

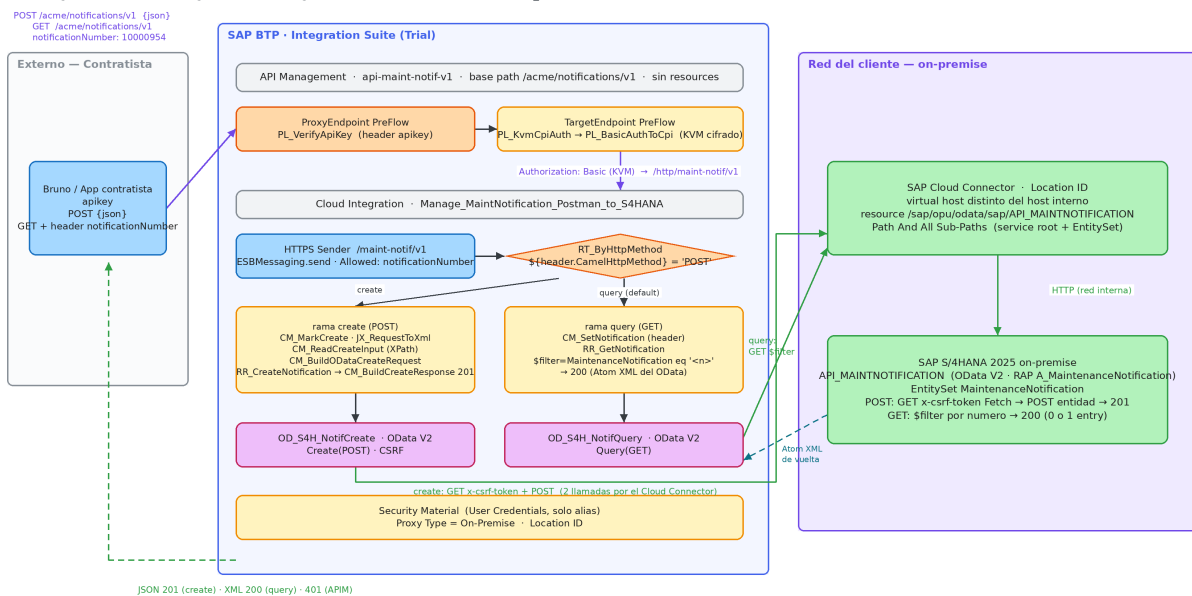
Crear y consultar avisos de mantenimiento en S/4HANA on-premise desde una API

SAP Integration Suite · Cloud Integration + API Management + Cloud Connector · Caso práctico MIX-003

Un endpoint REST que con POST **crea** un aviso de avería en S/4HANA (OData V2 Create con token CSRF, a través de Cloud Connector) y con GET **consulta** un aviso por su número. Un solo iFlow con Router por método HTTP, gobernado por API Management con API Key. Construido y probado de punta a punta en un tenant Trial contra un S/4HANA 2025 on-premise real.

Crear y consultar avisos de mantenimiento S/4HANA on-premise desde una API

SAP Integration Suite · Cloud Integration + API Management + Cloud Connector · OData V2 API_MAINTNOTIFICATION · as-built Variante A



Un endpoint, dos métodos. POST = OData Create con CSRF (2 llamadas al backend) · GET = Query con \$filter · Sin manejo de errores en esta iteración

Arquitectura end-to-end

1. El planteamiento

Contexto (empresa ficticia). Acme Industrial Services mantiene la planta DT01. En el caso anterior (MIX-002) los contratistas externos consiguieron **consultar** órdenes de mantenimiento desde su app móvil. El dolor siguiente iba en el sentido contrario: cuando un técnico detecta una avería en terreno, la anota en papel y la dicta por radio al planificador, que la transcribe a mano en SAP horas después. Se pierden datos (qué equipo, qué prioridad, quién reportó) y los avisos quedan **sin objeto técnico**, lo que después impide planificar la orden.

Lo que pidió Mantenimiento. Dos operaciones en una sola API:

1. Crear un aviso de avería (M2) contra un **equipo** concreto, con texto, prioridad y quién lo reporta, y recibir de vuelta el **número de aviso** que asignó SAP.
2. Consultar un aviso por su número para confirmar que quedó registrado.

Lo que exigió Seguridad. Nada llega directo al runtime de Cloud Integration ni al S/4HANA; el S/4HANA **no se publica a internet** (todo pasa por Cloud Connector); cada consumidor entra con su propia API Key.

Contrato del consumidor.

```
POST /acme/notifications/v1      GET /acme/notifications/v1
apikey: <clave de la app>        apikey: <clave de la app>
Content-Type: application/json   notificationNumber: 10000954

{
  "notificationType": "M2",
  "description": "Ruido anormal en rodamiento",
  "priority": "2",
  "equipment": "10000",
  "reportedBy": "CONTRATISTA1"
}
```

2. Antes de diseñar: mirar el backend

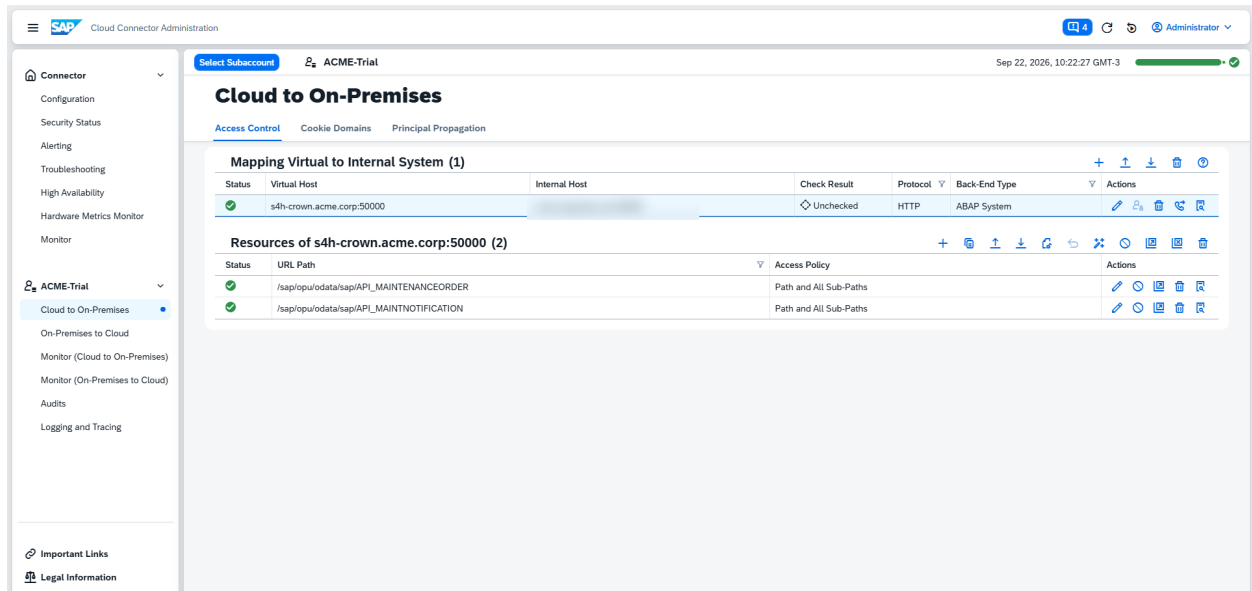
Antes de escribir una línea del iFlow verifiqué el servicio en el S/4HANA. Tres hallazgos que condicionaron todo el diseño:

Hallazgo	Consecuencia
API_MAINTNOTIFICATION es una proyección RAP (A_MaintenanceNotification) expuesta por SADL como OData V2	el Create exige el ciclo CSRF completo; los números viajan sin ceros a la izquierda
En el Behavior Definition, FunctionalLocation es readonly y NotificationText es el único campo mandatory	el equipo se envía en TechnicalObject + TechObjlsEquipOrFuncnlLoc = EAMS_EQUI; la ubicación técnica, la planta y el puesto los deriva el backend
Un POST con solo tipo + texto crea un aviso huérfano (sin equipo, sin planta)	el contrato del consumidor exige equipment; un 201 no prueba que el dato quedó bien

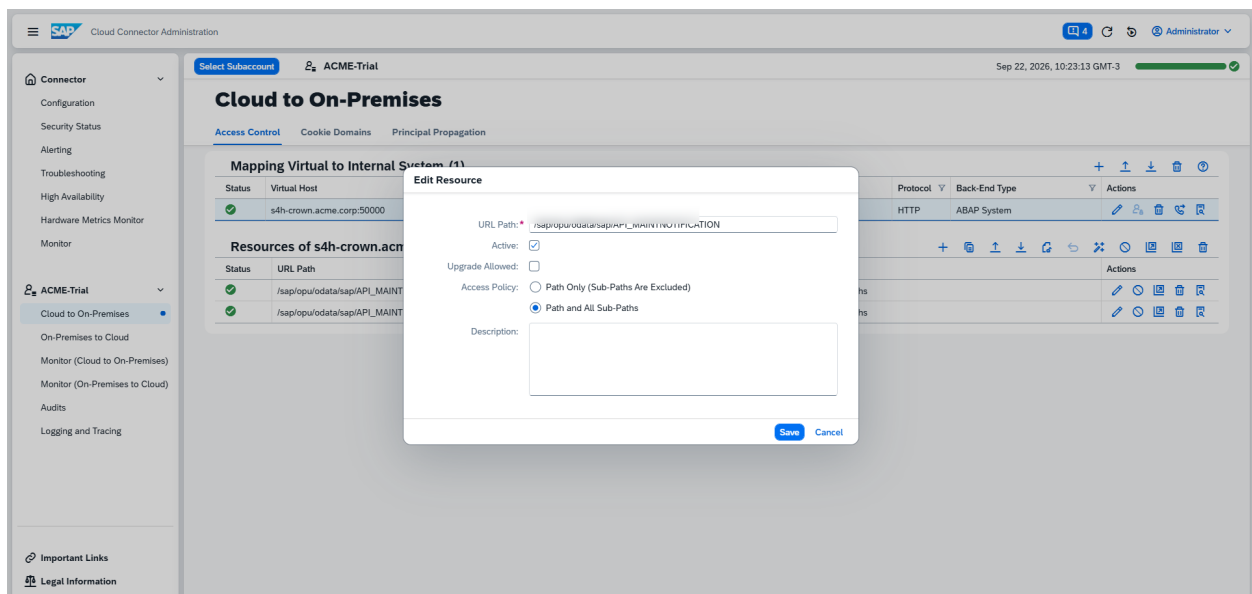
3. La solución

3.1 Cloud Connector — un resource nuevo

Sobre el mapping ya existente hacia el S/4HANA (virtual host distinto del interno, Location ID S4H_CROWN) se publica /sap/opu/odata/sap/API_MAINTNOTIFICATION con **Path And All Sub-Paths**. Es obligatorio: el Create hace dos llamadas, el GET al service root para el token y el POST al EntitySet.



Cloud Connector: mapping y resources



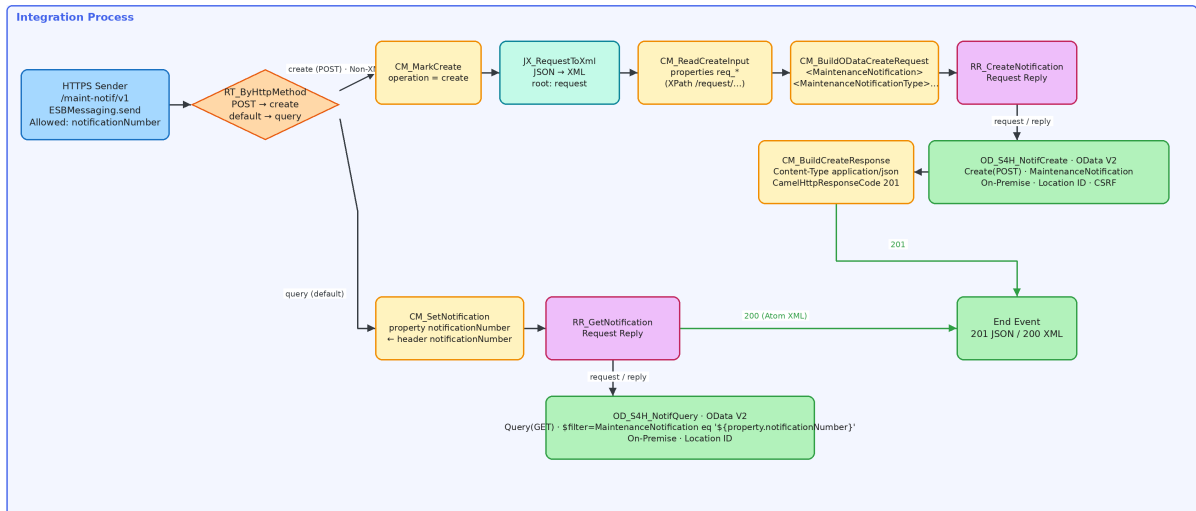
Resource con Path And All Sub-Paths

3.2 El iFlow: un endpoint, dos ramas

Manage_MaintNotification_Postman_to_S4HANA. HTTPS Sender en /maint-notif/v1; un Router por `${header.CamelHttpMethod}` separa create (POST) de query (GET). Sin Groovy: JSON to XML Converter, Content Modifiers con XPath y dos receivers OData V2.

