9. Aislamiento de Tenant

Nuestro recorrido por el aislamiento incluirá un análisis más profundo del papel que pueden desempeñar los microservicios y el código de la aplicación en el cumplimiento de las políticas de aislamiento.

El objetivo general es garantizar que el aislamiento de tenant sea una prioridad en las soluciones SaaS, revisando el conjunto de enfoques disponibles para construir un modelo de aislamiento multi-tenant robusto y no invasivo.

Conceptos fundamentales

Es posible elegir un modelo de despliegue para habilitar una experiencia de aislamiento específica, pero la realización y el cumplimiento efectivo de ese aislamiento se logran a través de un mecanismo completamente independiente que examina cada intento de acceso a un recurso de tenant y previene cualquier violación de aislamiento entre tenants.

Simplemente separar las bases de datos no garantizó que los datos estuvieran realmente aislados. Esto nos devuelve a la frontera entre despliegue y aislamiento.

Aquí es donde debemos introducir un mecanismo de aislamiento de tenant independiente que haga cumplir el aislamiento, independientemente de cómo estén desplegados los recursos. La idea es agregar un elemento que se sitúe entre el código y los recursos a los que ese código accede.

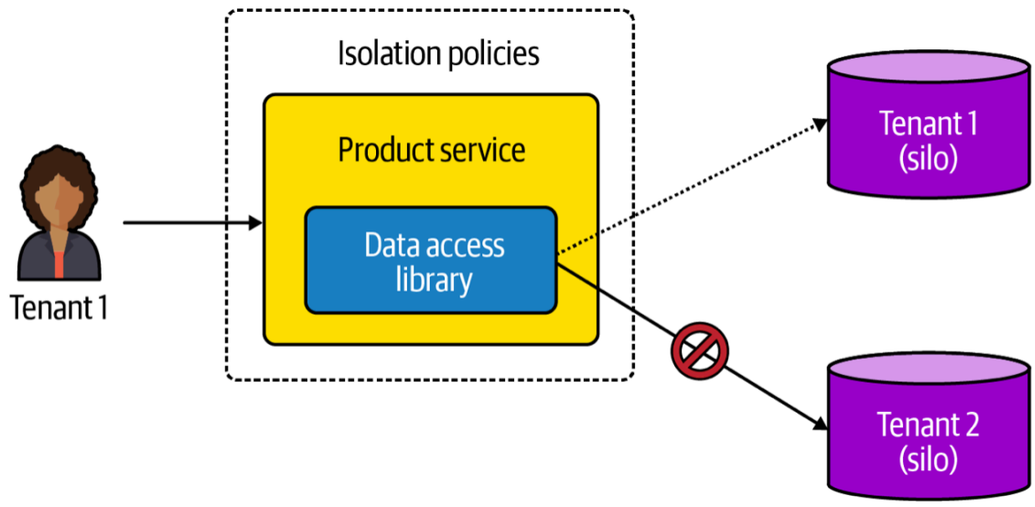

Figure 9-2. Siloed databases with tenant isolation

El único cambio es el envoltorio alrededor del servicio de Productos, que representa una visión conceptual del aislamiento de tenant. **Así, sin importar lo que haga un desarrollador en su código, el mecanismo de aislamiento impedirá cualquier intento de acceder a los recursos de otro tenant.**

Los desarrolladores suelen ver su código como "confiable" y asumir que sus equipos nunca escribirían código que viole un límite de tenant. **No es una buena política presumir que el código no romperá las reglas de aislamiento.**

La industria está plagada de ejemplos de soluciones que de alguna manera expusieron los datos de un tenant a otro. Incluso un solo caso de acceso entre tenants podría representar un retroceso significativo para un negocio SaaS.

También debe quedar claro que el aislamiento se crea como una parte muy intencional de la arquitectura. Se implementa explícitamente como un elemento central del diseño, preparado para capturar cualquier intento, intencional o no, de cruzar los límites entre tenants.

Categorización de los modelos de aislamiento

La Figura 9-4 muestra los tres tipos principales de aislamiento más habituales (sin descartar que puedan existir otros).

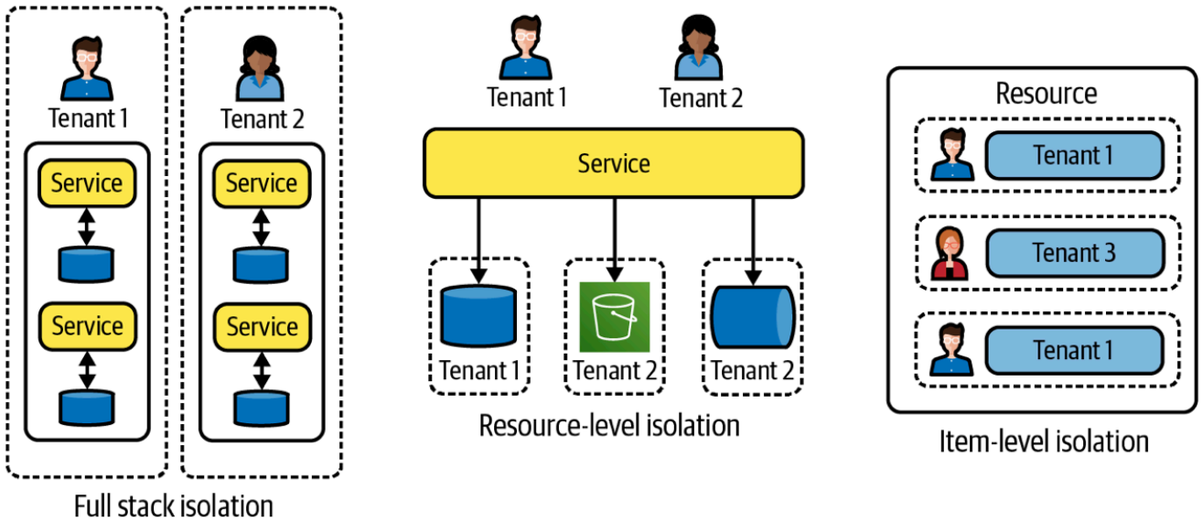

Figure 9-4. Categorizing isolation models

- **El primer tipo, a la izquierda, es el que denominamos aislamiento de pila completa (full stack isolation).** Se corresponde directamente con los entornos multi-tenant que utilizan un modelo de despliegue de pila completa, donde cada tenant recibe una pila de recursos dedicada. Aquí, aislar recursos es generalmente un proceso bastante directo, con acceso a un conjunto bien definido de mecanismos que permiten aislar los recursos de cada tenant.
- **Al avanzar hacia el centro, encontramos el aislamiento a nivel de recurso (resource-level isolation).** En este caso, existe una capa de cómputo compartida para el servicio que consume recursos de múltiples tenants. En este modelo, la unidad de aislamiento es un "recurso" completo. La idea fundamental es que el límite del aislamiento es un recurso entero, cuya definición puede variar según los servicios del entorno. **Con este tipo de aislamiento, es probable que aún exista algún elemento de aislamiento que controle el acceso al recurso.**
- **Finalmente, a la derecha se encuentra el modelo de aislamiento a nivel de elemento (item-level isolation).** En este modelo se accede al interior de un recurso donde conviven elementos de distintos tenants. El ejemplo más sencillo es una base de datos (el recurso) con una tabla compartida que mezcla datos de múltiples tenants (modelo pooled).

