南。

組織追蹤

由 建立的線索 AWS CloudTrail 會記錄 AWS 帳戶 組織中所有 的所有事件 AWS Organizations。在屬於組織的每個 AWS 帳戶 中建立此追蹤，它會跟蹤每個帳戶中的活動。如需詳細資訊，請參閱 CloudTrail 文件中的建立組織追蹤。

組織變更管理 (OCM)

用於從人員、文化和領導力層面管理重大、顛覆性業務轉型的架構。OCM 透過加速變更採用、解決過渡問題，以及推動文化和組織變更，協助組織為新系統和策略做好準備，並轉移至新系統和策略。在 AWS 遷移策略中，此架構稱為人員加速，因為雲端採用專案所需的變更速度。如需詳細資訊，請參閱 OCM 指南。

原始存取控制 (OAC)

CloudFront 中的增強型選項，用於限制存取以保護 Amazon Simple Storage Service (Amazon S3) 內容。OAC 支援所有 S3 儲存貯體中的所有伺服器端加密 AWS KMS (SSE-KMS) AWS 區域，以及對 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

CloudFront 中的一個選項，用於限制存取以保護 Amazon S3 內容。當您使用 OAI 時，CloudFront 會建立一個可供 Amazon S3 進行驗證的主體。經驗證的主體只能透過特定 CloudFront 分發來存取 S3 儲存貯體中的內容。另請參閱 OAC，它可提供更精細且增強的存取控制。

ORR

請參閱操作整備審核。

OT

請參閱操作技術。

傳出 (輸出) VPC

在 AWS 多帳戶架構中，處理從應用程式內啟動之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

P

許可界限

附接至 IAM 主體的 IAM 管理政策，可設定使用者或角色擁有的最大許可。如需詳細資訊，請參閱 IAM 文件中的許可界限。

個人身分識別資訊 (PII)

直接檢視或與其他相關資料配對時，可用來合理推斷個人身分的資訊。PII 的範例包括名稱、地址和聯絡資訊。

PII

請參閱個人身分識別資訊。

手冊

一組預先定義的步驟，可擷取與遷移關聯的工作，例如在雲端中提供核心操作功能。手冊可以採用指令碼、自動化執行手冊或操作現代化環境所需的程序或步驟摘要的形式。

PLC

請參閱可程式設計邏輯控制器。

PLM

請參閱產品生命週期管理。

政策

可定義許可的物件（請參閱身分型政策）、指定存取條件（請參閱資源型政策），或定義組織中所有帳戶的最大許可 AWS Organizations（請參閱服務控制政策）。

混合持久性

根據資料存取模式和其他需求，獨立選擇微服務的資料儲存技術。如果您的微服務具有相同的資料儲存技術，則其可能會遇到實作挑戰或效能不佳。如果微服務使用最適合其需求的資料儲存，則可以更輕鬆地實作並達到更好的效能和可擴展性。

組合評定

探索、分析應用程式組合並排定其優先順序以規劃遷移的程序。如需詳細資訊，請參閱評估遷移準備程度。

述詞

傳回 `true`或 的查詢條件`false`，通常位於 `WHERE` 子句中。

述詞下推

一種資料庫查詢最佳化技術，可在傳輸前篩選查詢中的資料。這可減少必須從關聯式資料庫擷取和處理的資料量，並改善查詢效能。

預防性控制

旨在防止事件發生的安全控制。這些控制是第一道防線，可協助防止對網路的未經授權存取或不必要變更。如需詳細資訊，請參閱在 AWS上實作安全控制中的預防性控制。

委託人

中可執行動作和存取資源 AWS 的實體。此實體通常是 AWS 帳戶、IAM 角色或使用者的根使用者。如需詳細資訊，請參閱 IAM 文件中角色術語和概念中的主體。

依設計的隱私權

透過整個開發程序將隱私權納入考量的系統工程方法。

私有託管區域

一種容器，它包含有關您希望 Amazon Route 53 如何回應一個或多個 VPC 內的域及其子域之 DNS 查詢的資訊。如需詳細資訊，請參閱 Route 53 文件中的使用私有託管區域。

主動控制

旨在防止部署不合規資源的安全控制。這些控制項會在佈建資源之前對其進行掃描。如果資源不符合控制項，則不會佈建。如需詳細資訊，請參閱 AWS Control Tower 文件中的控制項參考指南，並參閱實作安全控制項中的主動控制項。 AWS

產品生命週期管理 (PLM)

管理產品整個生命週期的資料和程序，從設計、開發和啟動，到成長和成熟，再到拒絕和移除。

生產環境

請參閱 環境。

可程式設計邏輯控制器 (PLC)