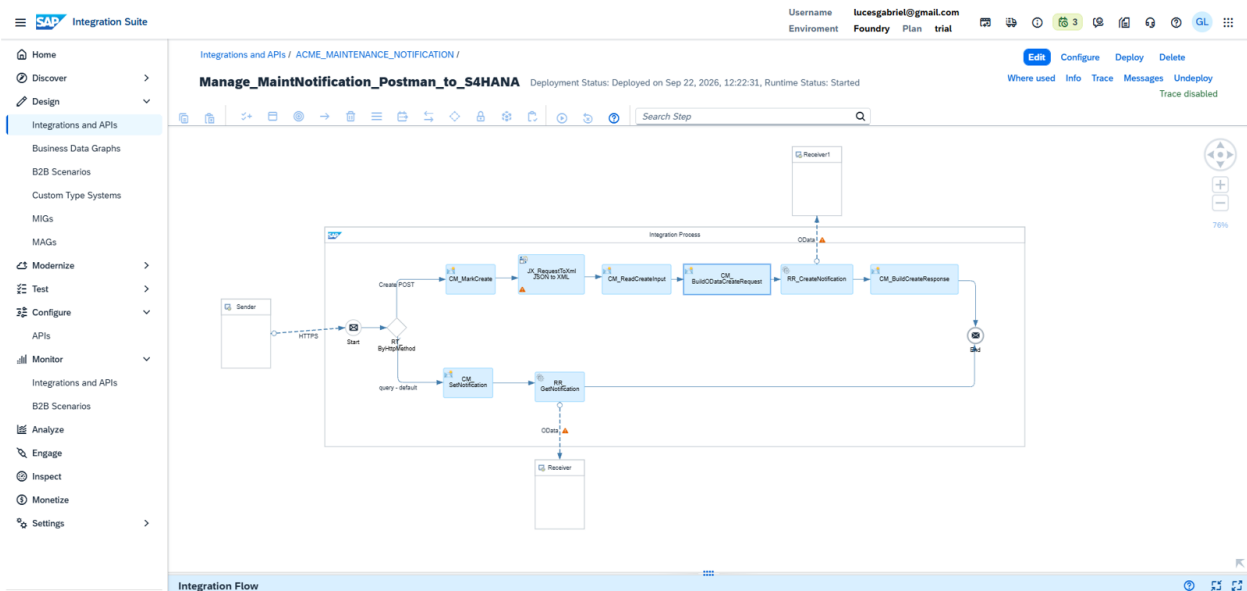
Detalle del iFlow Manage_MaintNotification_Postman_to_S4HANA (as-built)

Un Integration Process, dos ramas por metodo HTTP, dos receivers OData V2 · cero Groovy · el CSRF lo negocia el adapter



La ruta create del Router DEBE ser Non-XML: en XML el modelador infiere mensaje XML y marca en rojo al JSON to XML Converter (hallazgo del build).

