6. Autenticación y Enrutamiento de Tenant

Ahora es momento de comenzar a pensar en cómo un tenant utiliza estos elementos (y otros) para entrar por la puerta principal de nuestro entorno multi-tenant.

Entrada por la puerta principal

Existen múltiples opciones disponibles cuando se piensa en cómo los tenants podrían acceder a tu sistema.

- Podrías querer que el dominio de tu sistema, por ejemplo, incluya el nombre de un tenant y depender de este dominio como parte de tu estrategia de mapeo y enrutamiento de tenant.
- O podrías simplemente permitir que los tenants tengan sus propios dominios únicos basados en marca u otras consideraciones. En cualquier caso, el dominio típicamente termina jugando algún rol en la identificación del tenant que está accediendo a tu sistema.
- Algunas soluciones SaaS, sin embargo, no tienen dependencia del dominio, utilizando un único dominio para todos los tenants. **En este enfoque, necesitarás que el flujo de autenticación de tu entorno inyecte el contexto de tenant en el sistema.**

Acceso mediante un dominio de tenant

Una de las formas comunes en que los tenants entran por la puerta principal de tu aplicación es a través de un dominio. Más específicamente, los tenants pueden entrar con un dominio que incluye información utilizada para identificar a un tenant.

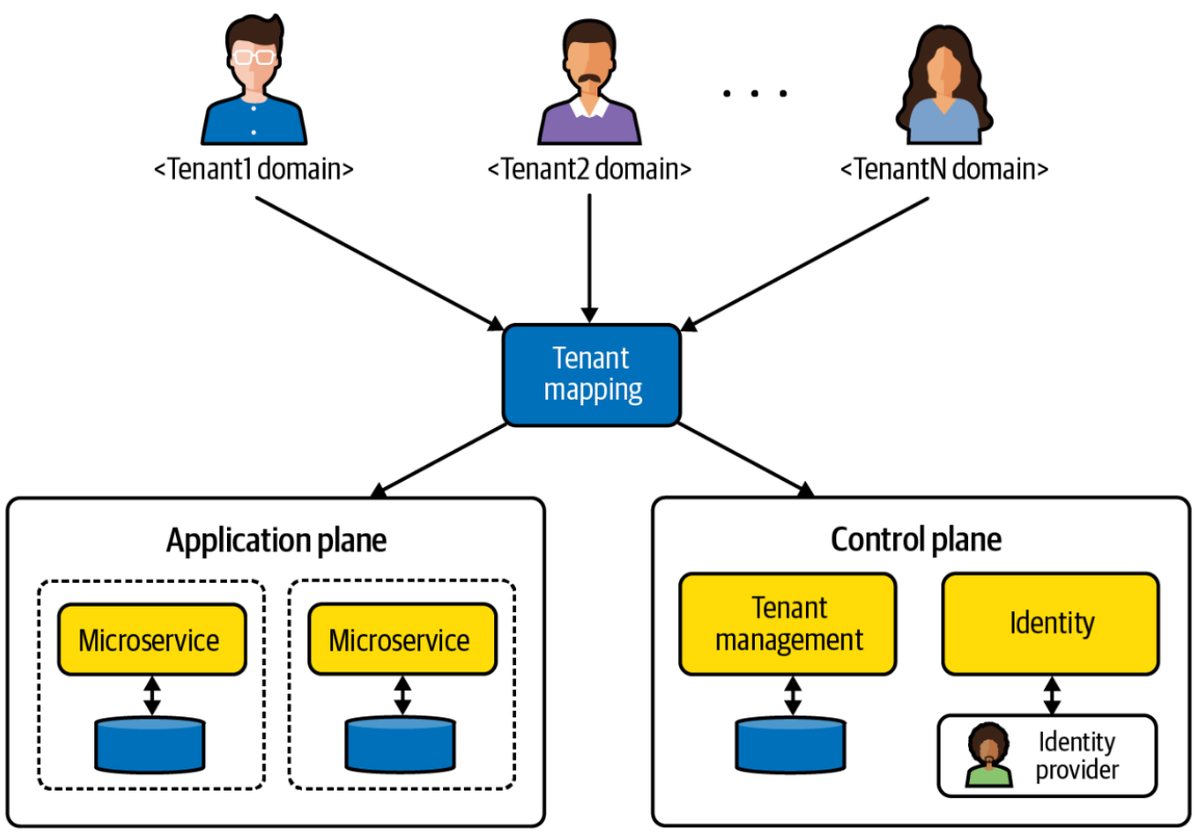

Figure 6-1. A domain-driven access model

Una vez que el dominio está configurado, la URL se comparte con el tenant como su punto de entrada al servicio SaaS.

Con este enfoque, todas las solicitudes entrantes de tenants desde estos diferentes dominios se enrutan a través de lo que he etiquetado como mapeo de tenant. Este servicio de mapeo generalmente es necesario para extraer el contexto de tenant entrante del dominio y usar ese contexto para resolver cualquier mapeo hacia recursos dedicados del tenant.

Hay al menos dos lugares comunes donde se puede aplicar este mapeo.