在製造中，高度可靠、可調整的電腦，可監控機器並自動化製造程序。

提示鏈結

使用一個 LLM 提示的輸出做為下一個提示的輸入，以產生更好的回應。此技術用於將複雜任務分解為子任務，或反覆精簡或展開初步回應。它有助於提高模型回應的準確性和相關性，並允許更精細、個人化的結果。

擬匿名化

將資料集中的個人識別符取代為預留位置值的程序。假名化有助於保護個人隱私權。假名化資料仍被視為個人資料。

發佈/訂閱 (pub/sub)

一種模式，可啟用微服務之間的非同步通訊，以提高可擴展性和回應能力。例如，在微服務型 MES 中，微服務可以將事件訊息發佈到其他微服務可訂閱的頻道。系統可以新增新的微服務，而無需變更發佈服務。

Q

查詢計劃

一系列步驟，如指示，用於存取 SQL 關聯式資料庫系統中的資料。

查詢計劃迴歸

在資料庫服務優化工具選擇的計畫比對資料庫環境進行指定的變更之前的計畫不太理想時。這可能因為對統計資料、限制條件、環境設定、查詢參數繫結的變更以及資料庫引擎的更新所導致。

R

RACI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RAG

請參閱擷取增強生成。

勒索軟體

一種惡意軟體，旨在阻止對計算機系統或資料的存取，直到付款為止。

RASCI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RCAC

請參閱資料列和資料欄存取控制。

僅供讀取複本

用於唯讀用途的資料庫複本。您可以將查詢路由至僅供讀取複本以減少主資料庫的負載。

重新架構師

請參閱 7 個 R。

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這會決定最後一個復原點與服務中斷之間可接受的資料遺失。

復原時間目標 (RTO)

服務中斷與服務還原之間的可接受延遲上限。

重構

請參閱 7 個 R。

區域

地理區域中的 AWS 資源集合。每個 AWS 區域 都獨立於其他 ，以提供容錯能力、穩定性和彈性。如需詳細資訊，請參閱指定 AWS 區域 您的帳戶可以使用哪些 。

迴歸

預測數值的 ML 技術。例如，為了解決「這房子會賣什麼價格？」的問題 ML 模型可以使用線性迴歸模型，根據已知的房屋事實 (例如，平方英尺) 來預測房屋的銷售價格。

重新託管

請參閱 7 個 R。

版本

在部署程序中，它是將變更提升至生產環境的動作。

重新定位

請參閱 7 個 R。

Replatform

請參閱 7 個 R。

回購

請參閱 7 個 R。

彈性

應用程式抵禦中斷或從中斷中復原的能力。在 中規劃彈性時，高可用性和災難復原是常見的考量 AWS 雲端。如需詳細資訊，請參閱AWS 雲端 彈性。

資源型政策

附接至資源的政策，例如 Amazon S3 儲存貯體、端點或加密金鑰。這種類型的政策會指定允許存取哪些主體、支援的動作以及必須滿足的任何其他條件。

負責者、當責者、事先諮詢者和事後告知者 (RACI) 矩陣

矩陣，定義所有參與遷移活動和雲端操作之各方的角色和責任。矩陣名稱衍生自矩陣中定義的責任類型：負責人 (R)、責任 (A)、已諮詢 (C) 和知情 (I)。支援 (S) 類型為選用。如果您包含支援，則矩陣稱為 RASCI 矩陣，如果您排除它，則稱為 RACI 矩陣。

回應性控制

一種安全控制，旨在驅動不良事件或偏離安全基準的補救措施。如需詳細資訊，請參閱在 AWS上實作安全控制中的回應性控制。

保留

請參閱 7 個 R。

淘汰

請參閱 7 Rs。

檢索增強生成 (RAG)

一種生成式 AI 技術，其中 LLM 會在產生回應之前參考訓練資料來源以外的授權資料來源。例如，RAG 模型可能會對組織的知識庫或自訂資料執行語意搜尋。如需詳細資訊，請參閱什麼是 RAG。

輪換

定期更新秘密的程序，讓攻擊者更難存取登入資料。

資料列和資料欄存取控制 (RCAC)

使用已定義存取規則的基本、彈性 SQL 表達式。RCAC 包含資料列許可和資料欄遮罩。

RPO

請參閱復原點目標。

RTO

請參閱復原時間目標。

執行手冊

執行特定任務所需的一組手動或自動程序。這些通常是為了簡化重複性操作或錯誤率較高的程序而建置。

S

SAML 2.0

許多身分提供者 (IdP) 使用的開放標準。此功能會啟用聯合單一登入 (SSO)，讓使用者可以登入 AWS 管理主控台 或呼叫 AWS API 操作，而不必為您組織中的每個人在 IAM 中建立使用者。如需有關以 SAML 2.0 為基礎的聯合詳細資訊，請參閱 IAM 文件中的關於以 SAML 2.0 為基礎的聯合。