El aislamiento a nivel de elemento es, con diferencia, el más desafiante de todos los esquemas de aislamiento. Una vez que se entra dentro de un recurso, la lista de elementos de aislamiento disponibles se reduce considerablemente.

Algunas tecnologías ofrecen herramientas para hacer cumplir el aislamiento a este nivel y otras no. En AWS, por ejemplo, hay casos en los que el elemento de gestión de identidad y acceso (IAM) puede utilizarse para implementar el aislamiento a nivel de elemento.

Aislamiento aplicado por la aplicación

En un mundo ideal, la tecnología utilizada tendría correspondencias claras con elementos de seguridad que podrían usarse para hacer cumplir la estrategia de aislamiento. Por ejemplo, la mayoría de los entornos cloud ofrecen alguna noción integrada de controles IAM para configurar las políticas que los desarrolladores SaaS usan para controlar el acceso a los distintos recursos del entorno.

El desafío, sin embargo, es que estas herramientas no siempre ofrecen el nivel de control necesario para expresar las políticas de aislamiento. Hay varios factores que dan forma al perfil IAM de cada tecnología o servicio. Los servicios nativos construidos desde cero por un proveedor cloud, por ejemplo, suelen tener controles de aislamiento más granulares que los servicios basados en tecnologías preexistentes.

En algún momento, es probable que se presente un escenario en el que el modelo multi-tenant preferido para un recurso no admita el nivel de control de aislamiento requerido. Aquí es donde puede ser necesario introducir mecanismos de aislamiento propios, aplicados por la aplicación, para prevenir el acceso entre tenants.

En general, será necesario evaluar los distintos marcos de políticas y control de acceso, bibliotecas o herramientas que permitan incorporar una capa propia de controles para los casos en que los mecanismos integrados no cubran las necesidades.

Aquí es donde pueden aparecer mecanismos como el control de acceso basado en atributos (ABAC) o Open Policy Agent (OPA) como parte del modelo de aislamiento.

La conclusión clave es que la solución debe aislar todos los recursos, incluso si es necesario construir las herramientas de aislamiento uno mismo.

RBAC, autorización y aislamiento

Los equipos suelen utilizar el control de acceso basado en roles (RBAC) y elementos de autorización para delimitar y controlar el acceso a la funcionalidad dentro de sus aplicaciones. En algunos casos, **se observa que los equipos utilizan esas mismas herramientas RBAC para implementar sus políticas de aislamiento de tenant.**

El RBAC también puede emplearse en otros contextos, como la autorización de acceso a la infraestructura.

La única tarea del aislamiento es garantizar que, para el tenant actual, el acceso a sus recursos quede restringido a ese tenant. Cualquier otra restricción que deba aplicarse en función de un rol u otro elemento de la aplicación se aborda fuera del modelo de

aislamiento.

El punto principal es mantener una separación clara entre las estrategias utilizadas para aislar los recursos de tenant y las estrategias utilizadas para controlar el acceso a funcionalidades y funciones específicas de la aplicación.

La mentalidad del aislamiento y la del control de acceso a la aplicación son conceptos muy distintos.

Aislamiento en la aplicación frente a aislamiento en la infraestructura

Cuando se trabaja con equipos centrados en la seguridad y se menciona el aislamiento de tenant, estos suelen orientarse hacia una noción de aislamiento más centrada en la infraestructura, lo cual tiene sentido.

Sin embargo, en el aislamiento multi-tenant, los límites y la naturaleza del aislamiento son realmente un elemento definido por la aplicación. **Al construir una aplicación SaaS, corresponde al desarrollador definir dónde se ubican esos límites e introducir los mecanismos a nivel de aplicación que garantizarán la protección de los recursos de tenant.**

Algunos de estos límites pueden coincidir con los límites de infraestructura. Sin embargo, en un entorno multi-tenant, la aplicación también puede definir su propio conjunto de límites de aislamiento sobre esa infraestructura.

El aislamiento de tenant es algo que se define y se hace cumplir mediante el diseño y la arquitectura de la aplicación. También es importante señalar que estos límites definidos por la aplicación pueden y deben apoyarse en el código y las bibliotecas de la aplicación como parte de su estrategia de aislamiento de tenant.

Las capas del modelo de aislamiento

Con los conceptos fundamentales de aislamiento ya revisados, podemos centrar la atención en elementos de aislamiento más concretos.

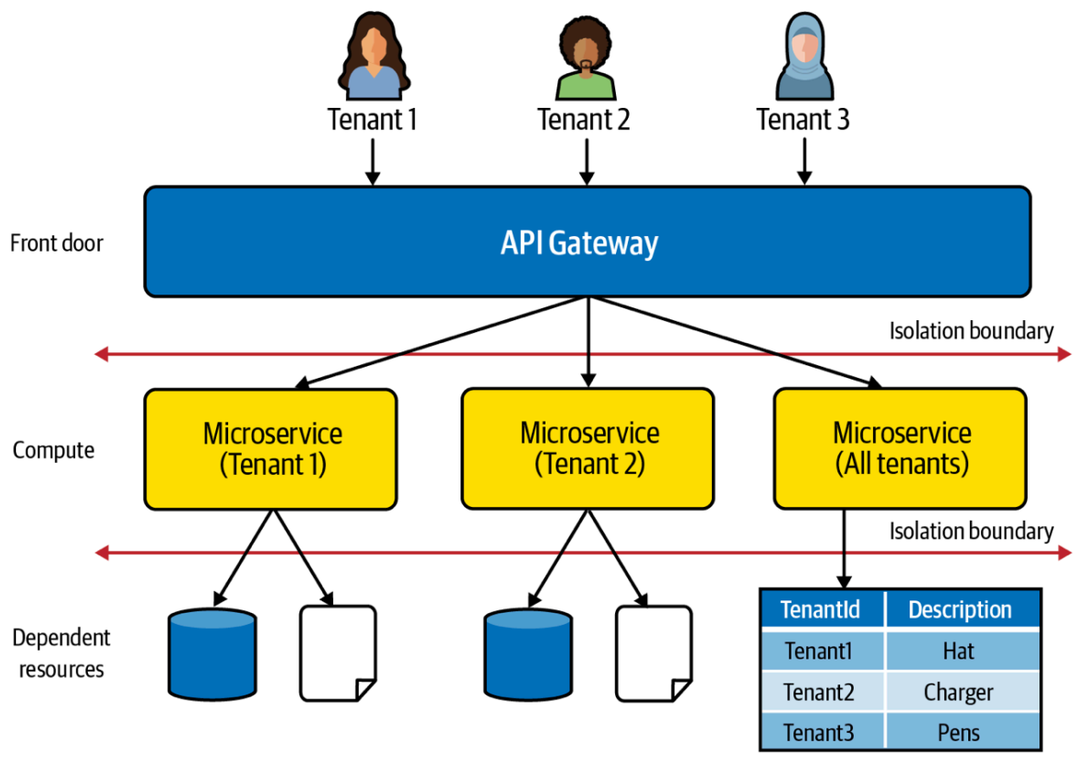

Figure 9-5. A layered view of isolation

- En la cima se encuentra la entrada o "puerta principal" de los servicios de la aplicación que deben aislarse. Cada solicitud que llega a través de esta API incluirá el contexto de tenant que el sistema utilizará para aplicar el aislamiento de tenant. **Esto puede impedir que los tenants invoquen solicitudes a recursos que no son válidos para un contexto de tenant determinado.** En esta capa no se previene el acceso a los datos de un tenant, pero sí se aísla el acceso a sus recursos de cómputo.
- Una vez en la capa de cómputo, se aplica el siguiente nivel de aislamiento cuando el microservicio intenta acceder a otros recursos dependientes (bases de datos, colas, sistemas de archivos, etc.). Aquí se necesitarán políticas de aislamiento para garantizar que cada microservicio solo pueda acceder a los recursos dedicados a ese tenant.
  - Para los dos recursos siloed, estas políticas serán relativamente directas.
  - Sin embargo, el microservicio pooled deberá implementar aislamiento a nivel de elemento para controlar el acceso a las filas de cada tenant dentro de su tabla compartida. Las políticas de aislamiento deben garantizar que cada solicitud del microservicio pooled quede restringida únicamente a los elementos asociados al contexto de tenant actual.