- El primero es la autenticación. Cuando el sistema está autenticando a un usuario de tenant, puede necesitar mapear una solicitud de autenticación entrante a su elemento de identidad correspondiente. Esto aplica principalmente a entornos de tenant que no tienen un único proveedor de identidad compartido. En estos casos, un proveedor de identidad separado puede soportar tu autenticación, o puede haber elementos de identidad distintos dentro de un único proveedor de identidad (grupos, pools de usuarios, etc.) que están vinculados a tenants individuales para proporcionar características o experiencias específicas.
- La otra área donde se aplica el mapeo de tenant está relacionada con el enrutamiento de solicitudes de aplicación. En los casos donde tu arquitectura está usando uno o más recursos de tenant siloed, las solicitudes entrantes del tenant necesitarán ser enrutadas a cada uno de estos recursos específicos.

También vale la pena señalar que el nombre de dominio (o subdominio) representa un nombre amigable y públicamente visible que se ha expuesto como la URL de entrada de un tenant. Este nombre puede cambiar durante la vida del tenant y, como tal, se

mantiene completamente separado de la noción interna de un identificador de tenant.

El modelo de subdominio por tenant

En muchos entornos SaaS, existe el deseo de utilizar dominios para identificar tenants sin requerir que cada tenant tenga un dominio completamente único. Sus clientes, en este ejemplo, tampoco ven valor en tener sus propios dominios.

El resultado final produciría dominios como `tenant1.abc-software.com` y `tenant2.abc-software.com`. Es probable que hayas visto este patrón implementado en soluciones SaaS existentes que consumes hoy en día.

Este enfoque a menudo resulta muy atractivo para los proveedores SaaS, permitiéndoles proporcionar un punto de acceso único para cada tenant sin absorber la complejidad y la sobrecarga de crear un dominio único para cada tenant.

El modelo de dominio personalizado por tenant

Este enfoque se utiliza frecuentemente en escenarios donde un tenant presenta una experiencia de marca a sus clientes. De hecho, en estos escenarios, los usuarios del tenant pueden no tener conocimiento del sistema SaaS subyacente que están utilizando.

En este escenario, los tenants tendrían sus propios dominios únicos que se utilizan para acceder a tu entorno multi-tenant.

Vale la pena señalar que este mismo modelo podría usarse como parte de una estrategia de marca blanca donde se permite a los tenants aplicar su propia marca. Por ejemplo, imagina una plataforma SaaS de comercio electrónico donde cada tienda en la plataforma usa un dominio personalizado y aplica su propia marca a toda la experiencia.

Onboarding con dominios de tenant

Ahora, a medida que se crea cada nuevo tenant, necesitarás configurar estos dominios y crear cualquier mapeo que sea necesario para dar soporte a la entrada de un tenant en tu entorno.

El modelo de subdominio por tenant tiene un impacto más ligero en tu flujo de onboarding. Aquí, tu esfuerzo se centra principalmente en configurar las diversas opciones de configuración DNS que son parte de tu entorno.

En este ejemplo, al dominio general se le asignó el nombre alterno `app.sassco.com`, donde "saasco" representa el nombre de dominio de marca de nuestra empresa SaaS ficticia. Ahora, cuando un nuevo tenant se incorpora al sistema, debes agregar una nueva fila a esta tabla de CDN. En este caso, he agregado un nuevo tenant que se configura como `tenant1.sassco.com`.

Amazon CloudFront

Origin	Domain Name	Alternate Names
App-Bucket	https://abc123.cloudfront.net	app.saasco.com
		tenant1.saasco.com

Además de configurar el CDN, también debemos configurar el enrutamiento DNS para este nuevo subdominio de tenant. La tabla en la parte inferior proporciona un ejemplo de cómo podrías configurar tu servicio DNS.

Amazon Route 53

Record Name	Type	Route To
app.saasco.com	A	https://abc123.cloudfront.net
tenant1.saasco.com	A	https://abc123.cloudfront.net

La tabla muestra dos entradas. La primera fila contiene el conjunto base de valores que se pueblan cuando configuras tu entorno por primera vez. La segunda fila se puebla cuando incorporas a tenant1. Esto crea el registro A que apunta el subdominio `tenant1.saasco.com` al dominio general de tu entorno SaaS.

Cuando un tenant requiere un dominio personalizado, tu proceso de onboarding necesitará dar soporte a una forma de poner ese dominio en funcionamiento dentro de tu entorno multi-tenant.

Si el dominio ya existe y está siendo migrado a tu entorno, tu flujo de onboarding necesitará incluir los pasos requeridos para hacer posible esta migración. Sin embargo, también puede haber casos donde el tenant crea su dominio como parte de su proceso de onboarding. Como puedes imaginar, este sería un proceso mucho más complejo.

Acceso mediante un dominio único

Sin embargo, algunos proveedores SaaS —especialmente aquellos que usan un modelo B2C— utilizarán un único dominio para todos los tenants. En estos casos, todos los tenants usarán el mismo dominio para acceder a tu entorno SaaS.

