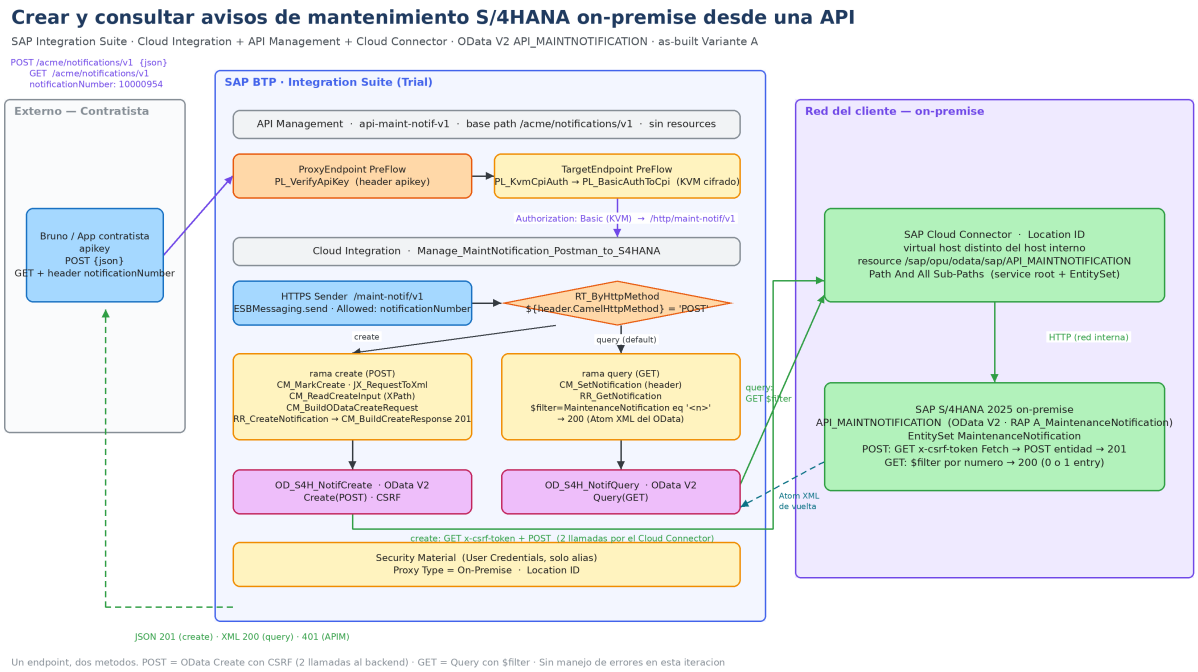
Crear y consultar avisos de mantenimiento en S/4HANA on-premise desde una API

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.



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
apikey: <clave de la app>
Content-Type: application/json

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

```
GET /acme/notifications/v1
apikey: <clave de la app>
notificationNumber: 10000954
```

---

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:

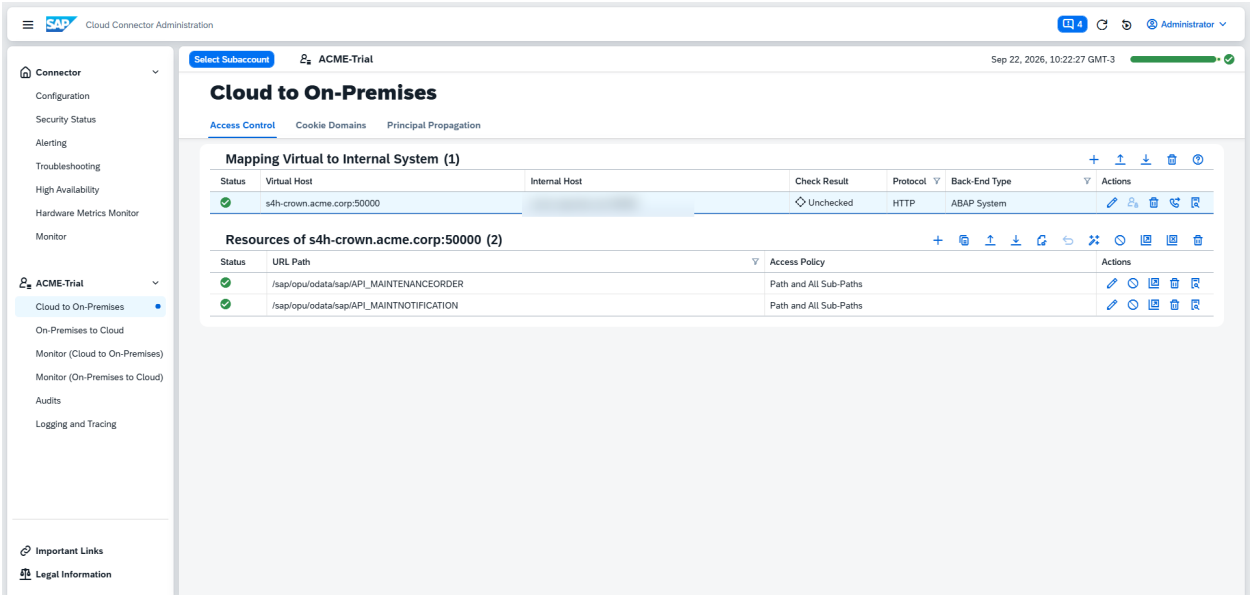
Hallazgo	Consecuencia
API_MAINTNOTIFICATION es una proyección **RAP** (A_MaintenanceNotification) expuesta por SADL como OData V2	el Create exige el ciclo **CSRF** completo; los números viajan sin ceros a la izquierda
En el Behavior Definition, FunctionalLocation es **readonly** y NotificationText es el único campo mandatory	el equipo se envía en TechnicalObject + TechObjIsEquipOrFuncnlLoc = 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

