Aislamiento de Tenant

Arquitectura de 9 Capas para Infraestructura Compartida

Hadda TIKIJJA
0DATA Lab, Francia

RESUMEN

El aislamiento de tenant en una infraestructura compartida no puede reposar sobre una única capa. Documentamos la arquitectura de aislamiento en **nueve capas** desplegada sobre la infraestructura 0DATA: Row-Level Security de PostgreSQL, SPINA por tenant, NOVA por tenant, Cockpit por dominio, ALFA por nivel, PWA offline, mapa de puertos, sujetos NATS con espacio de nombres y DNS wildcard. Cada capa funciona de forma independiente; el fallo de una no compromete las demás. Esta arquitectura se inspira en las **uniones estrechas** del epitelio biológico, donde cada célula mantiene su propia membrana participando a la vez en el tejido común. Describimos cada capa en detalle, presentamos el mapa de puertos documentado (ningún servicio escucha en 0.0.0.0) e informamos de los resultados del test de intrusión cross-tenant — todos los intentos de lectura entre tenants fracasaron.

EN UNA FRASE

Nueve capas de aislamiento independientes garantizan que un tenant no puede ni ver, ni alcanzar, ni alterar los datos de otro tenant — en ningún nivel de la pila.

1. Por Qué Nueve Capas

1.1 La Defensa en Profundidad No Es una Metáfora

En ciberseguridad clásica, la defensa en profundidad es un principio teórico. En una infraestructura compartida que aloja clientes MSP con intereses contrapuestos, se convierte en una necesidad vital. Un proveedor de servicios gestionados puede alojar simultáneamente un bufete de abogados, una clínica y un competidor directo del primero — en el mismo servidor físico.

El enfoque tradicional del multi-tenant se basa en una separación a nivel de aplicación: la aplicación filtra las consultas según un `tenant_id`. Este enfoque fracasa estrepitosamente cuando un bug, una falla de inyección o una escalada de privilegios elude esa única capa.

1.2 El Modelo Biológico: Membrana Celular y Uniones Estrechas

El epitelio intestinal humano ilustra nuestro paradigma. Cada célula posee su propia **membrana fosfolipídica** — es la primera barrera. Entre las células, las **uniones estrechas** (tight junctions) forman una segunda barrera, impermeable incluso a las moléculas pequeñas. Una tercera barrera — el **glucocáliz** — filtra las macromoléculas. Ninguna de estas barreras es suficiente por sí sola; su superposición crea una estanqueidad cercana a lo absoluto.

TRANSPOSICIÓN BIOMIMÉTICA

Membrana celular → Cada tenant tiene sus propias instancias SPINA, NOVA, ALFA
Uniones estrechas → PostgreSQL RLS impide cualquier fuga a nivel de la base de datos
Glucocáliz → El mapa de puertos, el DNS wildcard y NATS con espacio de nombres forman una capa de filtración de red

1.3 Las Nueve Capas

CAPA	NOMBRE	DOMINIO	MECANISMO
1	PostgreSQL RLS		Políticas Row-Level Security

		Base de datos	
2	Esquema PostgreSQL	Base de datos	Esquema lógico por tenant
3	Políticas de aplicación	Base de datos	Restricciones CHECK, triggers
4	SPINA v2	Firma	HMAC-SHA256 local + eIDAS RFC 3161
5	NOVA	Observación	Instancia por tenant, bind 127.0.0.1
6	ALFA	Decisional	Motor común, memoria aislada
7	Cockpit	Interfaz	Dominio propio, JWT por tenant
8	DNS / NATS	Comunicación	DNS wildcard, sujetos con namespace
9	Mapa de puertos	Red	Ningún servicio en 0.0.0.0, puertos documentados

2. Capas 1–3: PostgreSQL, la Base

2.1 Row-Level Security (Capa 1)

Row-Level Security de PostgreSQL es el mecanismo más profundo del aislamiento. Cada fila de cada tabla lleva un `tenant_id`. Una política RLS aplicada a cada tabla garantiza que la sesión de PostgreSQL conectada para el tenant A no puede leer físicamente las filas del tenant B.

```
ALTER TABLE organisms ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON organisms
    USING (tenant_id = current_setting('app.current_tenant_id')::uuid);
```

Figura 1: Política RLS de PostgreSQL — la columna tenant_id se compara con la variable de sesión.