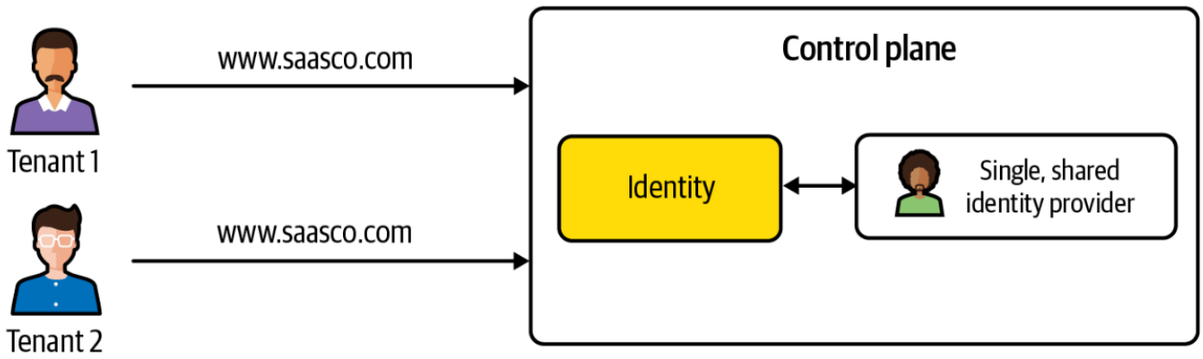

Figure 6-3. Single domain with a shared identity provider

En este ejemplo, tengo dos tenants que están accediendo a mi sistema a través de un único dominio compartido (`www.saasco.com`). Ahora, cuando estos tenants intentan autenticar a un usuario de tenant existente, son dirigidos al proveedor de identidad alojado dentro de nuestro control plane.

Donde esto se vuelve más interesante es cuando tu arquitectura soporta más variantes de tu experiencia de autenticación. En el Capítulo 4 hablé sobre cómo los proveedores de identidad pueden ofrecer diferentes elementos de agrupación para usuarios que te permiten tener políticas de autenticación separadas para cada tenant. Si tu proveedor de identidad soporta estos elementos de agrupación, esto puede permitirte ofrecer opciones de autenticación para los diferentes niveles de tu solución.

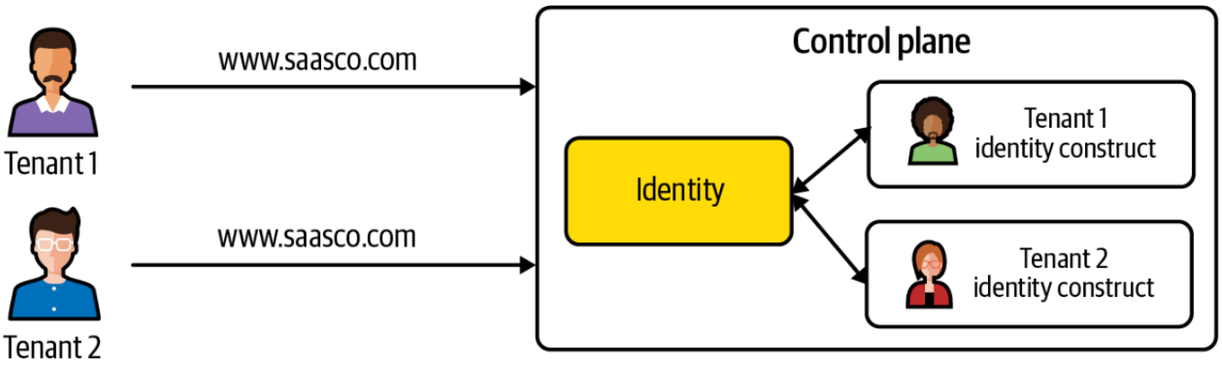

Figure 6-4. Single domain with separate identity constructs

Aquí es donde las cosas se complican un poco con el modelo de dominio único. Sin un dominio para identificar a los tenants cuando entran al entorno, no tienes contexto que te permita determinar qué elemento de identidad debe usarse para autenticar a tus usuarios.

Hay algunas formas en que podrías abordar la resolución del contexto de tenant para tu flujo de autenticación. Una estrategia es usar el dominio de la dirección de correo electrónico del usuario para asociar al usuario con un tenant determinado. En este modelo, asumirías que los tenants que provienen de un dominio/cliente específico se mapearían al tenant de ese cliente. Esto puede funcionar si puedes asumir que todos los tenants de un dominio pertenecen a un tenant específico. Sin embargo, esto impone límites en tu capacidad de dar soporte a un rango más amplio de usuarios con diversos dominios de correo electrónico. Otra posibilidad aquí es considerar el mapeo de identificadores de usuario individuales a tenants específicos.

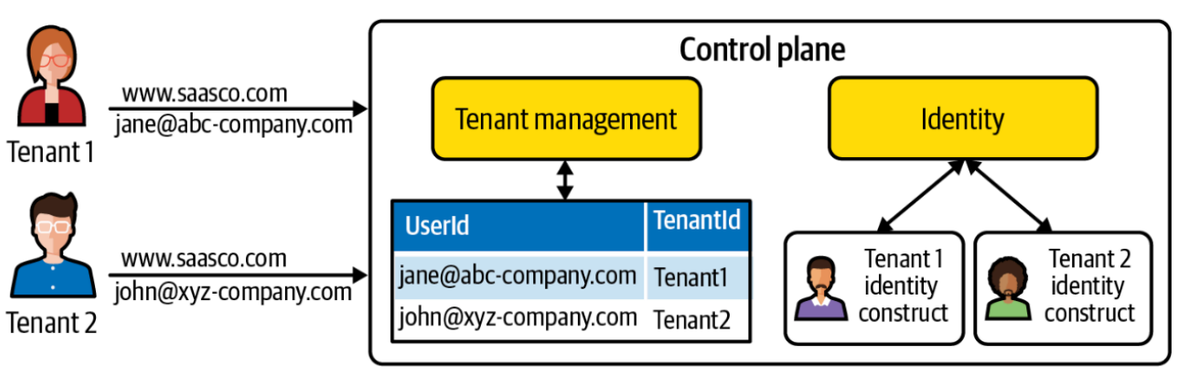

Figure 6-5. Authentication mapping based on tenant identifiers

El desafío del intermediario

Al observar la solución en la Figura 6-5, notarás que esta estrategia depende de un nivel de indirección para mapear exitosamente los tenants a sus proveedores de identidad.