En muchos aspectos, esto se nutre del concepto tradicional de seguridad en cada capa. Aquí, sin embargo, se amplía ese concepto añadiendo protecciones de aislamiento a medida que se transita entre las distintas capas del entorno.

Aislamiento en tiempo de despliegue frente a aislamiento en tiempo de ejecución

Además de estructurar el modelo de aislamiento en capas, es necesario considerar cuándo se aplicará el aislamiento al entorno.

- En algunos escenarios, las políticas de aislamiento pueden aplicarse en el momento en que los recursos se despliegan y configuran.
- En otros casos, el aislamiento deberá configurarse y aplicarse en tiempo de ejecución.

Implementaciones siloed (tiempo de despliegue)

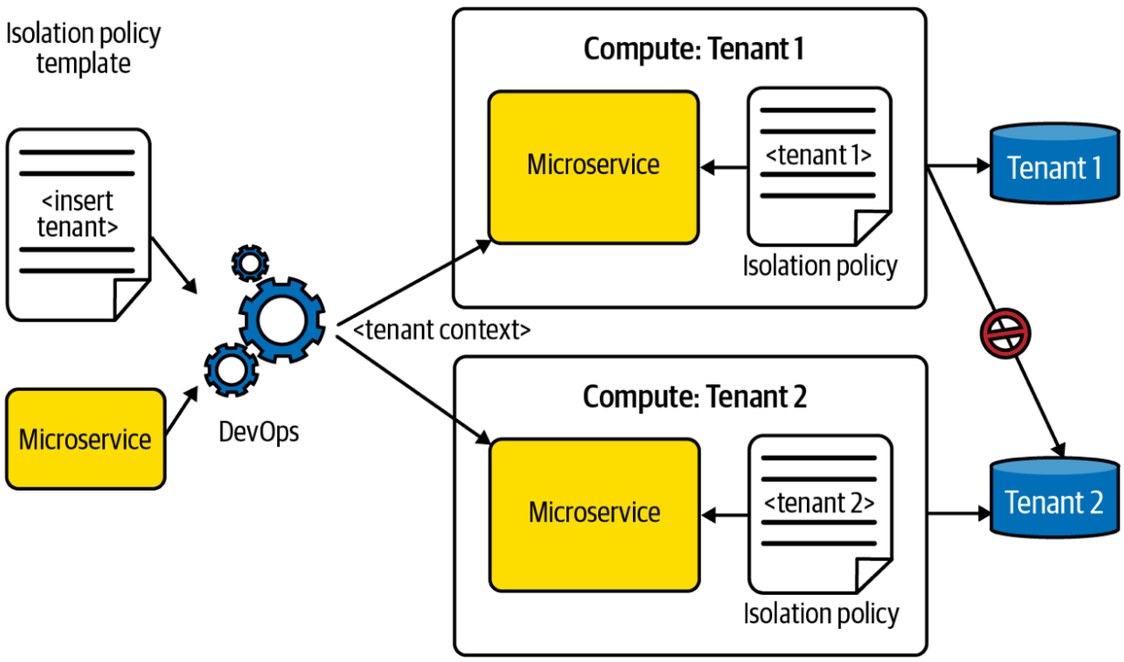

Figure 9-6. Deployment-time isolation model

Cada uno de estos microservicios está representado en el lado derecho del diagrama. Ambos microservicios tienen bases de datos siloed asociadas que también se despliegan en un modelo siloed.

Puede observarse que los microservicios de la solución también operan en modo siloed. Esto significa que, durante toda la vida útil de estos microservicios desplegados, estarán vinculados a un tenant específico.

Este vínculo crea la oportunidad de simplificar el esquema de aislamiento para estos microservicios, permitiendo asociar una política con alcance de tenant al cómputo de cada microservicio, lo que impide que ese microservicio acceda a recursos pertenecientes a otros tenants.

Cuando el proceso DevOps aprovisiona los recursos de cómputo para cada microservicio, puede insertar el contexto de tenant en la plantilla de política de aislamiento y asociar esa política a la infraestructura de cómputo.

Este modelo de tiempo de despliegue tiene un gran poder. Dado que la política se asocia en el momento del despliegue, el aislamiento no depende de ningún código del microservicio para cumplir con la estrategia de aislamiento.

Implementaciones pooled (tiempo de ejecución)

Aquí es donde se utiliza típicamente el modelo de aislamiento en tiempo de ejecución. Con el aislamiento en tiempo de ejecución, se comienzan a explorar estrategias que dependen de la cooperación del código de la aplicación u otros elementos para adquirir y aplicar dinámicamente las políticas de aislamiento.