---

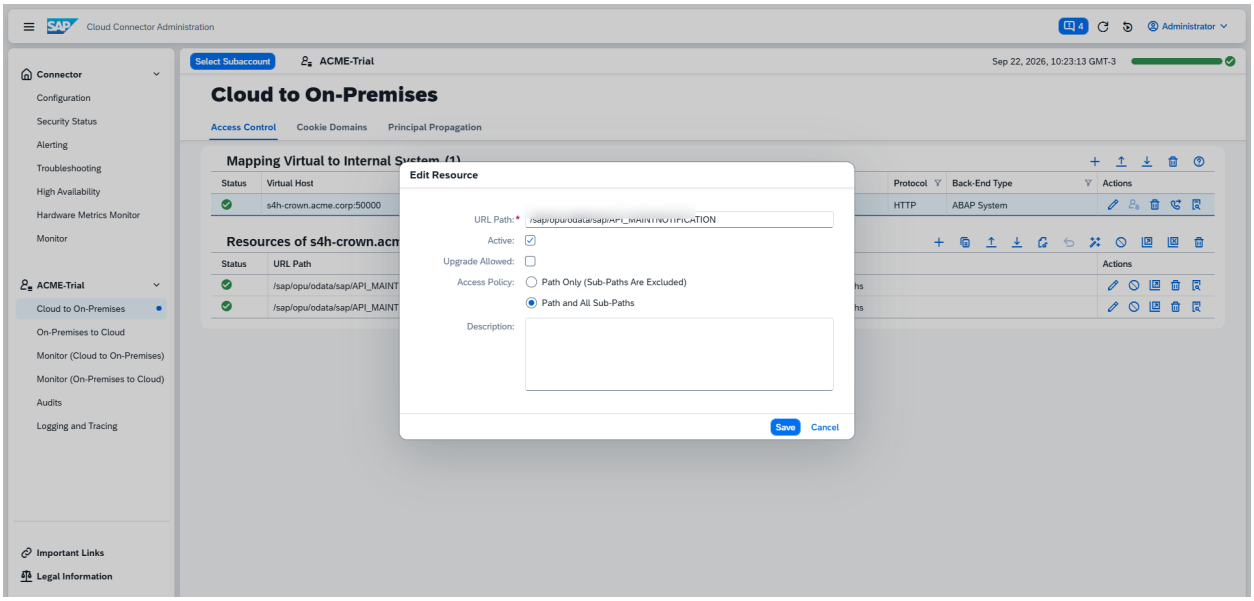
3. La solución

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.