La Figura 6-6 proporciona una vista conceptual de una experiencia de autenticación clásica que podrías haber implementado múltiples veces.

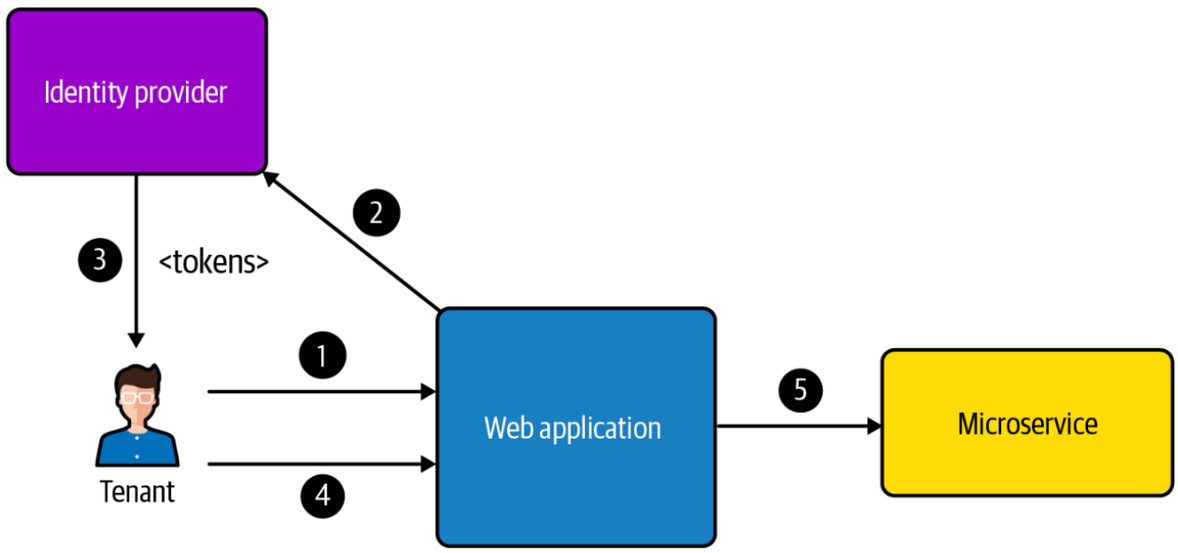

Figure 6-6. A classic web application authentication flow

La belleza de este flujo es que está orquestado completamente por los elementos de identidad de tu entorno. **La ruta hacia el proveedor de identidad y de regreso a tu aplicación web está mayormente fuera del control de tu código y se ajusta a los flujos de autenticación estándar que son soportados por los proveedores de identidad.**

Al mismo tiempo, también he hablado sobre cómo diferentes configuraciones de identidad y patrones de acceso pueden depender de código que pueda mapear un tenant a su elemento de identidad correspondiente. Estas estrategias a menudo requieren la inyección de elementos de mapeo adicionales que residen dentro del flujo de tu experiencia de autenticación.

En la Figura 6-5, por ejemplo, viste un caso donde el servicio de gestión de tenant se usó para buscar y mapear un usuario a un elemento de identidad. Este enfoque significa que tu flujo de autenticación no puede ir directamente al proveedor de identidad para procesar tu autenticación.

Cada vez que agregas esta capa de indirección a tu flujo de autenticación, estás esencialmente agregando un punto de escalabilidad y falla al modelo de autenticación de tu entorno. **Aunque esto puede ser exactamente lo que necesitas hacer, debes asegurarte de considerar las compensaciones asociadas con inyectarte en el flujo de autenticación.**

El flujo de autenticación multi-tenant

Es importante notar que este proceso está enfocado directamente en autenticar a un usuario de tenant y devolver la identidad SaaS. Generalmente, las partes móviles de esta experiencia se alinean con las especificaciones OAuth y OIDC que son implementadas por la mayoría de los proveedores de identidad. Aun así, es útil ver el flujo de extremo a extremo de esta experiencia para darte una mejor idea de lo que está sucediendo.