Detalle del iFlow

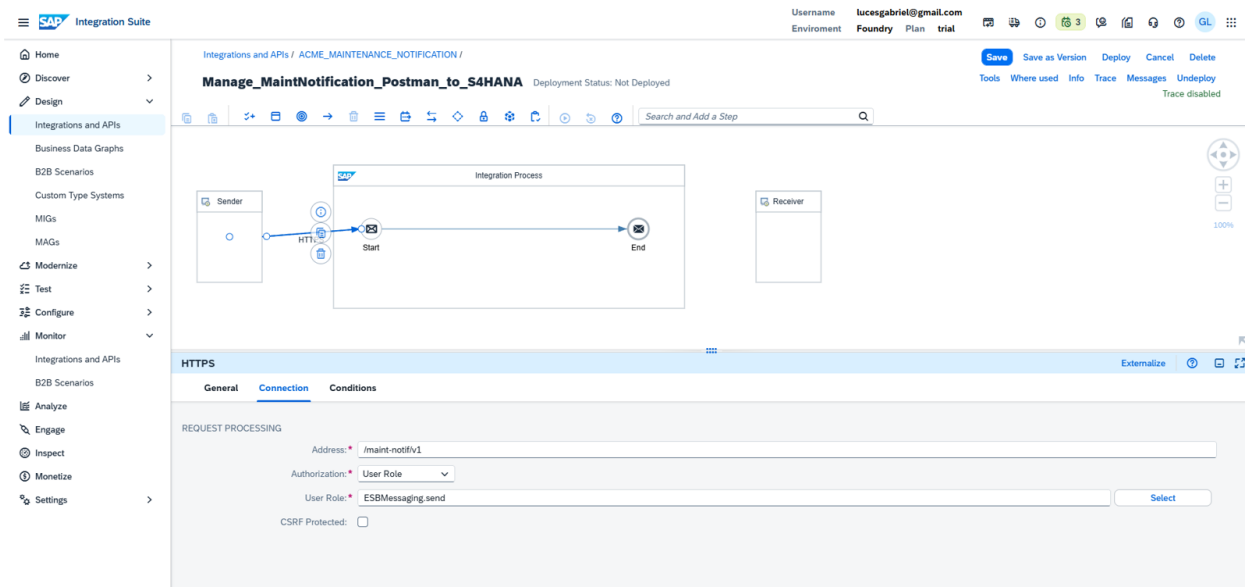


iFlow desplegado y en estado Started

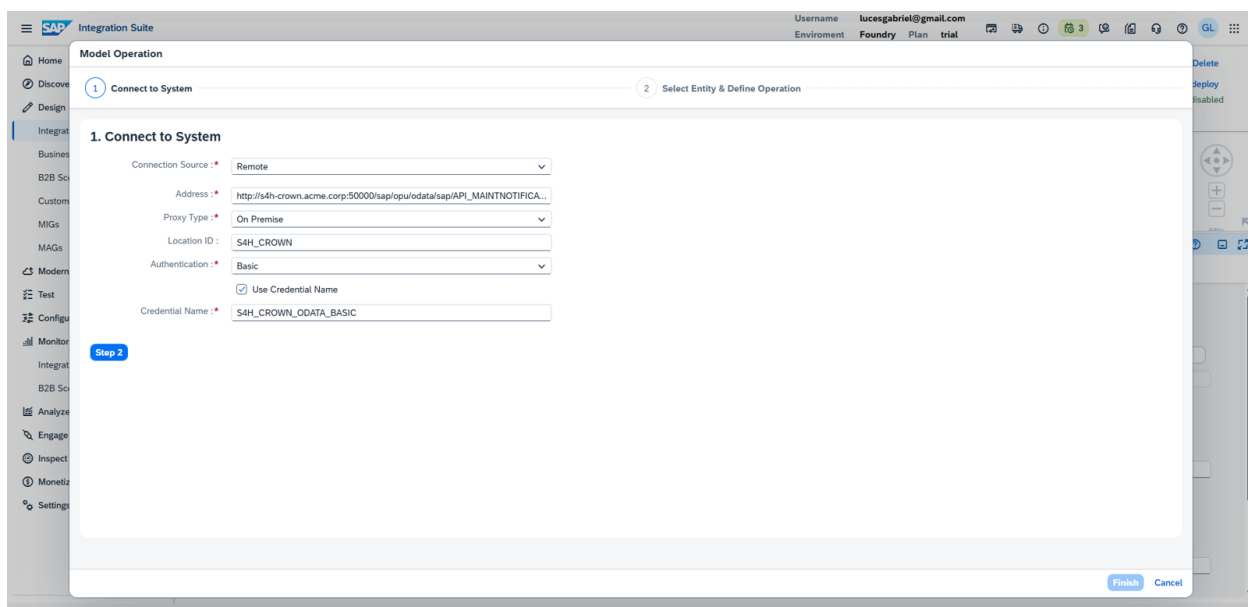
Rama create. El JSON del consumidor se convierte a XML, se leen cinco properties por XPath y un Content Modifier arma el body con la forma que espera el adapter: raíz = EntitySet, hijo = EntityType.

```
<MaintenanceNotification>
  <MaintenanceNotificationType>
    <NotificationType>${property.req_type}</NotificationType>
    <NotificationText>${property.req_text}</NotificationText>
    <MaintPriority>${property.req_priority}</MaintPriority>
    <TechnicalObject>${property.req_equipment}</TechnicalObject>
    <TechObjIsEquipOrFuncnlLoc>EAMS_EQUI</TechObjIsEquipOrFuncnlLoc>
    <ReportedByUser>${property.req_reportedBy}</ReportedByUser>
  </MaintenanceNotificationType>
</MaintenanceNotification>
```

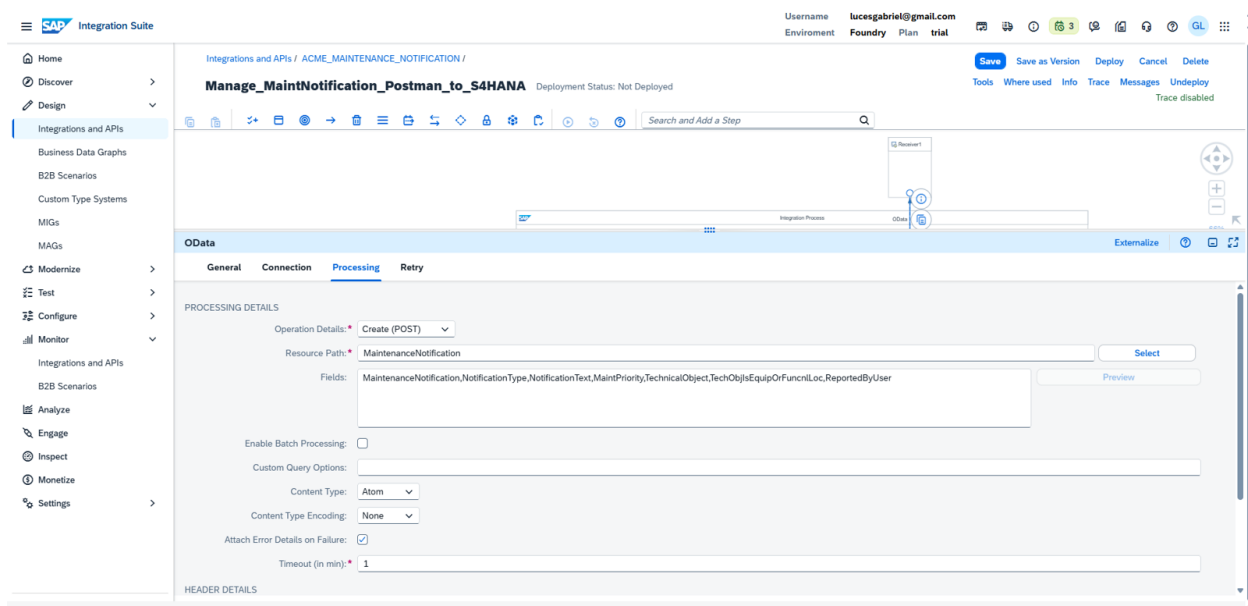
El receiver OD_S4H_NotifCreate (OData V2, Create (POST), Proxy Type = On-Premise, credencial por alias) se modela con el Query Modeler seleccionando solo esos seis campos.