La función `current_setting` lee una variable de sesión de PostgreSQL. Esta variable la establece el pool de conexiones en el momento de la autenticación del tenant. Ninguna

consulta de la aplicación puede modificarla — es de solo lectura desde la aplicación. Esta capa es inviolable sin acceso de superusuario de PostgreSQL.

2.2 Esquema por Tenant (Capa 2)

Más allá del RLS, cada tenant dispone de un **esquema lógico dedicado** — no un esquema PostgreSQL en el sentido de `CREATE SCHEMA`, sino una proyección de datos restringida. Las vistas materializadas, los índices parciales y las secuencias se filtran por `tenant_id`. Un tenant no puede enumerar a los demás tenants, porque la propia tabla `tenants` está protegida por RLS.

Esta capa impide los ataques por inferencia: aunque un tenant lograse medir el tiempo de ejecución de una consulta (timing attack), los índices parciales garantizan que los datos de los demás tenants nunca se recorren.

2.3 Restricciones y Triggers (Capa 3)

La tercera capa de PostgreSQL utiliza restricciones `CHECK` y triggers `BEFORE INSERT/UPDATE` para impedir la escritura de datos que no respetasen el aislamiento:

```
ALTER TABLE organisms ADD CONSTRAINT chk_tenant_owns
    CHECK (tenant_id IS NOT NULL
           AND tenant_id = current_setting('app.current_tenant_id')::uuid);
```

Figura 2: Restricción CHECK que impide la escritura cross-tenant.

Un trigger adicional bloquea cualquier intento de modificar la columna `tenant_id` tras la inserción. Esta redundancia es deliberada: si la política RLS se desactivara por error (por ejemplo durante un mantenimiento), las restricciones y los triggers mantienen el aislamiento.

3. Capas 4–6: SPINA, NOVA, ALFA

3.1 SPINA v2 — Firma por Tenant (Capa 4)

SPINA (Signature Protocol for Integrity and Non-repudiation of Assets) es el protocolo de firma de 0DATA. En la versión 2, cada tenant posee:

- Una **clave HMAC-SHA256 local**, generada y almacenada en el espacio del tenant
- Un **certificado eIDAS RFC 3161** para el sellado de tiempo cualificado
- Un **keystore aislado**, cifrado con una frase de contraseña derivada del secreto del tenant

Dos tenants nunca comparten material criptográfico. Si el tenant A compromete su clave HMAC, las firmas del tenant B siguen siendo válidas. El sellado de tiempo eIDAS está federado a nivel de infraestructura, pero el timestamp token se almacena en el esquema del tenant.

La verificación de firma se ejecuta en el contexto PostgreSQL del tenant. La función `spina_verify()` utiliza `SECURITY DEFINER` con un `SET app.current_tenant_id` explícito — no puede fugarse fuera del tenant.

3.2 NOVA — Instancia por Tenant (Capa 5)

NOVA es el motor de observación de la red. Cada tenant MSP posee su **propia instancia NOVA**, ejecutada como un proceso dedicado:

- Bind exclusivo en `127.0.0.1`, puerto distinto para cada tenant
- Base de datos de escaneo local (SQLite o esquema PostgreSQL aislado)
- Ningún uso compartido de memoria entre instancias
- Archivo de configuración distinto (`/opt/nova/tenants/<tenant_id>/config.yaml`)

Un tenant que controla su red local (y por tanto los objetivos escaneados por NOVA) no puede influir en la instancia NOVA de otro tenant. Los procesos están aislados a nivel del sistema operativo — `kill -9` sobre el PID del tenant A no afecta al tenant B.

3.3 ALFA — Motor Común, Memoria Aislada (Capa 6)

ALFA (Analytics Layer for Forensic Assessment) es el motor decisional compartido. A diferencia de NOVA, ALFA utiliza un **motor común** para la eficiencia de recursos, pero con un **aislamiento de memoria estricto**:

- Cada nivel de análisis (tenant, sitio, equipo) está compartimentado
- Los datos se cargan en segmentos de memoria estancos
- Los resultados del análisis se escriben en el esquema PostgreSQL del tenant
- No se realiza ninguna correlación entre tenants sin consentimiento explícito

ALFA aplica el principio de *necesidad de conocer* a nivel algorítmico: un modelo de análisis ejecutado para el tenant A solo recibe los datos del tenant A. Las estadísticas agregadas multi-tenant (útiles para el benchmarking) utilizan datos anonimizados con privacidad diferencial (ε = 1.0).