Un flujo de autenticación de ejemplo

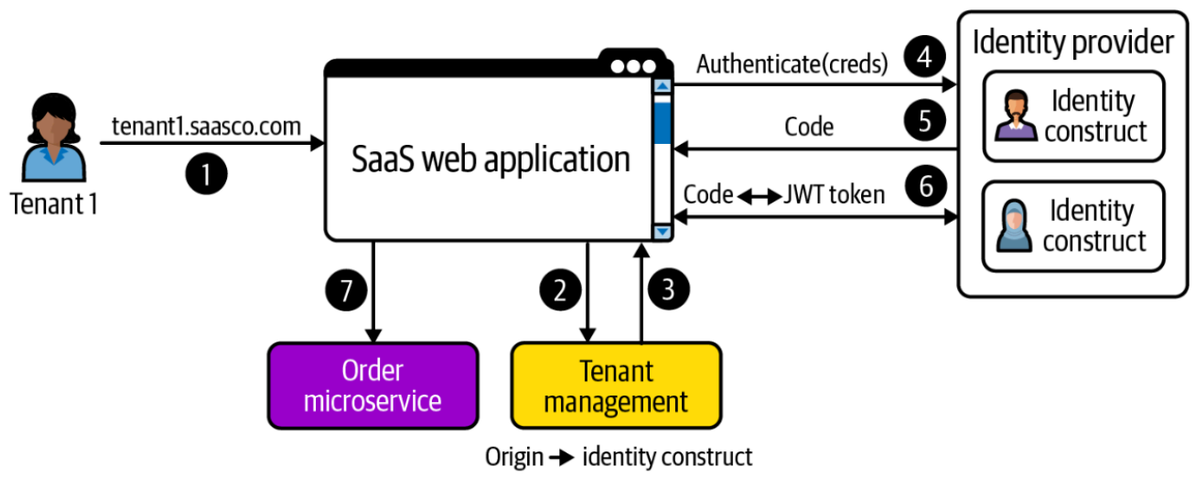

Figure 6-7. A sample multi-tenant authentication flow

En este ejemplo, he utilizado un modelo de subdominio por tenant como parte de este flujo de autenticación. Los tenants que se autentican en este entorno ingresan a través de su subdominio asignado, en este caso `tenant1.saasco.com`.

1. El usuario del tenant accede a nuestra aplicación web (paso 1) y nuestra aplicación determina que el usuario no está autenticado. Como el usuario no está autenticado, tu aplicación web redirigirá al usuario a la experiencia de inicio de sesión

de la aplicación donde el usuario ingresará sus credenciales. Esto es un poco diferente del flujo de autenticación clásico en que dependemos de la aplicación web para detectar y dirigir al usuario al formulario de inicio de sesión.

2. Ahora, antes de que puedas autenticar al usuario, necesitarás determinar el elemento de proveedor de identidad específico que está asociado con este tenant. Esto se logra llamando al servicio de gestión de tenant (paso 2) y solicitando la información de los elementos de identidad objetivo que se usarán para autenticar al usuario.
3. El servicio de gestión de tenant inspeccionará el origen (subdominio) del tenant, buscará su mapeo al proveedor de identidad y devolverá esta información a la aplicación web (paso 3).
4. Ahora tienes todo lo que necesitas para autenticar al usuario del tenant. Los siguientes pasos en este proceso siguen las partes clásicas de un flujo OAuth. Primero, llamamos al proveedor de identidad, pasando las credenciales del usuario del tenant (paso 4).
5. El proveedor de identidad entonces devolverá un código (paso 5)
6. antes de intercambiar este código por un JWT (paso 6).
7. Finalmente, ahora que tienes el JWT con nuestro contexto de tenant, este token puede ser inyectado en las llamadas a nuestros microservicios (paso 7).
8. Los tokens que resultan de este proceso, como se discutió en el Capítulo 4, se inyectan en los servicios downstream como bearer tokens donde el encabezado de autorización de tu solicitud HTTP se configura como un token "bearer" y se le asigna el valor del access token que regresa de tu flujo de autenticación.

Autenticación federada

Si tu solución, por ejemplo, necesita autenticar contra algún proveedor de identidad alojado externamente que está fuera de tu control, esto puede agregar una capa de complejidad a tu modelo general de autenticación multi-tenant.

No puedes realmente requerir que ese tercero incluya los claims personalizados que proporcionan el contexto de tenant. Al mismo tiempo, nuestro diseño multi-tenant depende fuertemente de su capacidad para emitir JWTs que incluyan este contexto de tenant.

La buena noticia es que existen soluciones de identidad federada que pueden cubrir partes de la experiencia multi-tenant. Con Amazon Cognito, por ejemplo, puedes tener claims personalizados configurados dentro de Cognito para usuarios que serán autenticados desde un proveedor externo.

La clave aquí es que si estás federando a un proveedor externo, tendrás que determinar qué estrategia usarás para adquirir el contexto de tenant. En algunos casos, esto puede requerir que manipules o inyectes JWTs para lograr la experiencia deseada.

No existe una autenticación única para todos los casos

La clave es que esto rara vez es tan simple como acceder a un proveedor de identidad y seguir cualquiera de los flujos de autenticación típicos que son descritos por los diversos proveedores de identidad. En cambio, tienes que envolver tus requisitos multi-tenant alrededor de estos flujos de identidad para alinearlos con los patrones multi-tenant que están implementados en tu entorno.

Enrutamiento de tenants autenticados

Después de haber pasado por la puerta principal de tu aplicación y estar listo para invocar servicios backend, aún necesitas considerar cómo el contexto de tenant puede influir en el enrutamiento de estas solicitudes.

Cuando hablo de enrutamiento aquí, me refiero generalmente a cómo los tenants se mapean a sus recursos correspondientes solicitud por solicitud.

Una vez que te has autenticado, tu aplicación comenzará a consumir los microservicios y la funcionalidad de tu aplicación SaaS.

Ahora, si todos tus servicios de aplicación están ejecutándose en un modelo pooled, puedes simplemente hacer estas llamadas directamente. Sin embargo, no sería inusual tener una huella de recursos de arquitectura más diversa para tus servicios de aplicación, algunos de los cuales se ejecutan en un modelo siloed y algunos en un modelo pooled.