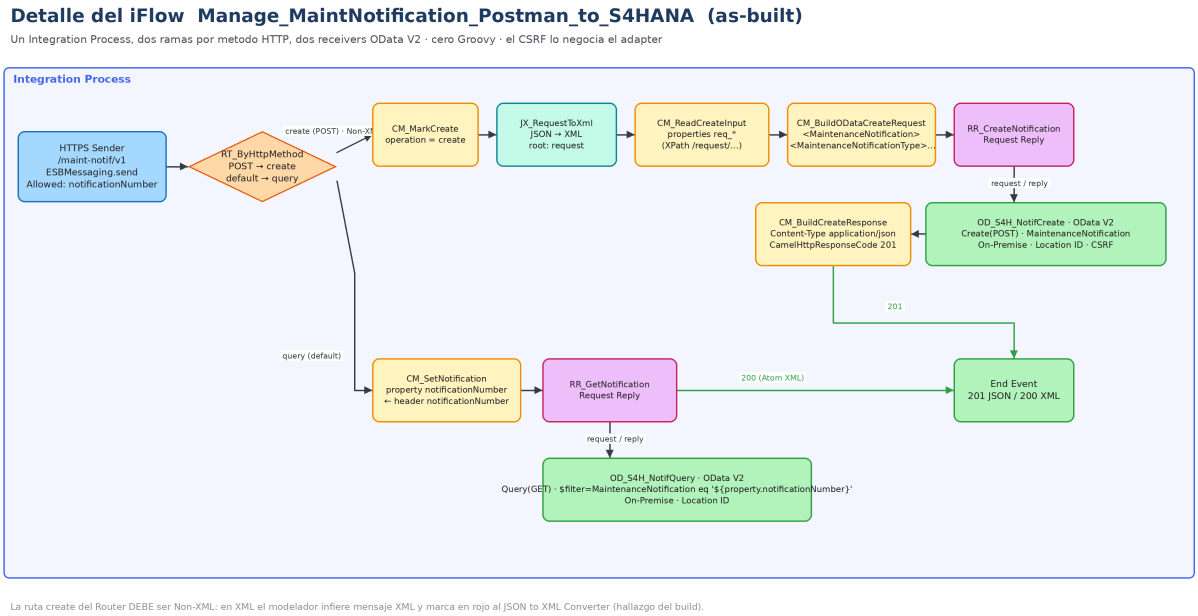
Cloud Connector: mapping y resources



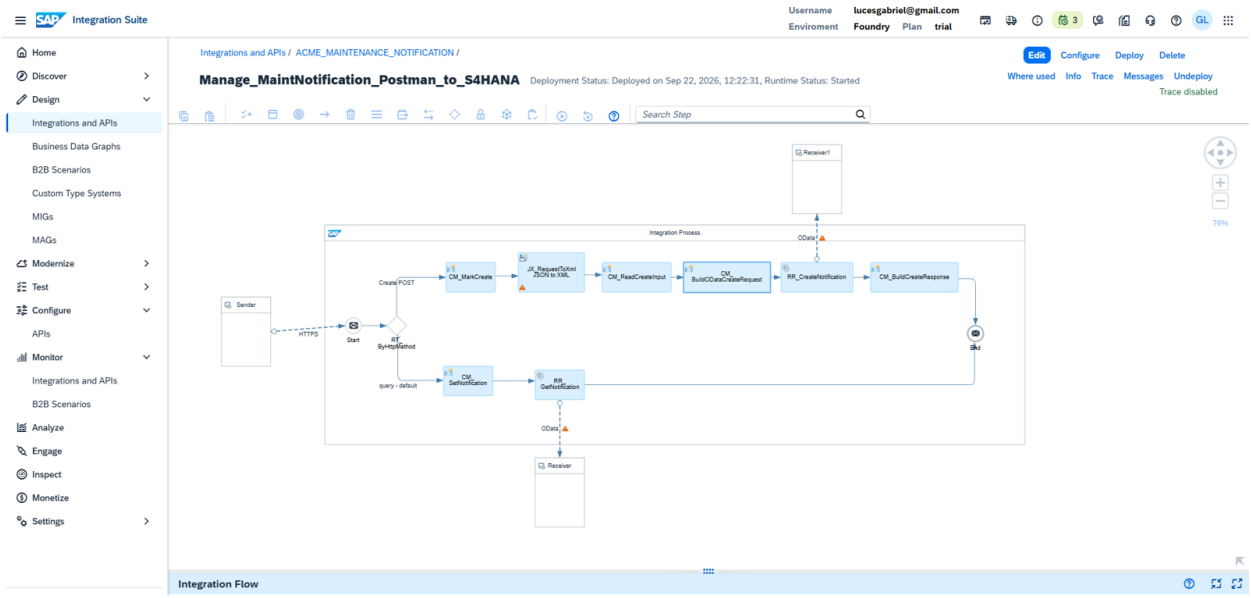
Resource con Path And All Sub-Paths

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

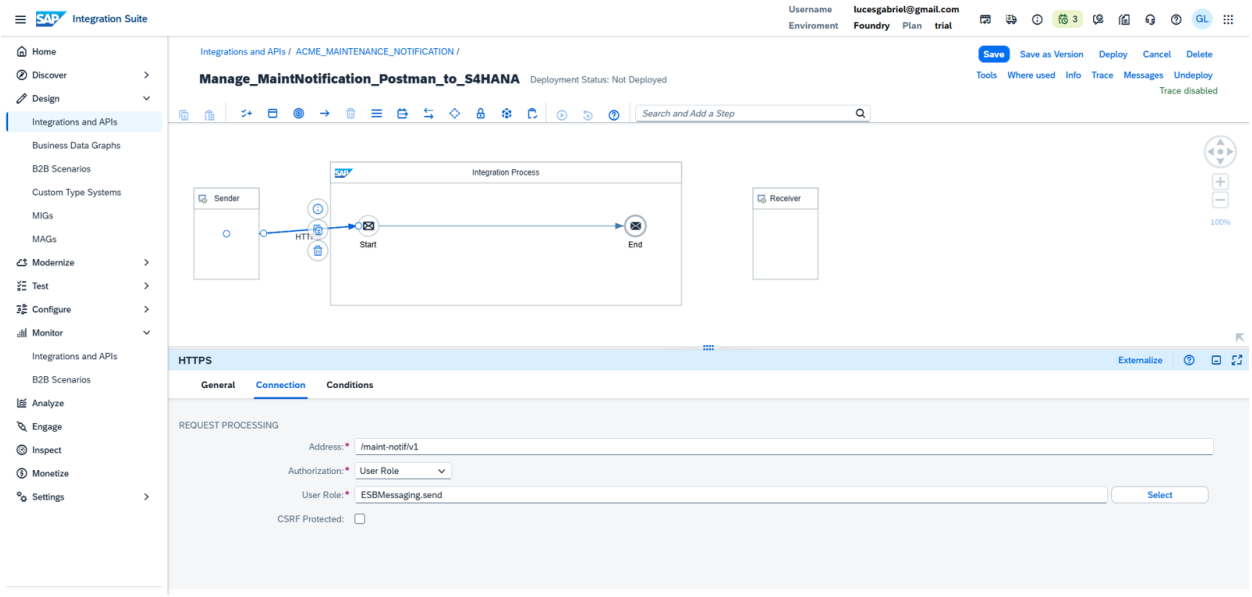


iFlow desplegado y en estado Started