4. Capas 7–9: Cockpit, DNS, NATS

4.1 Cockpit — Dominio Propio y JWT por Tenant (Capa 7)

Cada tenant accede a la interfaz Cockpit a través de un **dominio dedicado**: `client.odata.fr`. El certificado SSL (Let's Encrypt) se emite para este dominio. El JWT de autenticación se firma con una clave secreta propia del tenant y contiene el `tenant_id` en sus claims.

```
Authorization: Bearer eyJ...tenant_id:"a1b2c3d4-..."...
```

Figura 3: Cabecera de autenticación de Cockpit — JWT que contiene el tenant_id.

El middleware de nginx valida el JWT antes de enrutar la solicitud. Un intento de acceso a `client-b.odata.fr` con un JWT del tenant A se rechaza a nivel del reverse proxy — antes incluso de que la solicitud llegue a la API. El propio dominio constituye una frontera de aislamiento: un tenant no puede adivinar el subdominio de otro tenant (los nombres son UUIDs o identificadores opacos).

4.2 DNS Wildcard y NATS con Espacio de Nombres (Capa 8)

La capa 8 opera a nivel de la comunicación entre servicios:

DNS wildcard. El registro `*.odata.fr` apunta a la infraestructura. Cada subdominio se resuelve, pero el servidor de aplicaciones filtra los dominios no registrados. Un tenant no puede crear un subdominio arbitrario — el registro DNS está controlado por la infraestructura.

NATS con espacio de nombres. El bus de mensajes NATS utiliza sujetos con prefijo del tenant:

```
tenant.a1b2c3d4.nova.scan.start
tenant.a1b2c3d4.spina.sign
tenant.e5f6g7h8.nova.scan.start
```

Figura 4: Sujetos NATS con prefijo de tenant. Las ACL restringen el acceso por prefijo.

Las ACL de NATS están configuradas para que las credenciales del tenant A solo puedan publicar o suscribirse a los sujetos `tenant.a1b2c3d4.*`. El servidor NATS aplica estas ACL a nivel de protocolo — un cliente no puede eludir el espacio de nombres.

4.3 Mapa de Puertos — Ningún Servicio en 0.0.0.0 (Capa 9)

La capa 9 es la más física: el mapa de puertos documentado. Ningún servicio interno escucha en `0.0.0.0`. Cada servicio está vinculado a `127.0.0.1` con un puerto explícito. El firewall (UFW, política default deny) bloquea todo el tráfico entrante que no esté explícitamente autorizado. Esta capa se documenta en detalle en la sección 5.

5. Mapa de Puertos Documentado

La siguiente tabla documenta la totalidad de los puertos en escucha en la infraestructura. La columna **Bind** indica la interfaz de escucha. **Ningún servicio escucha en 0.0.0.0**, salvo los puertos públicos deliberadamente expuestos (HTTP/S, SSH, SIP).

5.1 Puertos Públicos (expuestos)

PUERTO	PROTOCOLO	SERVICIO	JUSTIFICACIÓN
22	TCP	SSH	Acceso administrativo
80	TCP	HTTP (nginx)	Redirección HTTPS + ACME
443	TCP/UDP	HTTPS (nginx)	Reverse proxy, HTTP/3 QUIC
5060	UDP	SIP (FreeSWITCH)	Telefonía VoIP
5080	UDP	SIP (FreeSWITCH)	Telefonía VoIP