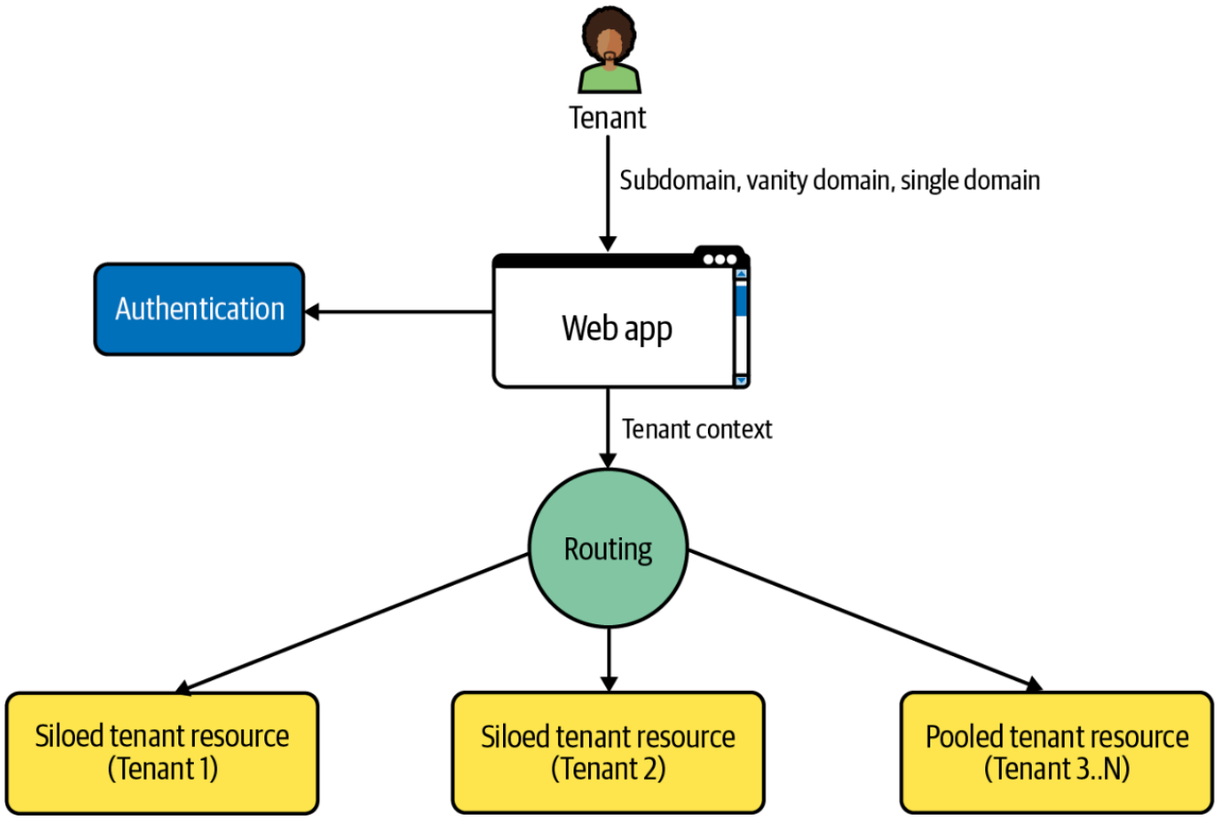

Figure 6-8. Tenant routing basics

El modelo de enrutamiento de tu aplicación también tiene implicaciones para la experiencia de onboarding de tu solución SaaS. A medida que cada nuevo tenant se incorpora a tu sistema, puede que necesites actualizar la configuración de tu infraestructura de enrutamiento para aprovisionar y configurar los elementos que serán necesarios para enrutar las cargas de trabajo de este nuevo tenant.

Enrutamiento con diferentes pilas tecnológicas

Para esta discusión, voy a examinar dos de los modelos tecnológicos SaaS más comunes: serverless y contenedores. Las secciones que siguen examinarán algunos de los matices asociados con el enrutamiento para cada una de estas pilas para darte una mejor idea de algunas de las variables que entran en juego como parte del desarrollo de un modelo de enrutamiento consciente del tenant.

Enrutamiento de tenant serverless

En un entorno serverless, tendrás un conjunto de funciones que se componen para crear los diversos microservicios que representan la funcionalidad de tu aplicación.

Con AWS Lambda, estas funciones típicamente se acceden a través de un API Gateway. Este gateway describe y expone los puntos de entrada HTTP a tus servicios, mapeando solicitudes a sus funciones correspondientes. En este sentido, puedes ver el API Gateway como una ruta esencial a través de la cual toda la actividad fluye hacia tus servicios de aplicación.

Ahora, si las funciones de nuestra aplicación serverless son todas pooled, entonces el rol del gateway es bastante directo. Todas las solicitudes simplemente se dirigen a su función objetivo sin considerar el contexto de tenant.