斯卡達

請參閱監督控制和資料擷取。

SCP

請參閱服務控制政策。

秘密

您以加密形式存放的 AWS Secrets Manager機密或限制資訊，例如密碼或使用者登入資料。它由秘密值及其中繼資料組成。秘密值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱 Secrets Manager 秘密中的內容？ 在 Secrets Manager 文件中。

依設計的安全性

透過整個開發程序將安全性納入考量的系統工程方法。

安全控制

一種技術或管理防護機制，它可預防、偵測或降低威脅行為者利用安全漏洞的能力。安全控制有四種主要類型：預防性、偵測性、回應性和主動性。

安全強化

減少受攻擊面以使其更能抵抗攻擊的過程。這可能包括一些動作，例如移除不再需要的資源、實作授予最低權限的安全最佳實務、或停用組態檔案中不必要的功能。

安全資訊與事件管理 (SIEM) 系統

結合安全資訊管理 (SIM) 和安全事件管理 (SEM) 系統的工具與服務。SIEM 系統會收集、監控和分析來自伺服器、網路、裝置和其他來源的資料，以偵測威脅和安全漏洞，並產生提醒。

安全回應自動化

預先定義和程式設計的動作，旨在自動回應或修復安全事件。這些自動化可做為偵測或回應式安全控制，協助您實作 AWS 安全最佳實務。自動化回應動作的範例包括修改 VPC 安全群組、修補 Amazon EC2 執行個體或輪換登入資料。

伺服器端加密

由 AWS 服務 接收資料的 在其目的地加密資料。

服務控制政策 (SCP)

為 AWS Organizations中的組織的所有帳戶提供集中控制許可的政策。SCP 會定義防護機制或設定管理員可委派給使用者或角色的動作限制。您可以使用 SCP 作為允許清單或拒絕清單，以指定允許或禁止哪些服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的服務控制政策。

服務端點

的進入點 URL AWS 服務。您可以使用端點，透過程式設計方式連接至目標服務。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS 服務 端點。

服務水準協議 (SLA)

一份協議，闡明 IT 團隊承諾向客戶提供的服務，例如服務正常執行時間和效能。

服務層級指標 (SLI)

服務效能層面的測量，例如其錯誤率、可用性或輸送量。

服務層級目標 (SLO)

代表服務運作狀態的目標指標，由服務層級指標測量。

共同責任模式

描述您與 共同 AWS 承擔雲端安全與合規責任的模型。 AWS 負責雲端的安全，而 負責雲端的安全。如需詳細資訊，請參閱共同責任模式。

陰影 AI

在組織內受管管道之外建置或使用的未授權 AI 應用程式。

SIEM

請參閱安全資訊和事件管理系統。

單一故障點 (SPOF)

應用程式的單一關鍵元件故障，可能會中斷系統。

SLA

請參閱服務層級協議。

SLI

請參閱服務層級指標。

SLO

請參閱服務層級目標。

先拆分後播種模型

擴展和加速現代化專案的模式。定義新功能和產品版本時，核心團隊會進行拆分以建立新的產品團隊。這有助於擴展組織的能力和服務，提高開發人員生產力，並支援快速創新。如需詳細資訊，請參閱 中的階段式應用程式現代化方法 AWS 雲端。

SPOF

請參閱單一故障點。

星狀結構描述

使用一個大型事實資料表來存放交易或測量資料的資料庫組織結構，並使用一或多個較小的維度資料表來存放資料屬性。此結構旨在用於資料倉儲或商業智慧用途。

Strangler Fig 模式

一種現代化單一系統的方法，它會逐步重寫和取代系統功能，直到舊式系統停止使用為止。此模式源自無花果藤，它長成一棵馴化樹並最終戰勝且取代了其宿主。該模式由 Martin Fowler 引入，作為重寫單一系統時管理風險的方式。如需有關如何套用此模式的範例，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

子網

您 VPC 中的 IP 地址範圍。子網必須位於單一可用區域。

監控控制和資料擷取 (SCADA)

在製造中，使用硬體和軟體來監控實體資產和生產操作的系統。

對稱加密

使用相同金鑰來加密及解密資料的加密演算法。

合成測試

以模擬使用者互動的方式測試系統，以偵測潛在問題或監控效能。您可以使用 Amazon CloudWatch Synthetics 來建立這些測試。

系統提示

一種向 LLM 提供內容、指示或指導方針以指示其行為的技術。系統提示有助於設定內容，並建立與使用者互動的規則。

T

標籤

做為中繼資料的鍵值對，用於組織您的 AWS 資源。標籤可協助您管理、識別、組織、搜尋及篩選資源。如需詳細資訊，請參閱標記您的 AWS 資源。

目標變數

您嘗試在受監督的 ML 中預測的值。這也被稱為結果變數。例如，在製造設定中，目標變數可能是產品瑕疵。

任務清單

用於透過執行手冊追蹤進度的工具。任務清單包含執行手冊的概觀以及要完成的一般任務清單。對於每個一般任務，它包括所需的預估時間量、擁有者和進度。

測試環境

請參閱 環境。

訓練

為 ML 模型提供資料以供學習。訓練資料必須包含正確答案。學習演算法會在訓練資料中尋找將輸入資料屬性映射至目標的模式 (您想要預測的答案)。它會輸出擷取這些模式的 ML 模型。可以使用 ML 模型，來預測您不知道的目標新資料。