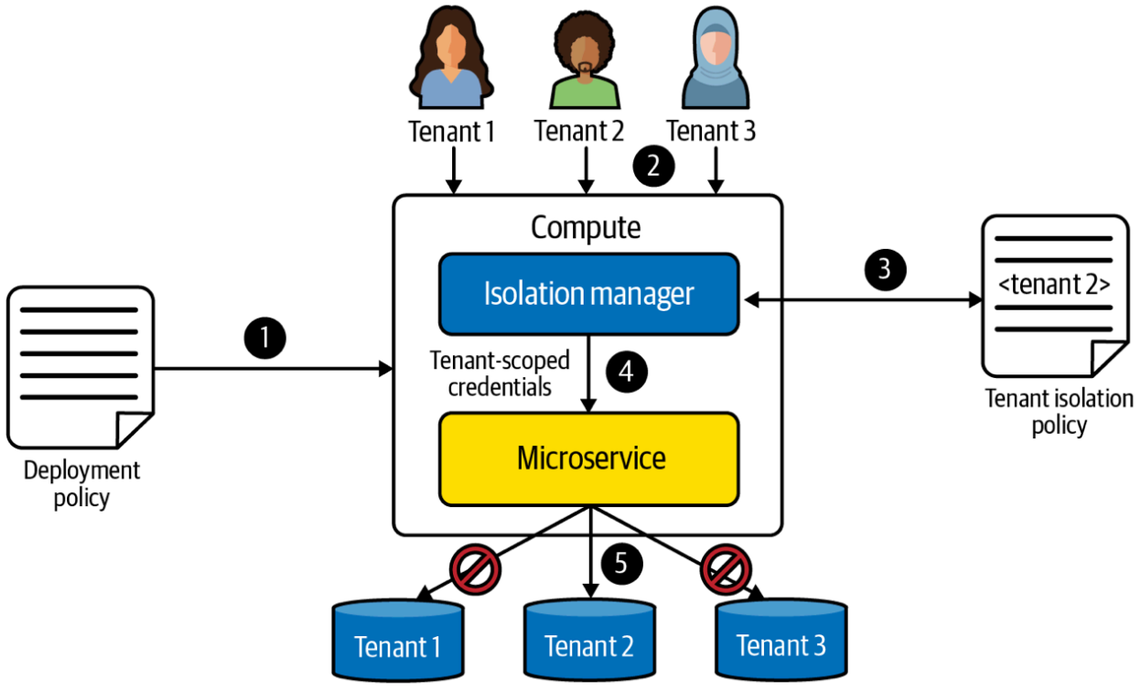

Figure 9-7. Runtime isolation model

1. Dado que este cómputo debe poder procesar solicitudes de todos los tenants, debe desplegarse con una política que los cubra a todos (paso 1).
2. El flujo comienza en la parte superior, donde los tenants acceden al microservicio enviando su contexto de tenant (paso 2). En algún punto del entorno de cómputo del microservicio, **será necesario usar el contexto de tenant para completar una política y obtener las credenciales con alcance de tenant que se utilizarán para acceder a los recursos posteriores.**
3. En esta visión conceptual se incluye un gestor de aislamiento que realiza este proceso (paso 3). En la práctica, la forma en que se obtienen estas credenciales con alcance de tenant puede variar significativamente según la tecnología y el stack de lenguaje. Se puede envolver el microservicio, usar un sidecar, usar aspects: la lista de opciones es bastante larga.
4. Sin embargo, lo clave es que algún mecanismo obtenga estas credenciales con alcance de tenant a partir de la política completada dinámicamente y las proporcione al microservicio (paso 4). Una vez que el microservicio dispone de estas credenciales, las utilizará para acceder a sus recursos de tenant asociados. **Incluso si el código intentara insertar el identificador de otro tenant en su solicitud de acceso a la base de datos, esa solicitud no devolvería los datos de otro tenant.**

Puede verse que este enfoque depende en gran medida del código y las bibliotecas de la solución. Esto deja margen para que los microservicios tomen decisiones que eludan los mecanismos de aislamiento. Sin embargo, **cuanto más se trabaje en inyectar este contexto de aislamiento en el código fuera de la vista de los desarrolladores, mayores serán las posibilidades de hacer cumplir la estrategia general de aislamiento.**

También vale la pena señalar que existen estrategias alternativas que trasladan la resolución en tiempo de ejecución fuera del alcance del microservicio.

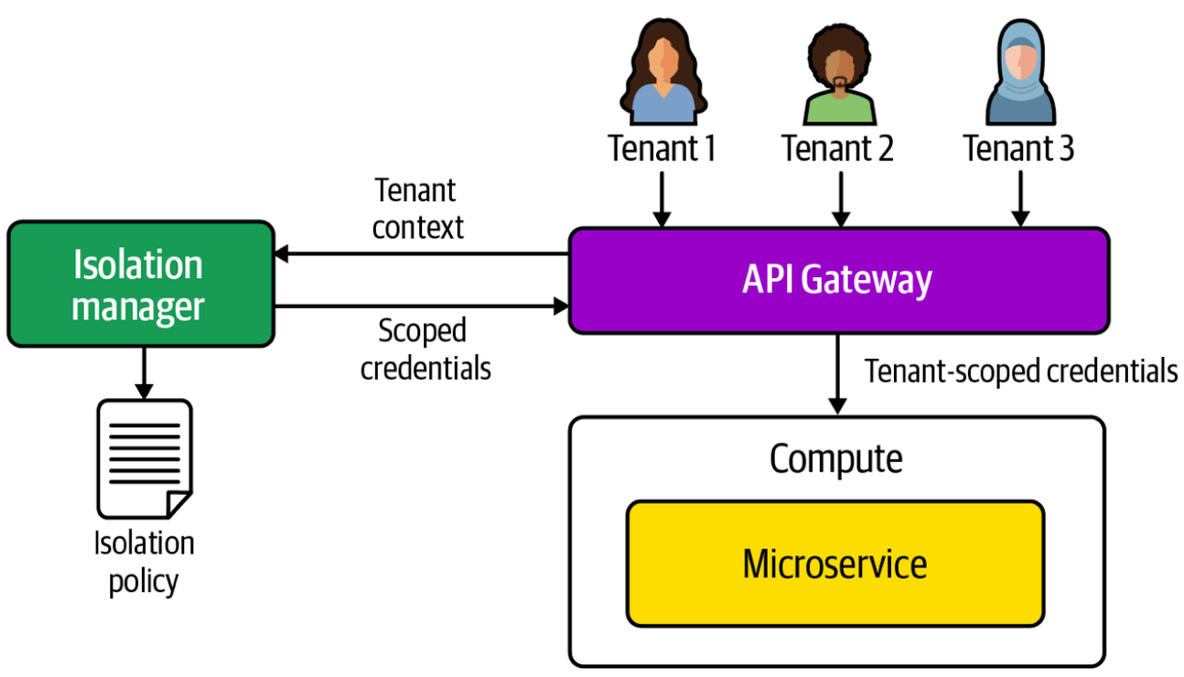

Figure 9-8. Injecting runtime scoped credentials

Este gateway preprocesará las solicitudes, enviando el contexto de tenant al código del gestor de aislamiento, que usará dicho contexto junto con una política de aislamiento para obtener las credenciales con alcance de tenant.

La resolución de las credenciales con alcance se ha trasladado al API Gateway. Una vez que el gateway obtiene las credenciales, las pasa al microservicio donde se utilizan para delimitar el acceso a los recursos de tenant.

Este modelo tiene algunas ventajas: traslada la resolución de las credenciales fuera de la vista de los desarrolladores de microservicios y crea oportunidades más naturales para cachear las credenciales con alcance y gestionar posibles problemas de latencia.

La desventaja de este modelo, sin embargo, es que las políticas de aislamiento se desplazan fuera del microservicio. En general, la definición del alcance de las políticas se considera parte del microservicio y está estrechamente vinculada a su implementación.

Aislamiento mediante intercepción

Uno de los objetivos del aislamiento en tiempo de ejecución es eliminar a los desarrolladores de la ecuación de aislamiento en la mayor medida posible. **Cuanto más se dependa de los desarrolladores como parte del esquema de aislamiento, más engorroso y frágil se vuelve.**

La Figura 9-9 ofrece una visión conceptual de dos estrategias generales de intercepción.