HTTPS Sender



Query Modeler: conexión remota vía Cloud Connector

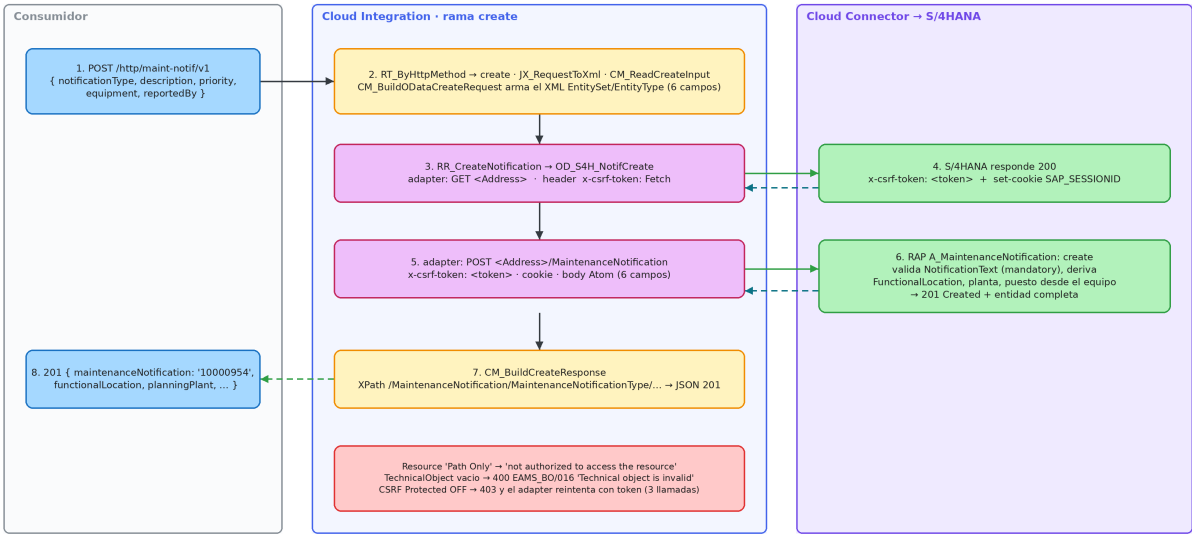


Receiver OData: Create (POST) con los 6 campos

Lo que hace el adapter por vos. Con CSRF Protected el adapter obtiene el token y lo reutiliza en el POST. Ambas llamadas cruzan el Cloud Connector.

Secuencia del Create: lo que hace el adapter OData V2 con el token CSRF

Por que el resource del Cloud Connector necesita Path And All Sub-Paths: dos llamadas al backend por cada aviso creado



Los pasos 3-6 los ejecuta el adapter solo: no hay Request Reply HTTP ni Content Modifier para el token. Ambas llamadas cruzan el Cloud Connector.

Secuencia del Create con CSRF

Rama query. Un Content Modifier toma el header notificationNumber, y el receiver OD_S4H_NotifQuery ejecuta Query (GET) con \$filter=MaintenanceNotification eq '{property.notificationNumber}'. Filtro y no acceso por clave: un aviso inexistente devuelve 200 con feed vacío, sin excepción, lo que deja el 404 en manos del iFlow en la siguiente iteración.

3.3 API Management: gobierno mínimo

Proxy api-maint-notif-v1 sobre el API Provider de Cloud Integration, base path /acme/notifications/v1, **sin resources** (así POST y GET pasan sin declarar nada). Tres policies:

Dónde	Policy	Qué hace
ProxyEndpoint · PreFlow	PL_VerifyApiKey	rechaza con 401 cualquier request sin apikey válida
TargetEndpoint · PreFlow	PL_KvmCpiAuth	lee usuario y password de un Key Value Map cifrado
TargetEndpoint · PreFlow	PL_BasicAuthToCpi	arma el Authorization: Basic hacia Cloud Integration

El consumidor nunca ve la credencial de CPI; el proxy tampoco la lleva en su XML.

SAP Integration Suite

Username: **lucgabriel@gmail.com**
Environment: **Foundry** Plan: **trial**

API Artifacts for api-maint-notif-v1 (Draft)

Policy Editor

Flows

- ProxyEndpoint
 - PreFlow: 1
 - PostFlow: 0
- TargetEndpoint: default
 - PreFlow: 2
 - PostFlow: 0
- Created Policies
 - PL_BasicAuthToCpi
 - PL_KvmCpiAuth
 - PL_VerifyApiKey
 - defaultRaiseFaultPolicy
- Scripts

Diagram: PL_VerifyApiKey

Condition String

```

1 <VerifyApiKey xmlns="http://www.sap.com/soapnet"
2   async="false" continueOnError="false" enabled="true">
3   <APIKey ref="request.header.apikey"/>
4 </VerifyApiKey>

```

Policies

- Security Policies
 - Basic Authentication
 - Decode.JWT
 - Generate.JWT
 - JSON Threat Protection
 - OAuth v2.0
 - OAuth v2.0 GET
 - OAuth v2.0 SET
 - Regular Expression Protection
 - SAML Assertion Generation
 - SAML Assertion Validation
 - Verify API Key
 - Verify.JWT
 - XML Threat Protection
- Traffic Management Policies
 - Access Control
 - Invalidate Cache
 - Lookup Cache
 - Populate Cache
 - Quota
 - Reset Quota
 - Response Cache
 - Spike Arrest
- Mediation Policies

Policy Verify API Key

SAP Integration Suite

Username: **lucgabriel@gmail.com**
Environment: **Foundry** Plan: **trial**

API Artifacts for api-maint-notif-v1 (Draft)

Policy Editor

Flows

- ProxyEndpoint
 - PreFlow: 1
 - PostFlow: 0
- TargetEndpoint: default
 - PreFlow: 2
 - PostFlow: 0
- Created Policies
 - PL_BasicAuthToCpi
 - PL_KvmCpiAuth
 - PL_VerifyApiKey
 - defaultRaiseFaultPolicy
- Scripts

Diagram: PL_KvmCpiAuth

Condition String

```

1 <KeyValueMapOperations xmlns="http://www.sap.com/soapnet"
2   mapIdentifier="KVM_CPI_Auth" async="false" continueOnError="false" enabled="true">
3   <get assignTo="private.cpiUser" index="1">
4     <key>
5       <parameter:username/>
6     </key>
7   </get>
8   <get assignTo="private.cpiPassword" index="1">
9     <key>
10      <parameter:password/>
11    </key>
12  </get>
13  <scope:environment/>
14 </KeyValueMapOperations>

```

Policies

- Security Policies
 - Basic Authentication
 - Decode.JWT
 - Generate.JWT
 - JSON Threat Protection
 - OAuth v2.0
 - OAuth v2.0 GET
 - OAuth v2.0 SET
 - Regular Expression Protection
 - SAML Assertion Generation
 - SAML Assertion Validation
 - Verify API Key
 - Verify.JWT
 - XML Threat Protection
- Traffic Management Policies
 - Access Control
 - Invalidate Cache
 - Lookup Cache
 - Populate Cache
 - Quota
 - Reset Quota
 - Response Cache
 - Spike Arrest
- Mediation Policies

Policy Key Value Map Operations

SAP Integration Suite

Username: **lucgabriel@gmail.com**
Environment: **Foundry** Plan: **trial**

API Artifacts for api-maint-notif-v1 (Draft)