5.2 Puertos Internos (bind 127.0.0.1)

PUERTO	SERVICIO	TENANT	ROL
5432	PostgreSQL	Compartido (RLS)	Base de datos principal
6379	Redis	Compartido (con prefijo)	Caché, sesiones
4222	NATS	Compartido (con namespace)	Bus de mensajes
5090–5099	Instancias NOVA	Por tenant	Escaneo de red
5100–5109	API NOVA	Por tenant	API REST de NOVA
5110–5119	Workers SPINA	Por tenant	Firma/sellado de tiempo
8300	API principal	Compartido (RLS)	API FastAPI
8400–8409	Workers Cockpit	Por tenant	Interfaz web Cockpit
9090	Cockpit de sistema	Infraestructura	Administración del sistema

5.3 Reglas UFW

```
ufw default deny incoming
ufw default allow outgoing
ufw allow 22/tcp
ufw allow 80/tcp
ufw allow 443
ufw allow 5060/udp
ufw allow 5080/udp
```

Figura 5: Reglas UFW — solo 5 puertos están expuestos públicamente.

Todos los demás puertos están bloqueados a nivel de firewall. El bind `127.0.0.1` añade una segunda barrera: aunque el firewall se desactivara, los servicios internos no serían accesibles desde el exterior.

6. Verificación — Test de Intrusión Cross-Tenant

6.1 Protocolo de Test

El test de intrusión cross-tenant se llevó a cabo el 23 de julio de 2026. Se crearon dos tenants de prueba:

- **Tenant A** (`a1b2c3d4-...`): datos de prueba «clínica veterinaria»
- **Tenant B** (`e5f6g7h8-...`): datos de prueba «gabinete contable»

Los tests se ejecutaron desde el contexto del tenant A, intentando acceder a los datos del tenant B. Todos los tests utilizaron credenciales válidas del tenant A.

6.2 Vectores Probados

#	VECTOR	MÉTODO	RESULTADO
1	Lectura directa SQL	`SELECT * FROM organisms` (sin filtro tenant_id)	**Fallo** — RLS devuelve 0 filas
2	Modificación de tenant_id	`UPDATE organisms SET tenant_id = 'e5f6...'`	**Fallo** — La restricción CHECK lo rechaza
3			

	Acceso API cross-tenant	`GET /api/organisms?tenant_id=e5f6...`	**Fallo** — La API ignora el parámetro, usa el JWT
4	JWT falsificado	JWT con `tenant_id: "e5f6..."` firmado con clave A	**Fallo** — Firma inválida
5	Acceso Cockpit dominio B	Conexión a `tenant-b.odata.fr` con JWT A	**Fallo** — Dominio + JWT no coinciden, 403
6	Suscripción NATS	`nats.sub("tenant.e5f6.*")` con credenciales A	**Fallo** — La ACL rechaza la suscripción
7	Proceso NOVA	Conexión al puerto NOVA del tenant B	**Fallo** — Bind 127.0.0.1, el firewall bloquea
8	Clave SPINA	Intento de verificación con clave del tenant B	**Fallo** — HMAC no coincide
9	Enumeración DNS	Intento de descubrimiento de subdominios	**Fallo** — UUIDs opacos, sin transferencia de zona

6.3 Resultado

RESULTADO DEL PENTEST CROSS-TENANT

Cero éxitos en nueve vectores. Cada capa bloqueó los intentos que le concernían. Las capas 1 (RLS) y 2 (restricciones) bloquearon los intentos SQL. La capa 4 (SPINA) bloqueó la falsificación criptográfica. La capa 8 (NATS) bloqueó la interceptación de mensajes. La capa 9 (firewall + bind) bloqueó el acceso directo a la red.

Ninguna capa fue eludida. Ninguna combinación de vectores permitió inferir datos entre tenants.

7. Por Qué el Multi-Tenant Clásico Es Insuficiente

7.1 El Modelo de Capa Única

La arquitectura multi-tenant clásica — la de la mayoría de los SaaS — se basa en una separación única a nivel de aplicación. La aplicación añade `WHERE tenant_id = ?` a cada consulta. Este enfoque presenta tres fallos estructurales:

1. **Punto único de fallo.** Un olvido de la cláusula WHERE en una unión, una migración de esquema mal probada o un bug del ORM expone los datos de todos los tenants.
2. **Sin defensa contra la escalada de privilegios.** Un atacante que obtiene acceso de administrador de la aplicación (mediante inyección SQL, credential stuffing o una vulnerabilidad de dependencia) elude la totalidad del aislamiento.
3. **Sin aislamiento a nivel de infraestructura.** Base de datos, caché, bus de mensajes — todos estos componentes se comparten sin barrera interna. Una fuga de memoria en Redis expone las sesiones de todos los tenants.