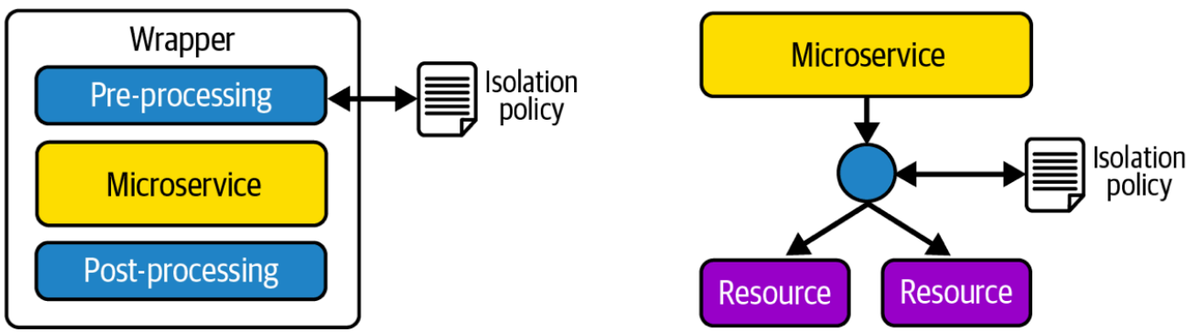

Figure 9-9. Isolation through interception strategies

A la izquierda se ve un enfoque de intercepción basado en el lenguaje o el framework. Las herramientas aplicadas aquí tienden a insertarse en la ruta de ejecución del código, interceptando y preprocesando las solicitudes fuera de la vista del código del microservicio. **Los aspects, el middleware y las bibliotecas de envoltura son algunas de las opciones posibles.**

El otro modelo, mostrado a la derecha, adopta un enfoque ligeramente diferente. Aquí se coloca un mecanismo entre el recurso y el microservicio que intercepta cada intento de acceso a un recurso de tenant (como un proxy). Este mecanismo resuelve el contexto de tenant y lo aplica a medida que se accede a cada recurso; aquí es donde se utilizan conceptos como los sidecars para implementar este esquema de intercepción.

Consideraciones de escalabilidad

El aislamiento aplicado en tiempo de ejecución, si bien es efectivo, puede introducir problemas de escalabilidad en el entorno. Si hay un servicio pooled que procesa un alto volumen de solicitudes y cada solicitud debe obtener credenciales con alcance de tenant, existe el riesgo de que esto introduzca un nivel de latencia inaceptable en el entorno, lo que puede afectar la escalabilidad general del sistema.

Existen estrategias de caché que pueden incorporarse para mantener las credenciales con alcance. Con este enfoque, algunos sistemas emplean una configuración de time-to-live (TTL) para gestionar el ciclo de vida de estas credenciales.

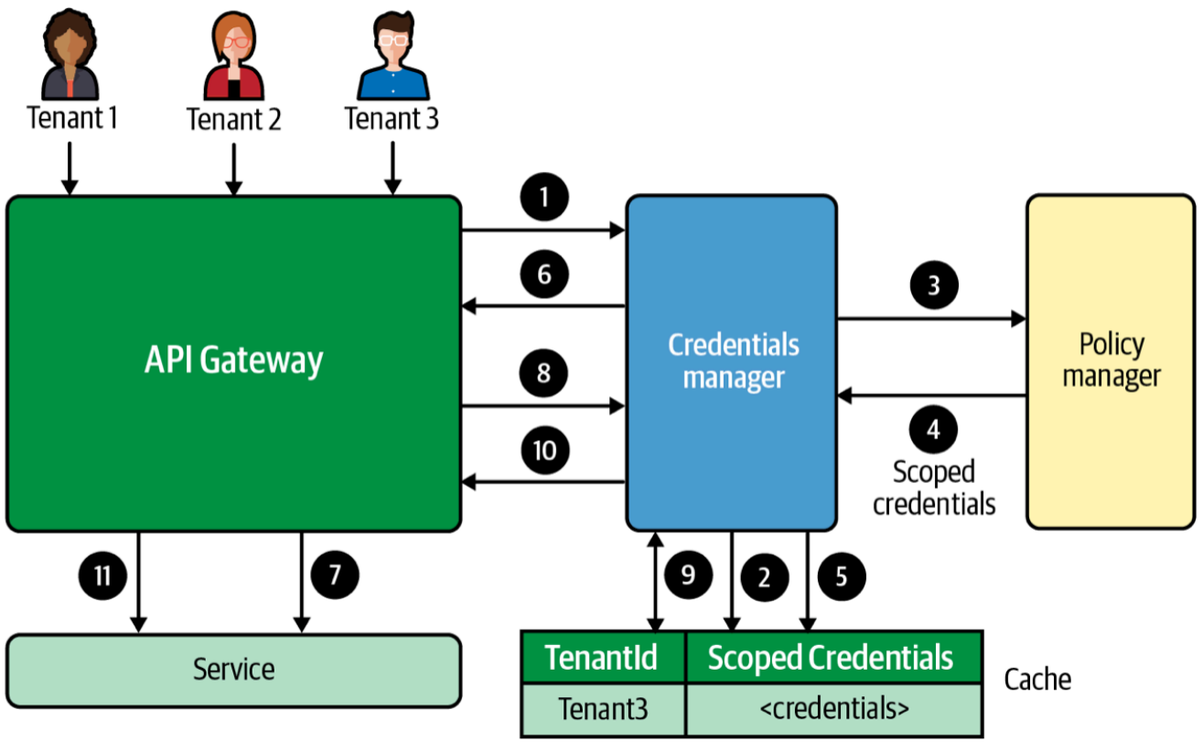

Figure 9-10. Caching isolation credentials

En la parte superior izquierda hay una serie de tenants que acceden a los servicios de la aplicación SaaS a través de un API Gateway. En este escenario particular, supongamos que el Tenant 3 está realizando su primera solicitud a un servicio.

1. Cuando llega esa solicitud, el gateway invoca a un gestor de credenciales para obtener las credenciales con alcance del Tenant 3 (paso 1).
2. El gestor de credenciales intentará buscar la credencial en la caché (paso 2).
3. En este caso, supongamos que el Tenant 3 no se encuentra. El gestor de credenciales obtiene entonces las credenciales con alcance de tenant del gestor de políticas (pasos 3 y 4).
4. Las credenciales con alcance se almacenan en la caché (paso 5),
5. se devuelven al API Gateway (paso 6),
6. y se inyectan en las interacciones posteriores con los servicios (paso 7).
7. En este momento, también se les asigna un TTL al almacenarse en la caché.
8. En la siguiente llamada del Tenant 3, el sistema llama al gestor de credenciales (paso 8) y encuentra las credenciales almacenadas en la caché (paso 9).
9. Las credenciales cacheadas se devuelven al API Gateway (paso 10) y se inyectan en las interacciones posteriores con los servicios (paso 11).

En este modelo, el gestor de credenciales y sus componentes auxiliares no son servicios separados; todos se ejecutan dentro del mismo proceso.

Algunos equipos intentan centralizar todos estos mecanismos de aislamiento en servicios independientes. En general, la sobrecarga de cruzar el límite hacia otro servicio puede añadir otra capa de latencia que genera problemas de rendimiento adicionales.

Estos conceptos pueden trasladarse a bibliotecas y otros elementos compartidos, pero se recomienda mantener el manejo de las solicitudes de gestión de aislamiento dentro del mismo proceso.

Finalmente, también es necesario considerar los límites de escalabilidad de los servicios que se estén utilizando. **Si, por ejemplo, se utilizan políticas IAM para implementar el aislamiento en AWS, habrá que verificar si la cantidad de políticas necesarias podría superar los límites del servicio.** Por eso se observa un uso intensivo de plantillas de políticas en los ejemplos, donde una única política puede utilizarse en múltiples contextos de tenant.

Ejemplos del mundo real

Las secciones siguientes ofrecen una muestra de estrategias de implementación que abarcan los distintos tipos de aislamiento revisados en las secciones anteriores, conectando los conceptos con soluciones reales.