Policy Editor

Flows

- ProxyEndpoint
 - PreFlow: 1
 - PostFlow: 0
- TargetEndpoint: default
 - PreFlow: 2
 - PostFlow: 0
- Created Policies
 - PL_BasicAuthToCpi
 - PL_KvmCpiAuth
 - PL_VerifyApiKey
 - defaultRaiseFaultPolicy
- Scripts

Diagram: PL_BasicAuthToCpi

Condition String

```

1 <BasicAuthentication xmlns="http://www.sap.com/soapnet"
2   async="false" continueOnError="false" enabled="true">
3   <operation:encode/>
4   <ignoreUnresolvedVariables>false/>
5   <user ref="private.cpiUser"/>
6   <password ref="private.cpiPassword"/>
7   <assignTo createNew="false" request.header.Authorization/>
8 </BasicAuthentication>

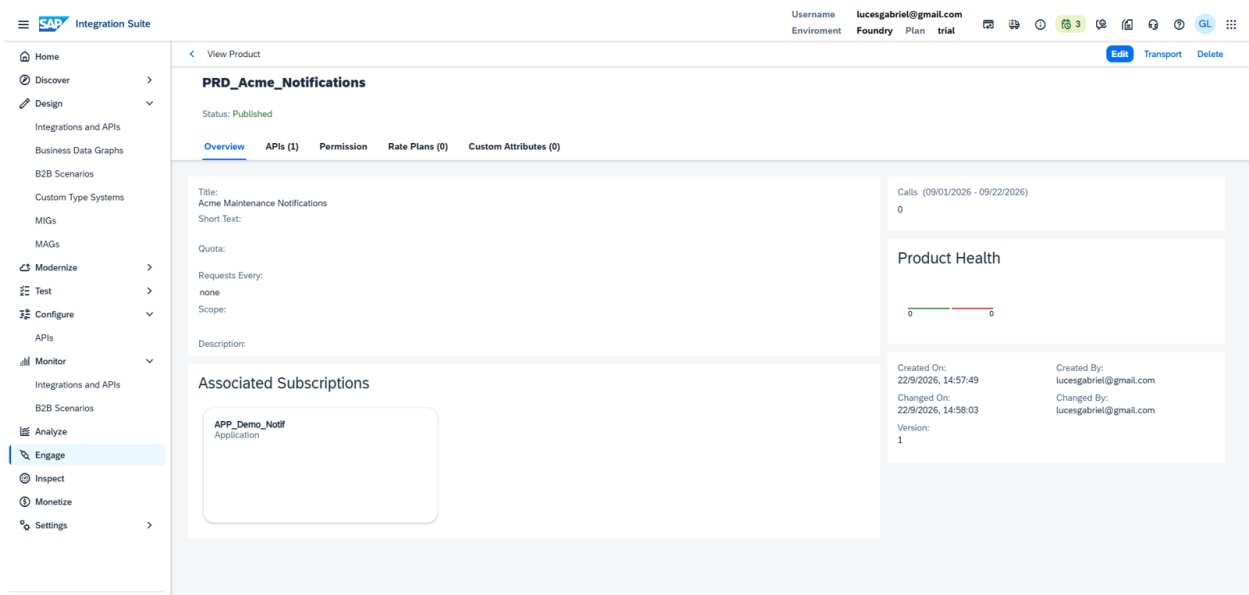
```

Policies

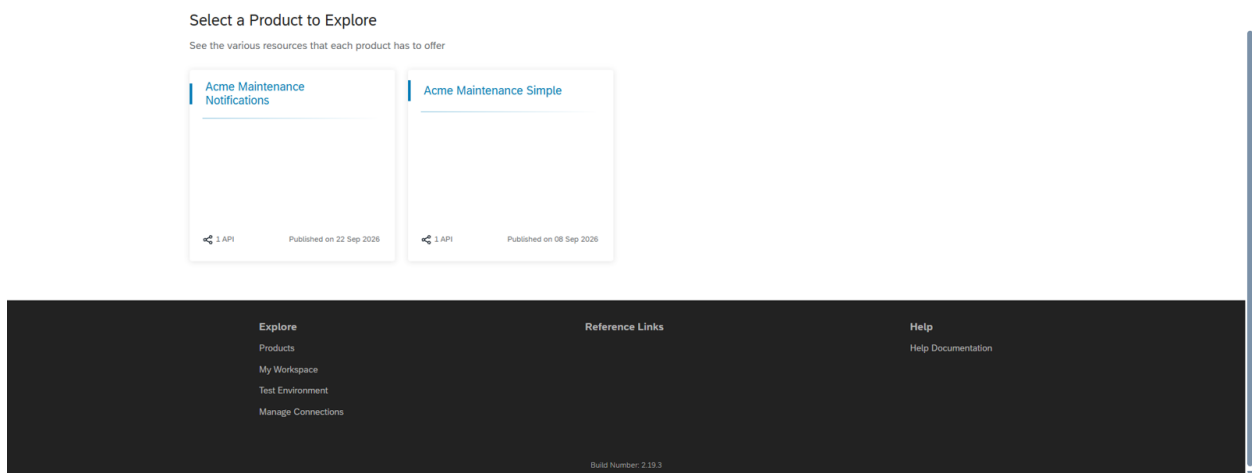
- Security Policies
 - Basic Authentication
 - Decode.JWT
 - Generate.JWT
 - JSON Threat Protection
 - OAuth v2.0
 - OAuth v2.0 GET
 - OAuth v2.0 SET
 - Regular Expression Protection
 - SAML Assertion Generation
 - SAML Assertion Validation
 - Verify API Key
 - Verify.JWT
 - XML Threat Protection
- Traffic Management Policies
 - Access Control
 - Invalidate Cache
 - Lookup Cache
 - Populate Cache
 - Quota
 - Reset Quota
 - Response Cache
 - Spike Arrest
- Mediation Policies

Policy Basic Authentication hacia CPI

Product PRD_Acme_Notifications publicado en el Developer Hub y una App suscrita: de ahí sale la apikey.