tool

代理程式可以叫用以在外部系統中執行操作的函數或 API。

傳輸閘道

可以用於互連 VPC 和內部部署網路的網路傳輸中樞。如需詳細資訊，請參閱 AWS Transit Gateway 文件中的什麼是傳輸閘道。

主幹型工作流程

這是一種方法，開發人員可在功能分支中本地建置和測試功能，然後將這些變更合併到主要分支中。然後，主要分支會依序建置到開發環境、生產前環境和生產環境中。

受信任的存取權

將許可授予您指定的服務，以代表您在組織中 AWS Organizations 及其帳戶中執行任務。受信任的服務會在需要該角色時，在每個帳戶中建立服務連結角色，以便為您執行管理工作。如需詳細資訊，請參閱 文件中的 AWS Organizations 搭配使用 AWS Organizations 與其他 AWS 服務。

調校

變更訓練程序的各個層面，以提高 ML 模型的準確性。例如，可以透過產生標籤集、新增標籤、然後在不同的設定下多次重複這些步驟來訓練 ML 模型，以優化模型。

雙比薩團隊

兩個比薩就能吃飽的小型 DevOps 團隊。雙披薩團隊規模可確保軟體開發中的最佳協作。

U

不確定性

這是一個概念，指的是不精確、不完整或未知的資訊，其可能會破壞預測性 ML 模型的可靠性。有兩種類型的不確定性：認知不確定性是由有限的、不完整的資料引起的，而隨機不確定性是由資料中固有的噪聲和隨機性引起的。

未區分的任務

也稱為繁重工作，這是建立和操作應用程式的必要工作，但不為最終使用者提供直接價值或提供競爭優勢。未區分任務的範例包括採購、維護和容量規劃。

較高的環境

請參閱 環境。

V

清空

一種資料庫維護操作，涉及增量更新後的清理工作，以回收儲存並提升效能。

版本控制

追蹤變更的程序和工具，例如儲存庫中原始程式碼的變更。

VPC 對等互連

兩個 VPC 之間的連線，可讓您使用私有 IP 地址路由流量。如需詳細資訊，請參閱 Amazon VPC 文件中的什麼是 VPC 對等互連。

漏洞

危害系統安全性的軟體或硬體瑕疵。

W

暖快取

包含經常存取的目前相關資料的緩衝快取。資料庫執行個體可以從緩衝快取讀取，這比從主記憶體或磁碟讀取更快。

暖資料

不常存取的資料。查詢這類資料時，通常可接受中等緩慢的查詢。

視窗函數

SQL 函數，會對與目前記錄在某種程度上相關的資料列群組執行計算。視窗函數適用於處理任務，例如根據目前資料列的相對位置計算移動平均值或存取資料列的值。

工作負載

提供商業價值的資源和程式碼集合，例如面向客戶的應用程式或後端流程。

工作串流

遷移專案中負責一組特定任務的功能群組。每個工作串流都是獨立的，但支援專案中的其他工作串流。例如，組合工作串流負責排定應用程式、波次規劃和收集遷移中繼資料的優先順序。組合工作串流將這些資產交付至遷移工作串流，然後再遷移伺服器和應用程式。

WORM

請參閱寫入一次，讀取許多。

WQF

請參閱AWS 工作負載資格架構。

寫入一次，讀取許多 (WORM)

儲存模型，可一次性寫入資料，並防止刪除或修改資料。授權使用者可以視需要多次讀取資料，但無法變更資料。此資料儲存基礎設施被視為不可變。

Z

零時差入侵

利用零時差漏洞的攻擊，通常是惡意軟體。

零時差漏洞

生產系統中未緩解的瑕疵或漏洞。威脅行為者可以使用這種類型的漏洞來攻擊系統。開發人員經常因為攻擊而意識到漏洞。

零鏡頭提示

提供 LLM 執行任務的指示，但沒有可協助引導任務的範例 (快照)。LLM 必須使用其預先訓練的知識來處理任務。零鏡頭提示的有效性取決於任務的複雜性和提示的品質。另請參閱少量擷取提示。

殭屍應用程式

CPU 和記憶體平均使用率低於 5% 的應用程式。在遷移專案中，通常會淘汰這些應用程式。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。