Aislamiento full stack

En la figura se representa una serie de entornos de tipo full stack silo. La idea fundamental es que cada tenant dispone de recursos de infraestructura completamente dedicados. Este modelo, como era de esperar, tiene una correspondencia natural con las herramientas y tecnologías que ya se utilizan para establecer límites entre recursos.

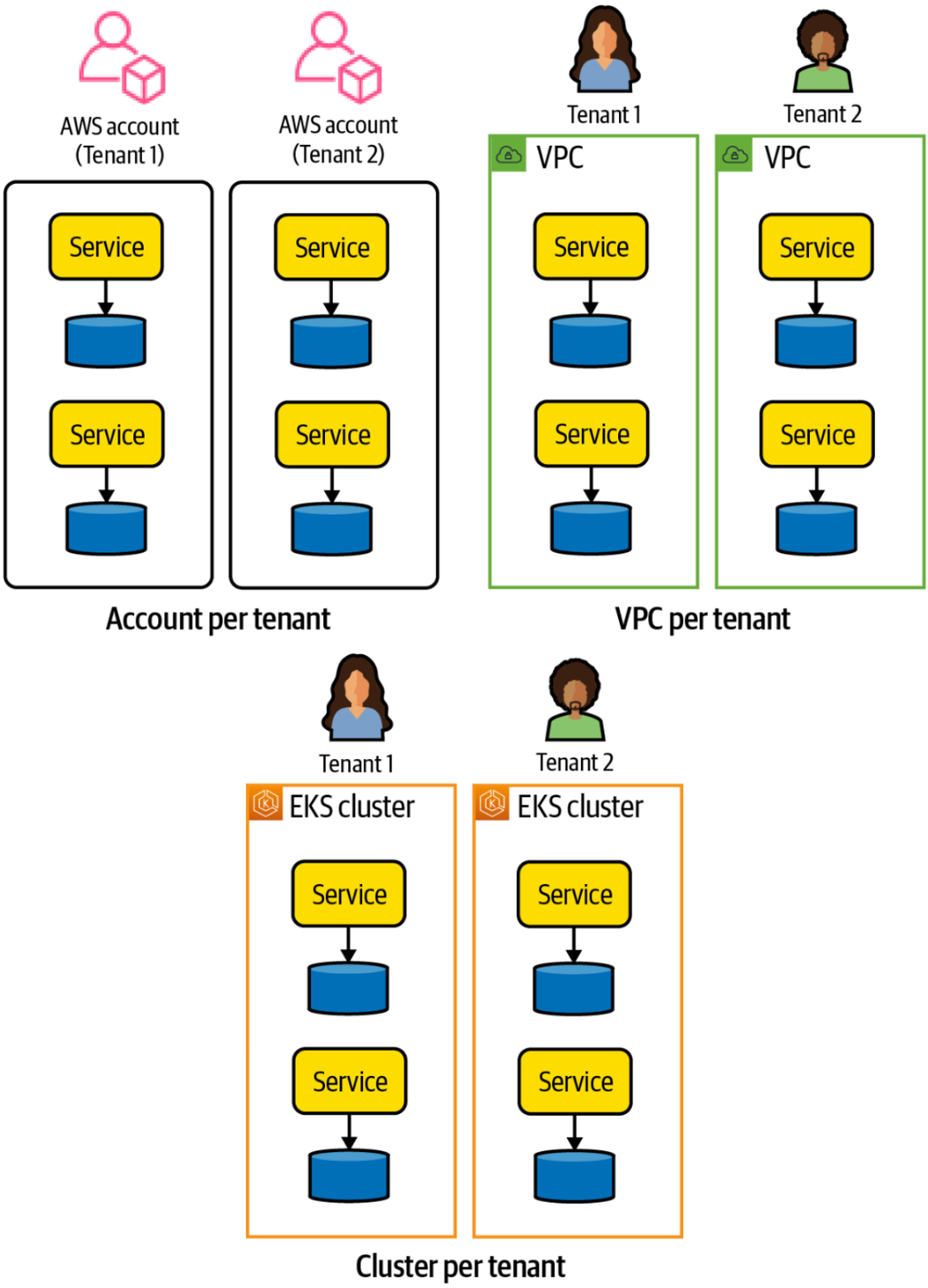

Figure 9-11. Coarse-grained isolation constructs

- En la parte superior izquierda se muestra un modelo de aislamiento de una cuenta por tenant, donde cada cuenta del proveedor cloud (AWS) se utiliza para definir el límite de aislamiento. Dado que las cuentas impiden el acceso entre cuentas por defecto, este representa uno de los elementos más simples para lograr el aislamiento en un entorno full stack silo.
- En la parte superior derecha se muestra cómo un elemento de red puede utilizarse para aislar los silos de tenant. En este caso, se utiliza una Amazon Virtual Private Cloud (VPC) para aislar los recursos de tenant. La VPC, como la mayoría de los elementos de red, ofrece numerosas opciones integradas para configurar el flujo de tráfico hacia y desde la red.
- Finalmente, en la parte inferior del diagrama se muestra un ejemplo que utiliza Amazon Elastic Kubernetes Service (EKS) para implementar un modelo de aislamiento de pila completa. Sin embargo, en este escenario se ha optado por el extremo de tener un clúster separado para cada tenant, lo que implica aprovisionar un entorno EKS completamente independiente para cada tenant y confiar en los límites naturales del clúster para hacer cumplir el aislamiento.

Aislamiento a nivel de recurso

El aislamiento a nivel de recurso tiende a contar con buenos elementos que se corresponden bien con los mecanismos de aislamiento. Cuando se trabaja con recursos dedicados, el aislamiento solo necesita encontrar un mecanismo que pueda controlar el acceso a ese recurso.