Product publicado



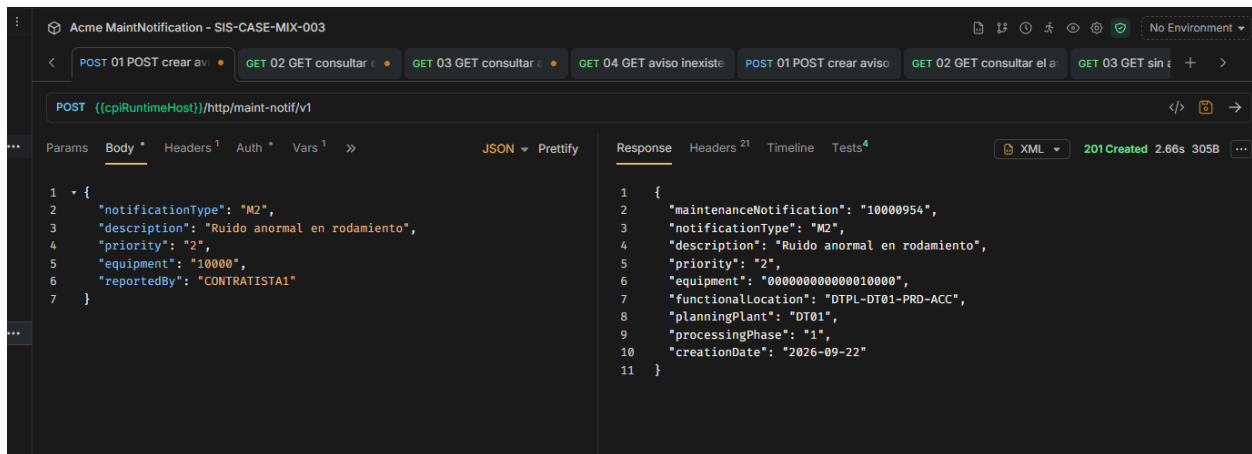
Developer Hub: catálogo de productos

4. Las pruebas

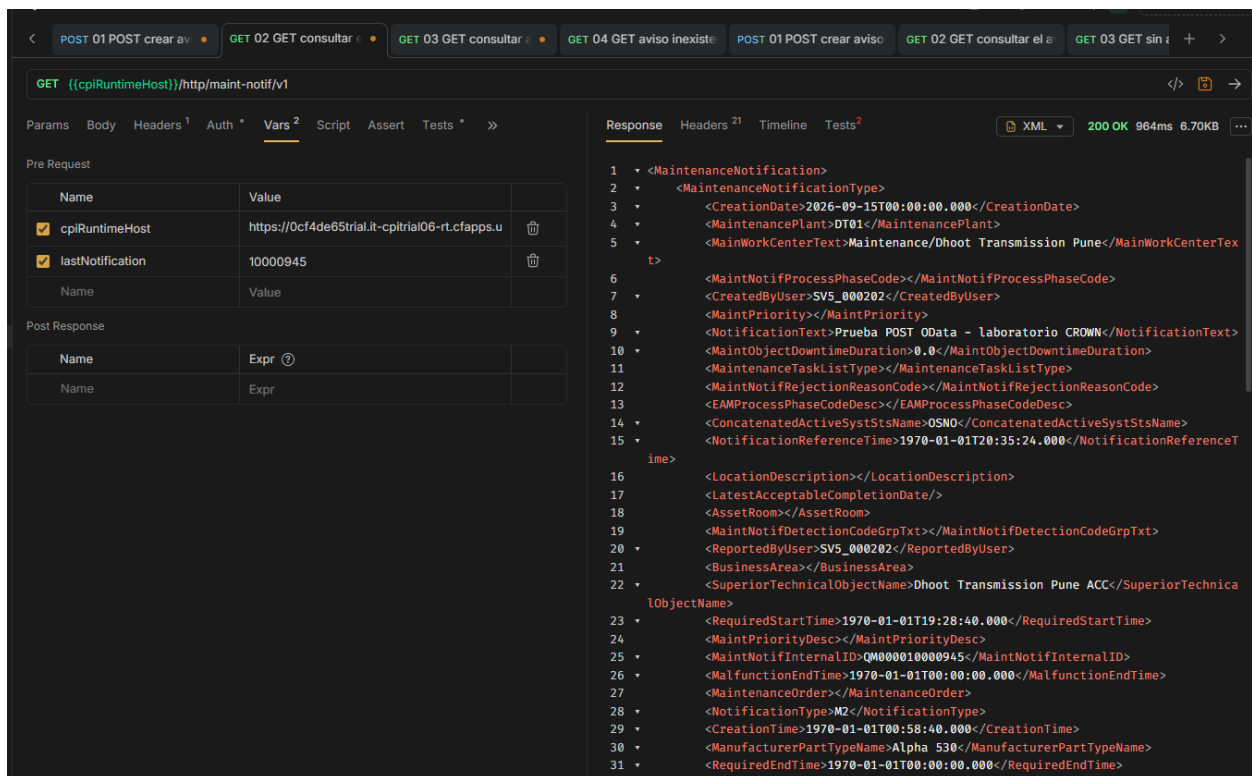
Colección Bruno con dos carpetas: 01 CPI directo (Basic con service key) y 02 APIM (solo apikey). El POST guarda el número devuelto en una variable y el GET siguiente lo consume.

Directo a Cloud Integration. POST → **201** con el aviso 10000954; la ubicación técnica

DTPL-DT01-PRD-ACC y la planta DT01 **no se enviaron**: las derivó S/4HANA desde el equipo.