7.2 Comparación con el Enfoque 0DATA

DIMENSIÓN	MULTI-TENANT CLÁSICO	AISLAMIENTO 0DATA (9 CAPAS)
Base de datos	`WHERE tenant_id = ?`	RLS + restricciones + triggers
Firma	Compartida	HMAC por tenant + eIDAS
Observación	Compartida	Instancia NOVA por tenant
Análisis	Compartido	ALFA con memoria aislada
Interfaz	Mismo dominio	Dominio por tenant, JWT dedicado
Mensajería	Sujetos compartidos	NATS con namespace y ACL
Red	Servicios en 0.0.0.0	Bind 127.0.0.1, firewall
Profundidad	1 capa	9 capas independientes

7.3 El Coste del Aislamiento

El aislamiento en nueve capas tiene un coste: complejidad de despliegue, consumo adicional de memoria (una instancia NOVA por tenant) y mantenimiento de las ACL de NATS. Este coste se acepta como una **inversión estructural**. En el contexto MSP, don-

de una fuga de datos cross-tenant puede acarrear acciones judiciales, sanciones RGPD y la pérdida de clientes, el coste del aislamiento es inferior al coste de una violación.

La arquitectura está documentada, automatizada (despliegue con Ansible) y probada (pentest cross-tenant en cada despliegue). La complejidad se mantiene bajo control mediante la reproducibilidad.

8. Trabajos Relacionados y Perspectivas

El aislamiento de tenant en infraestructuras compartidas es un tema activo en la literatura. Las arquitecturas de tipo «jaula» (Google Borg, AWS Nitro Enclaves) utilizan la virtualización hardware para aislar los tenants — un enfoque robusto pero costoso en recursos. Las arquitecturas «shared-nothing» (cada tenant en su propio clúster) ofrecen un aislamiento perfecto a costa de una ineficiencia económica.

El enfoque 0DATA ocupa un punto de equilibrio: **infraestructura compartida, datos aislados**. El modelo biológico de las uniones estrechas guía esta arquitectura — cada capa es independiente, redundante y verificable.

Los trabajos futuros incluyen:

- **Rotación automática de las claves SPINA por tenant** (periodicidad configurable)
- **Privacidad diferencial multi-tenant** para las agregaciones estadísticas
- **Aislamiento hardware** (SGX/TDX) para tenants con altas exigencias regulatorias
- **Auditoría continua cross-tenant** — test automático de las nueve capas en cada despliegue

Referencias

[1] TIKIJJA, Hadda. «La Disciplina». 0DATA Lab, Documento 001, Julio 2026.

[2] TIKIJJA, Hadda. «El Sistema Nervioso». 0DATA Lab, Documento 003, Julio 2026.

[3] TIKIJJA, Hadda. «El Injerto Digital». 0DATA Lab, Documento 004, Julio 2026.

[4] TIKIJJA, Hadda. «El Sistema Inmunitario». 0DATA Lab, Documento 005, Julio 2026.

[5] TIKIJJA, Hadda. «Auditoría de Superficie». 0DATA Lab, Documento 014, Julio 2026.

[6] PostgreSQL Documentation. «Row Security Policies». PostgreSQL 16, 2024.

[7] NATS Documentation. «Subject-Based Messaging and ACLs». Synadia Communications, 2024.

[8] eIDAS Regulation (EU) No 910/2014. «Electronic Identification and Trust Services».

[9] RFC 3161. «Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)».

[10] Dwork, C., Roth, A. «The Algorithmic Foundations of Differential Privacy». Foundations and Trends in Theoretical Computer Science, 2014.

Agradecimientos. Al equipo de PostgreSQL por el Row-Level Security, a los mantenedores de NATS por el espacio de nombres mediante ACL, y a la infraestructura 0DATA que soportó el pentest cross-tenant sin inmutarse. El aislamiento no es una funcionalidad — es la estructura misma de lo vivo.

«El aislamiento no es una funcionalidad. Es la estructura misma de lo vivo.»