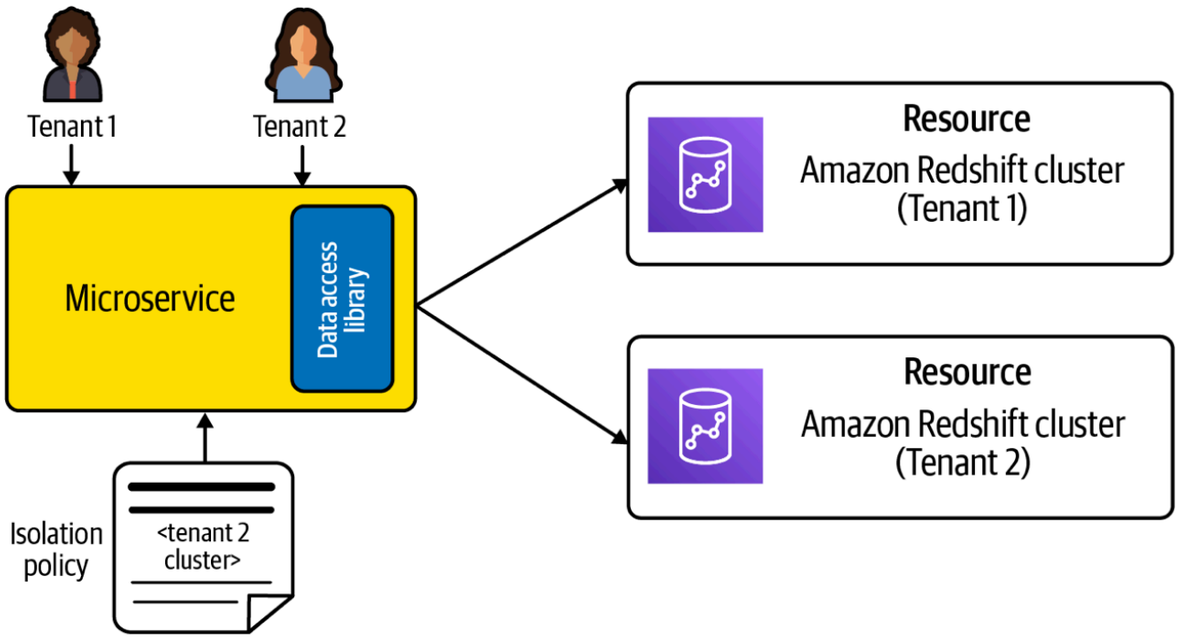

Figure 9-12. Resource-level isolation with Amazon Redshift

Dentro de estos clústeres, los datos se gestionan y acceden de forma independiente para cada tenant. Aunque se utiliza un clúster como unidad de aislamiento, este sigue correspondiendo conceptualmente a un recurso. Los límites de un recurso en este modelo pueden adoptar múltiples formas: una base de datos, una cola, un clúster de análisis, son algunos de los distintos elementos que se clasificarían como recurso en este escenario.

El microservicio utiliza una biblioteca de acceso a datos (DAL) para gestionar sus interacciones con los clústeres de Redshift asociados. La DAL, como parte del procesamiento de cada solicitud, utiliza el contexto de tenant actual y su política de aislamiento para obtener credenciales con alcance de tenant que limitarán el acceso a un clúster específico de tenant.

Como contraste, puede verse cómo esto cambia cuando la solución dispone de recursos de cómputo siloed (Figura 9-13).

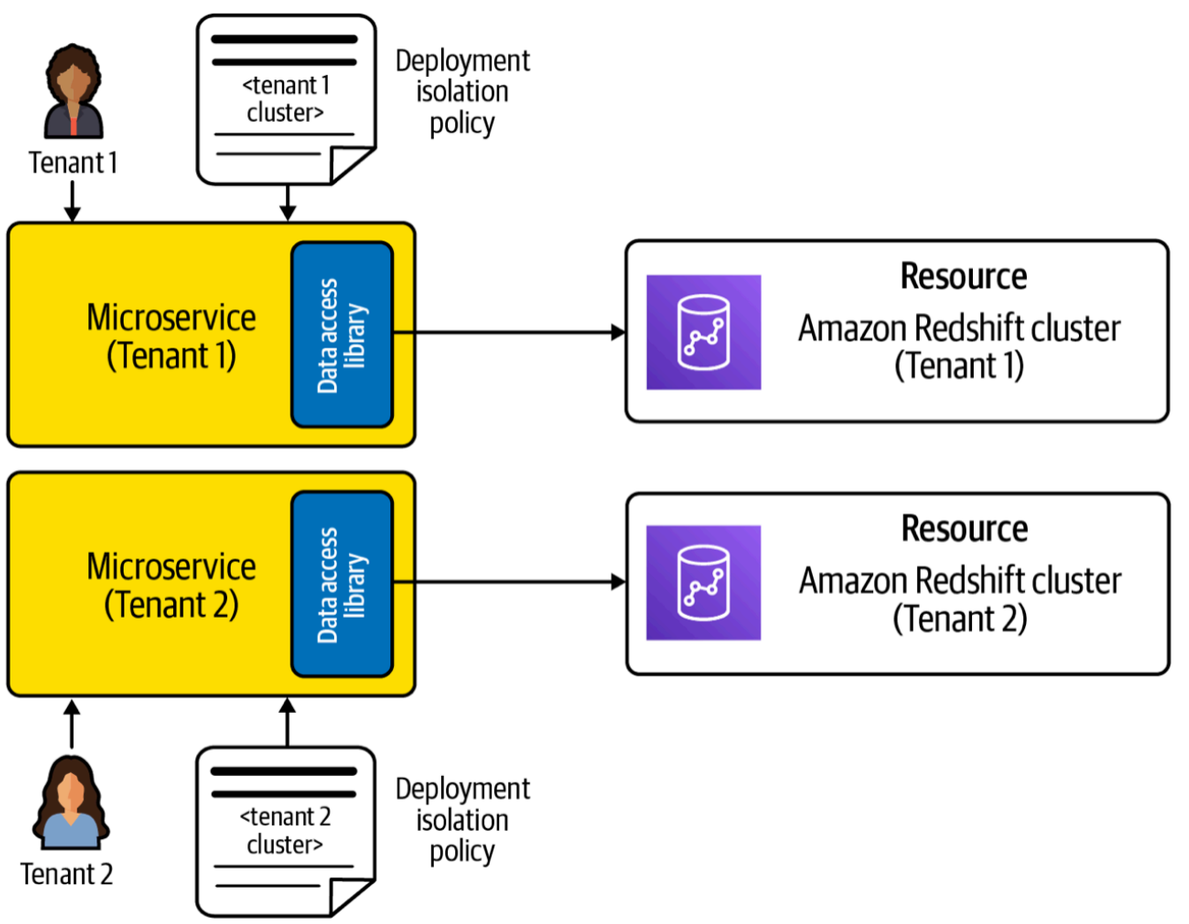

Figure 9-13. A deployment-time view of resource-level isolation

Lo que cambia ahora es que se dispone de microservicios siloed separados, dedicados a cada tenant. La naturaleza siloed de este modelo permite modificar el enfoque de aislamiento. En lugar de obtener y resolver el alcance del aislamiento en tiempo de ejecución, esta solución puede asociar una estrategia de aislamiento específica durante el aprovisionamiento y la configuración del cómputo que ejecuta los microservicios.

Ahora, la DAL no necesita absorber ninguna sobrecarga ni complejidad adicional para aplicar las políticas de aislamiento, ya que estas ya están aplicadas por la política asociada al cómputo durante el despliegue.

Aislamiento a nivel de elemento

El aislamiento a nivel de elemento representa uno de los modelos de aislamiento más desafiantes, principalmente porque requiere un nivel de granularidad que muchas tecnologías no suelen admitir.

Al mismo tiempo, en entornos multi-tenant donde hay una fuerte tendencia hacia la infraestructura compartida, se presentarán numerosos escenarios en los que será necesario idear estrategias que permitan implementar políticas capaces de aislar elementos individuales que se ejecutan en infraestructura compartida.

La buena noticia es que existen escenarios, especialmente en la nube, donde algunos servicios incluyen soporte integrado para el aislamiento a nivel de elemento.