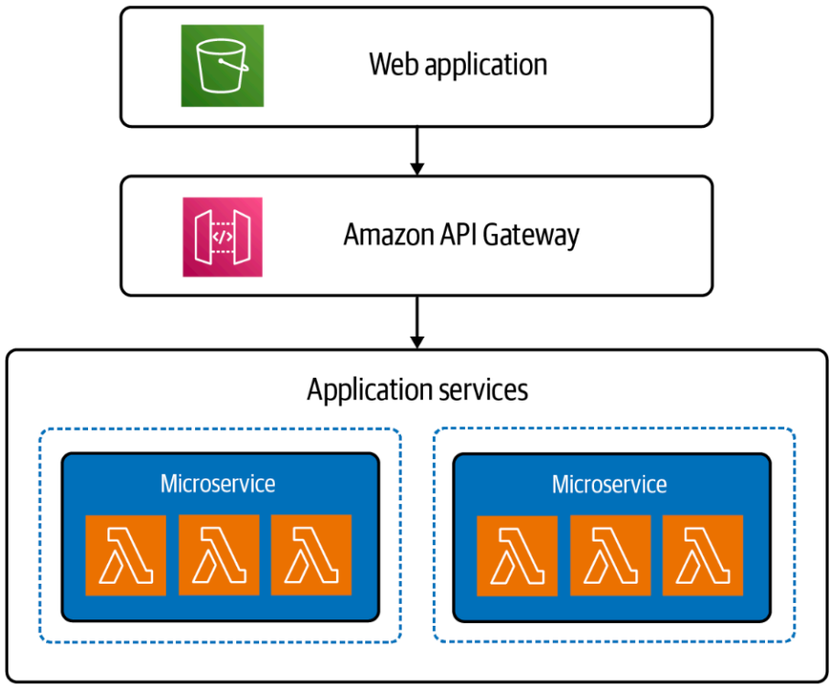

Figure 6-9. API Gateway routing in serverless environments

Sin embargo, imagina un escenario donde algunos o todos tus microservicios (y funciones) se ejecutan en un modelo siloed. Aquí es donde necesitarás evaluar el contexto de tenant y enrutar las solicitudes a las funciones correctas del tenant.

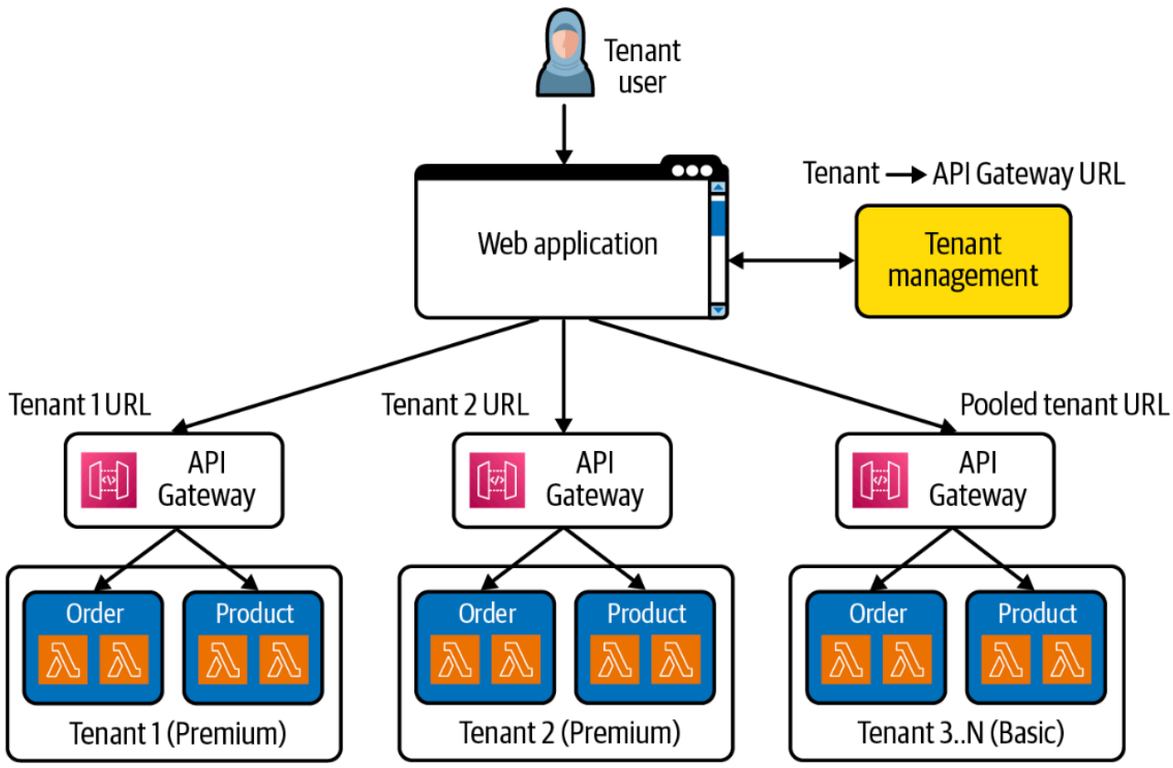

Figure 6-10. Routing to tenant-specific gateways

Asumí que tenía un único dominio para todos los tenants, lo cual requería el uso del servicio de gestión de tenant (en la esquina superior derecha del diagrama) para buscar la URL del API Gateway de cada tenant.

La desventaja de este modelo particular es que requiere que el cliente juegue un rol en este ejercicio de mapeo. El cliente debe adquirir la URL del servicio de gestión de tenant y aplicar esa URL como parte de las solicitudes.