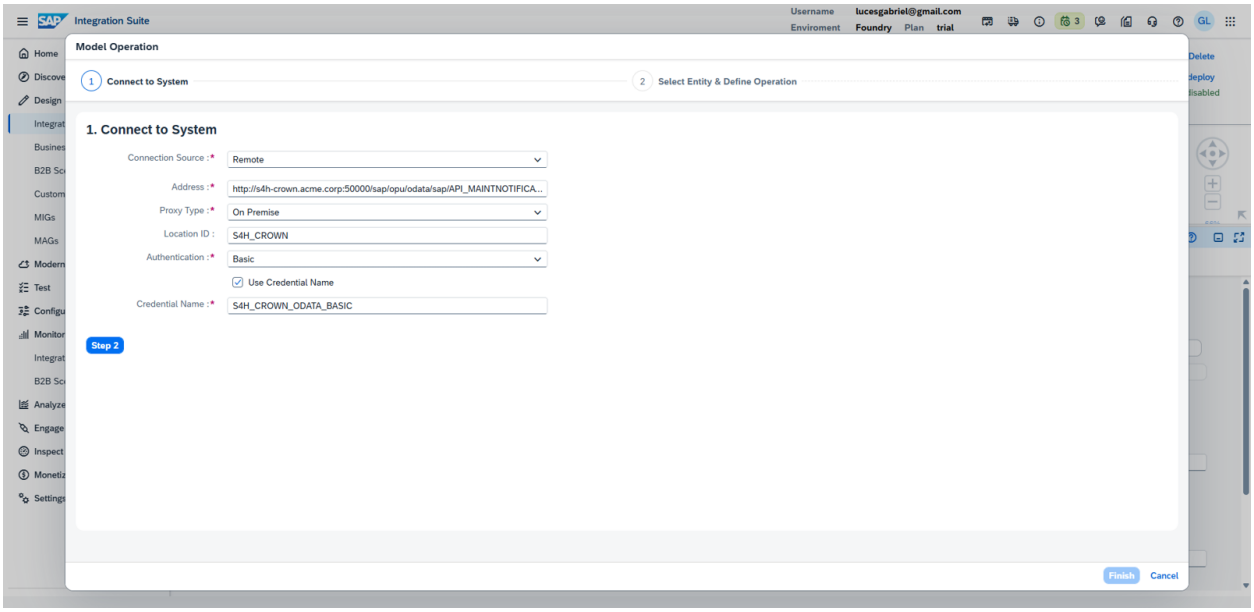
Rama create. El JSON del consumidor se convierte a XML, se leen cinco properties por XPath y un Content Modifier arma el body con la forma que espera el adapter: raíz = EntitySet, hijo = EntityType.

```
<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>
```