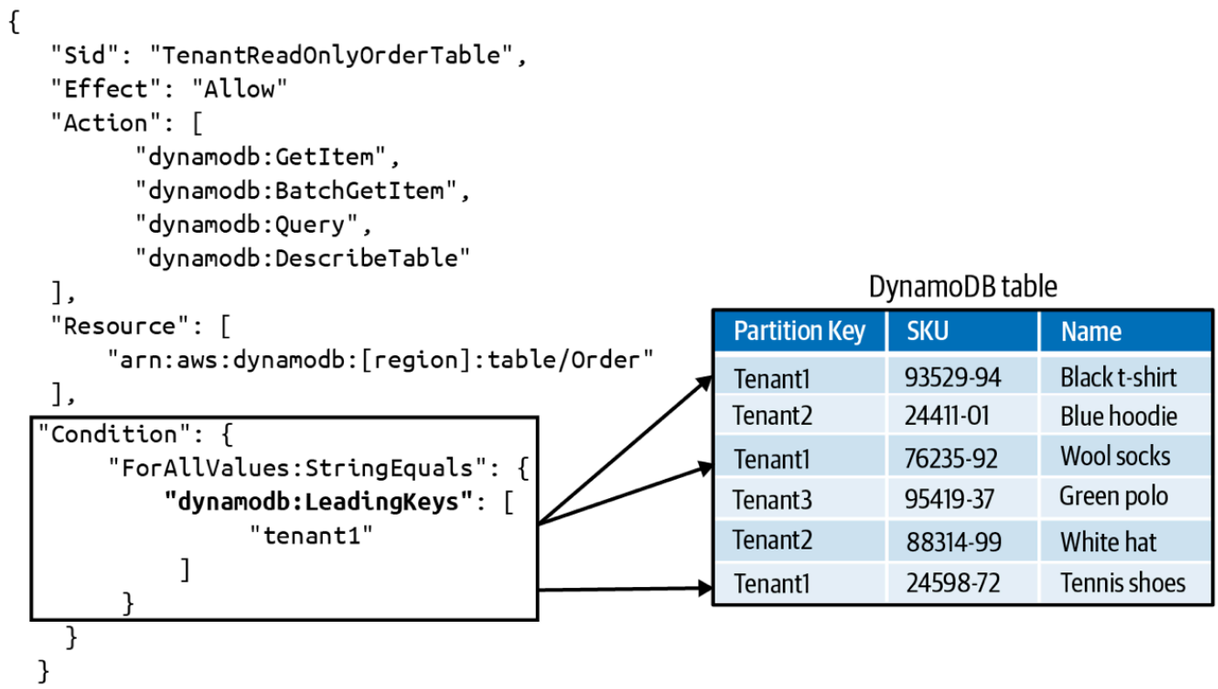

Figure 9-14. Item-level isolation with DynamoDB

La política que hace cumplir este aislamiento se muestra a la izquierda. Utiliza el mecanismo IAM de AWS para definir el alcance del acceso a una tabla de DynamoDB.

Este es el esquema básico que debe aplicarse en cualquier aislamiento a nivel de elemento. Sin embargo, es posible que no siempre se cuente con algo tan directo como esta política IAM para implementar el aislamiento.

Las herramientas utilizadas pueden introducir algunas complicaciones adicionales en la estrategia de aislamiento a nivel de elemento. El objetivo principal es evitar un modelo que dependa por completo de asumir que el código simplemente no cruzará los límites entre tenants.

Gestión de políticas de aislamiento

Dónde se almacenan las políticas, quién las posee y gestiona, cuándo y cómo se despliegan, cómo se versionan: todas estas son preguntas que deben considerarse como parte de la creación y construcción de una experiencia de despliegue para la infraestructura y los microservicios de la aplicación.

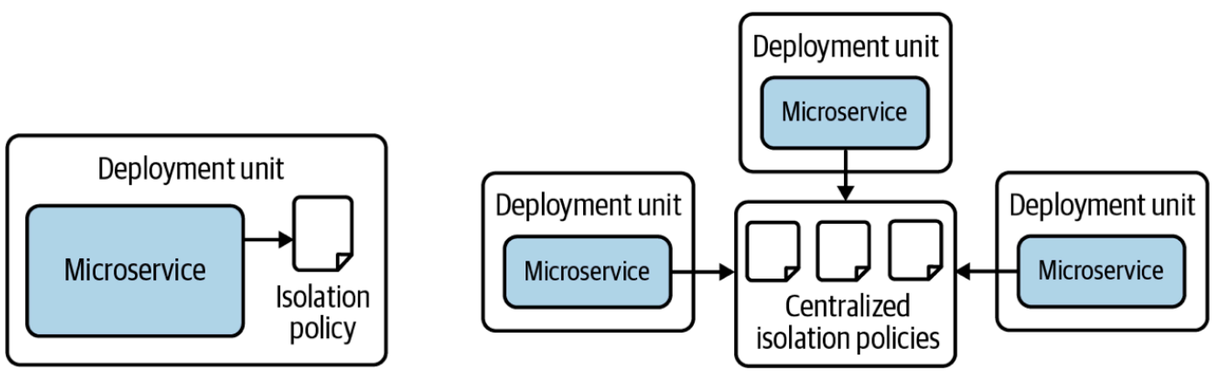

Figure 9-15. The deployment, managing, and versioning of your isolation policies

El modelo de la izquierda de la Figura 9-15 representa un enfoque más centrado en el desarrollo de microservicios individuales. Aquí, los desarrolladores consideran sus políticas de aislamiento como una extensión de su microservicio, aplicando estrategias

directamente vinculadas a la implementación de su solución.

Este enfoque es el que más se comparte. La tendencia es considerar una política de aislamiento como uno de los entregables vinculados al microservicio. Como propietario de un microservicio, el equipo también es responsable del mantenimiento y la definición de las políticas necesarias para garantizar que el microservicio cumpla con sus requisitos de aislamiento. **Esto implica versionar, gestionar e integrar la configuración de políticas dentro del repositorio del microservicio.**

La otra corriente de pensamiento (mostrada en el lado derecho del diagrama) prefiere un modelo donde todas estas políticas se gestionan y versionan a través de un mecanismo centralizado. **Con este enfoque, los equipos seguirían siendo propietarios de la actualización de sus políticas, pero todas se versionarían y gestionarían fuera del alcance de cualquier microservicio.**

Se desplegarían en una ubicación centralizada a la que harían referencia todos los microservicios. Esto proporciona un lugar común y un modelo de despliegue independiente para las políticas, facilitando su gestión y ofreciendo un esquema uniforme para referenciar los servicios desde cada parte de la arquitectura.

Al pensar en la operacionalización de la gestión y el despliegue del aislamiento, también es necesario considerar cómo se introducirán pruebas que validen que el aislamiento realmente funciona.

Esto, sorprendentemente, ha sido uno de los aspectos más desafiantes del aislamiento. Resulta que no existen mecanismos naturales que permitan simular que un tenant cruza un límite.

Si bien realizar pruebas para estas condiciones puede ser difícil, no hacerlas no parece una opción viable. Contar con pruebas de aislamiento permite detectar potencialmente fugas no intencionadas en las políticas o en la implementación del aislamiento.

La pregunta real es: ¿qué se puede hacer para simular un evento entre tenants dentro del entorno? Esto debe ocurrir en algún punto dentro de los internos de los servicios de la aplicación. **Hay un momento en el que el código accederá a un recurso de tenant basándose en un contexto de tenant proporcionado, y es aquí donde será necesario inyectar de alguna manera un contexto de tenant inválido.**

Esto probablemente implicará crear rutas o casos especiales en el código para permitir esta inyección de un tenant inválido. También es un área donde podría considerarse el uso de estrategias clásicas de ingeniería del caos para generar excepciones de aislamiento.

Esto puede parecer un poco artificial, y es posible encontrar formas más creativas de simular estas condiciones entre tenants. Sin embargo, aunque resulte algo forzado, sigue siendo esencial para validar el modelo de aislamiento del entorno.