Realizar estos mapeos en cada solicitud puede agregar sobrecarga y latencia que impacta el rendimiento de tu solución. **El enfoque más típico aquí es introducir una estrategia de caché a nivel de mapeo o gateway para retener los tenants mapeados recientemente. La clave es asegurarte de que estás considerando los impactos de rendimiento de este mapeo en la estrategia general.**

Tener un gateway por tenant puede no escalar bien en entornos con un gran número de tenants. **Aquí es donde limitar el número de recursos de tenant siloed se convierte en una pieza importante del rompecabezas.**

Enrutamiento de tenant en contenedores

Como contraste, veamos cómo podrías implementar enrutamiento consciente del tenant en un entorno donde tu application plane está construido principalmente con Kubernetes.

Dentro de una arquitectura multi-tenant de Kubernetes, tienes muchos elementos nativos que pueden usarse para dar forma a la huella de recursos de enrutamiento de tu aplicación SaaS.

La lista de opciones es bastante extensa, pero, por ahora, quiero ilustrar cómo podrías usar un service mesh para construir esta experiencia de enrutamiento.

Existen múltiples implementaciones de service mesh. Para esta solución, he elegido usar Istio para implementar nuestro modelo de enrutamiento.

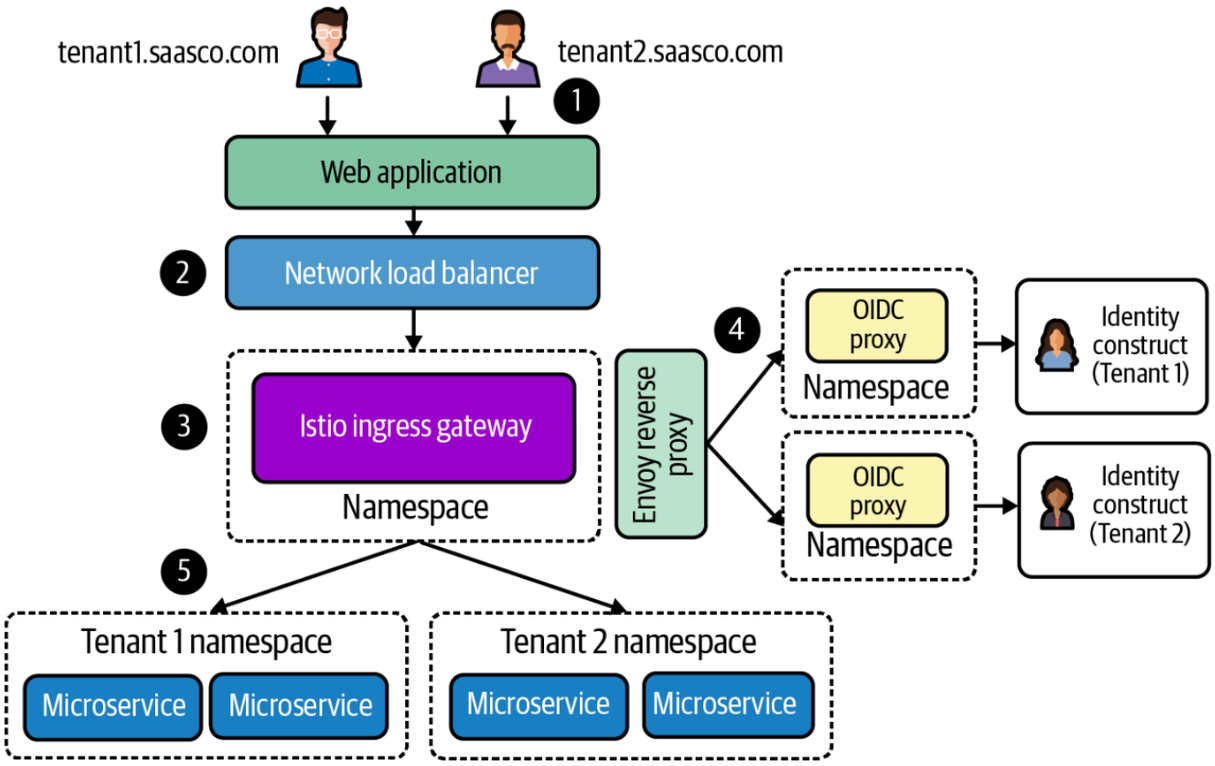

Figure 6-11. Routing tenant requests with a service mesh

La solicitud luego pasa a través del balanceador de carga de red y llega al gateway de ingreso de Istio que se ejecuta en su propio namespace dentro de tu clúster de Kubernetes. Es este gateway el que orquesta y aplica todas las políticas que se requieren para enrutar hacia nuestros proveedores de identidad así como hacia nuestros microservicios de aplicación.

Supongamos que, en este ejemplo, el tenant no está autenticado. En este escenario, el gateway enviará la solicitud a través de un proxy inverso Envoy (paso 4), que usará el subdominio de origen entrante para enrutar las solicitudes de autenticación al proxy OIDC específico del tenant que también se ejecuta en un namespace separado de Kubernetes.

En este ejemplo, verás que tengo dos tenants ejecutándose en namespaces separados de Kubernetes. Esto significa que nuestro gateway debe examinar el origen y enrutar cada solicitud al namespace de tenant apropiado.

Aunque hay muchas partes móviles en esta solución, para mí todavía se siente un poco más elegante que aquellas estrategias que dependen de que tu servicio busque y mapee tenants.

Generalmente, si puedes delegar estas responsabilidades de enrutamiento a otras piezas de infraestructura que ya gestionan enrutamiento, esto a menudo representa una experiencia de enrutamiento más limpia y manejable.