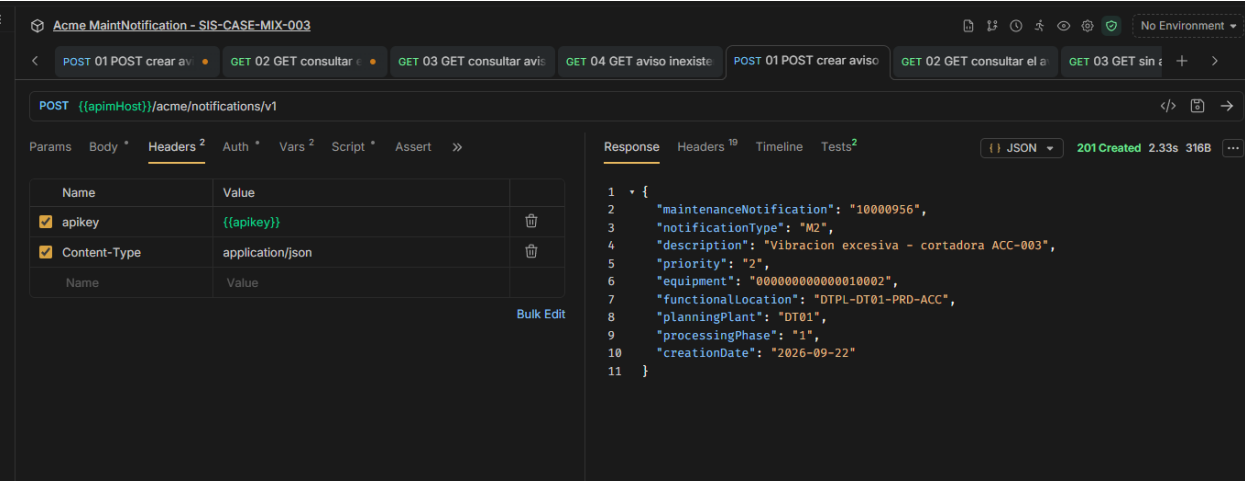
POST directo a CPI: 201



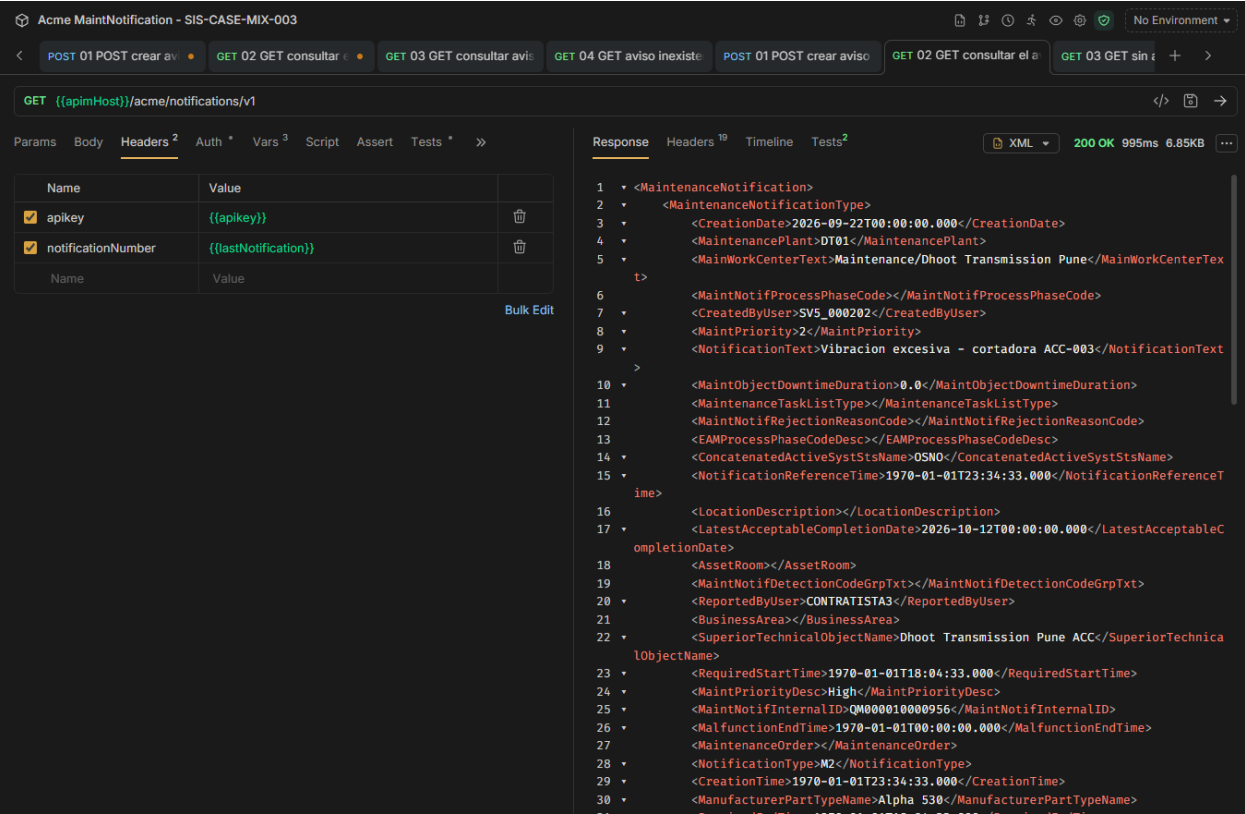
GET directo a CPI: 200

A través de API Management. Mismo resultado, sin Basic Auth en el cliente: POST → 201

con el aviso 10000956; GET → 200.



POST vía APIM: 201



GET vía APIM: 200

5. Lo que me rompió el build (y cómo lo resolví)

Síntoma	Causa real	Fix
JSON to XML Converter en rojo: <code>***cannot process the message type passed by element Http Sender Component***</code> , el Save falla	la ruta create del Router quedó con Expression Type = XML (valor por defecto) y el modelador infirió mensaje XML por esa rama	Non-XML + <code>\$(header.CamelHttpMethod) = 'POST'</code>
POST → 500; el backend responde <code>EAMS_BO/016 Technical object is invalid (con &1 vacío)</code>	typo en el nombre de una Exchange Property (<code>req equipmen</code> vs <code>\$(property.req_equipment)</code>): la property no existía y SAP recibió <code>TechnicalObject</code> en blanco	corregir el nombre. En el MPL con Trace, <code>setProperty[...]</code> muestra los nombres reales
El Query Modeler abre en <code>*Local EDMX File*</code> y dice <code>***Unable to get edmx file content***</code>	EDMX huérfano de un intento anterior	Connection Source = Remote

Síntoma	Causa real	Fix
El create funcionaba con CSRF Protected desmarcado	el adapter OData V2 (1.30.2) recibe el 403, obtiene el token del service document y repite el POST solo	marcar la casilla igual: pasa de 3 llamadas a 2 por aviso

Y una lección de diseño: **un 201 no es prueba de que el dato quedó bien**. El primer aviso que creé por API salió sin prioridad ni reportante. Verificar en el backend forma parte de la prueba.

6. Lo que queda para la siguiente iteración

Validación del JSON de entrada (400), Router de existencia en el GET (404 en vez de 200 vacío), Exception Subprocess con contrato de error {code, message, correlationId} y 502 cuando el Cloud Connector no responde, respuesta del GET mapeada a JSON plano, parámetros externalizados, y en API Management: Spike Arrest, Quota, eliminar la apikey antes del backend y Fault Rules.

Herramientas: SAP Integration Suite Trial (Cloud Integration, API Management, Developer Hub), SAP Cloud Connector, SAP S/4HANA 2025 on-premise, Bruno. Los datos técnicos provienen de un sistema de laboratorio; la empresa es ficticia. Sin credenciales ni hosts internos en este documento.