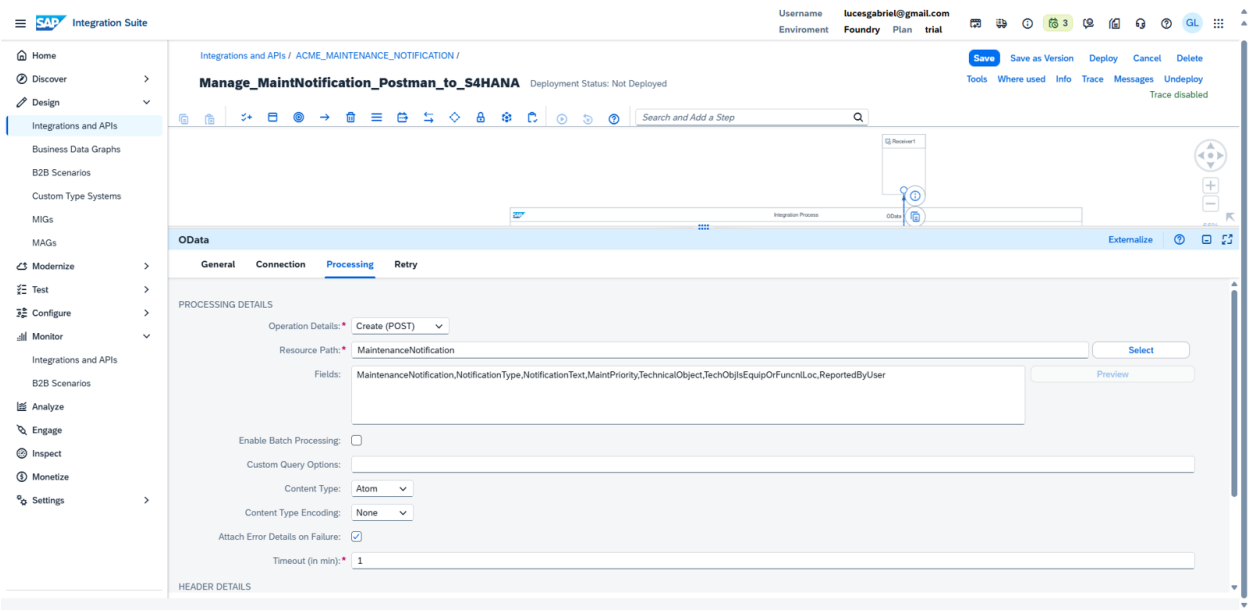
El receiver OD_S4H_NotifCreate (OData V2, Create (POST), Proxy Type = On-Premise, credencial por alias) se modela con el Query Modeler seleccionando solo esos seis campos.



HTTPS Sender

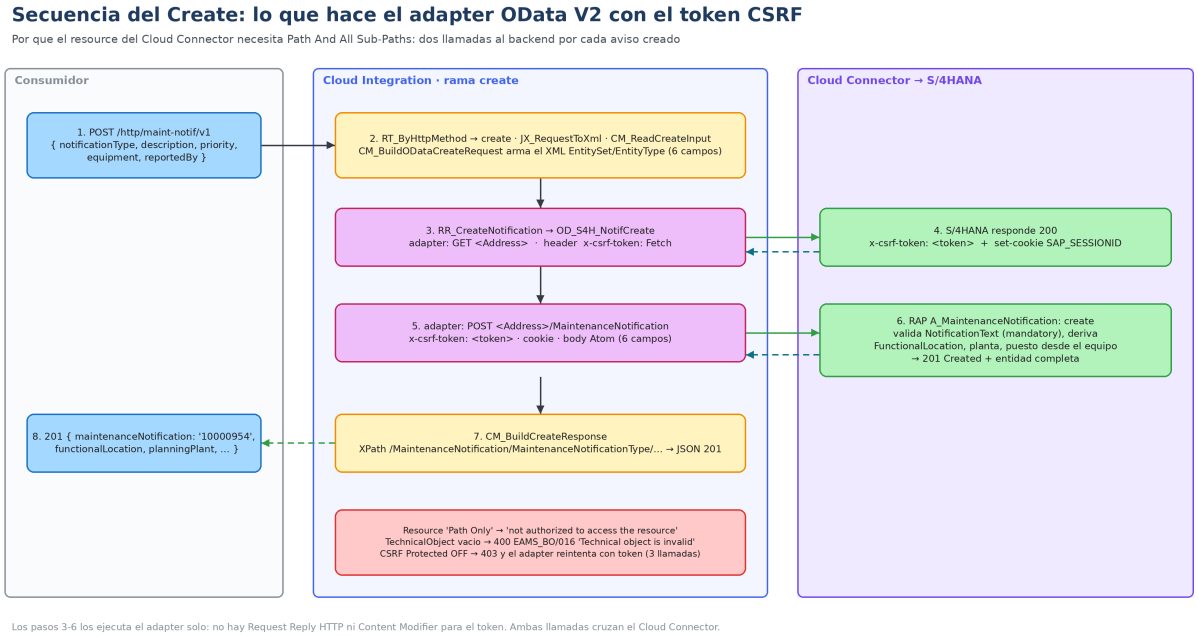


Query Modeler: conexión remota vía Cloud Connector



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 con CSRF

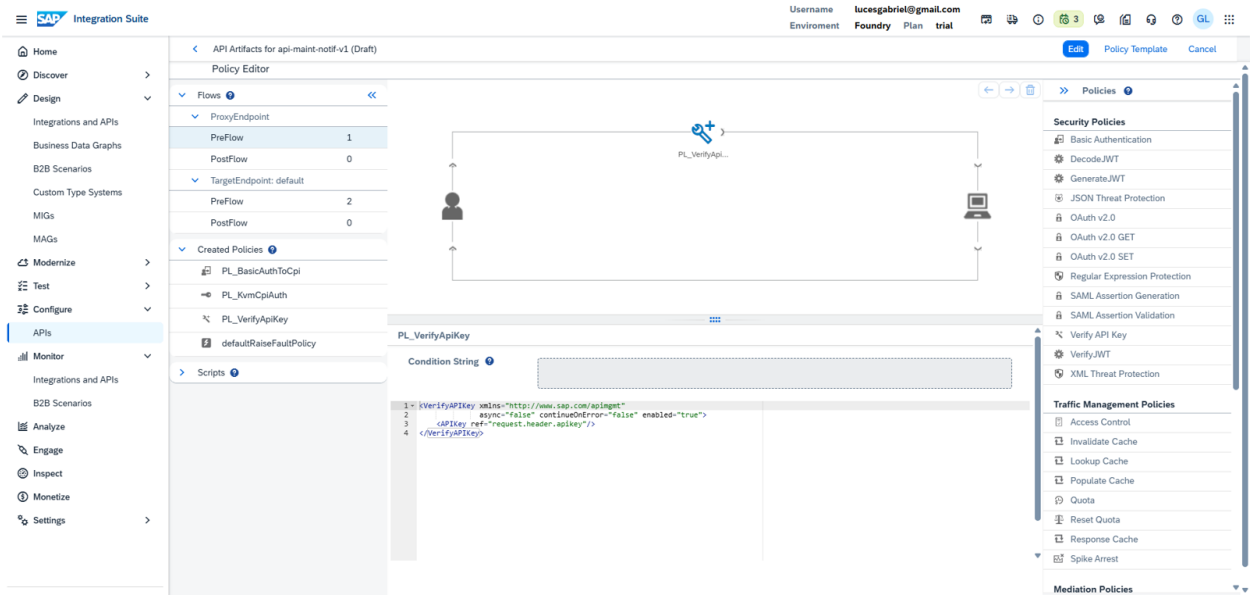
Rama query. Un Content Modifier toma el header notificationNumber, y el receiver OD_S4H_NotifQuery ejecuta Query (GET) con $filter=MaintenanceNotification eq '${property.notificationNumber}'. Filtro y no acceso por clave: un aviso inexistente devuelve 200 con feed vacío, sin excepción, lo que deja el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

3.3 API Management: gobierno mínimo

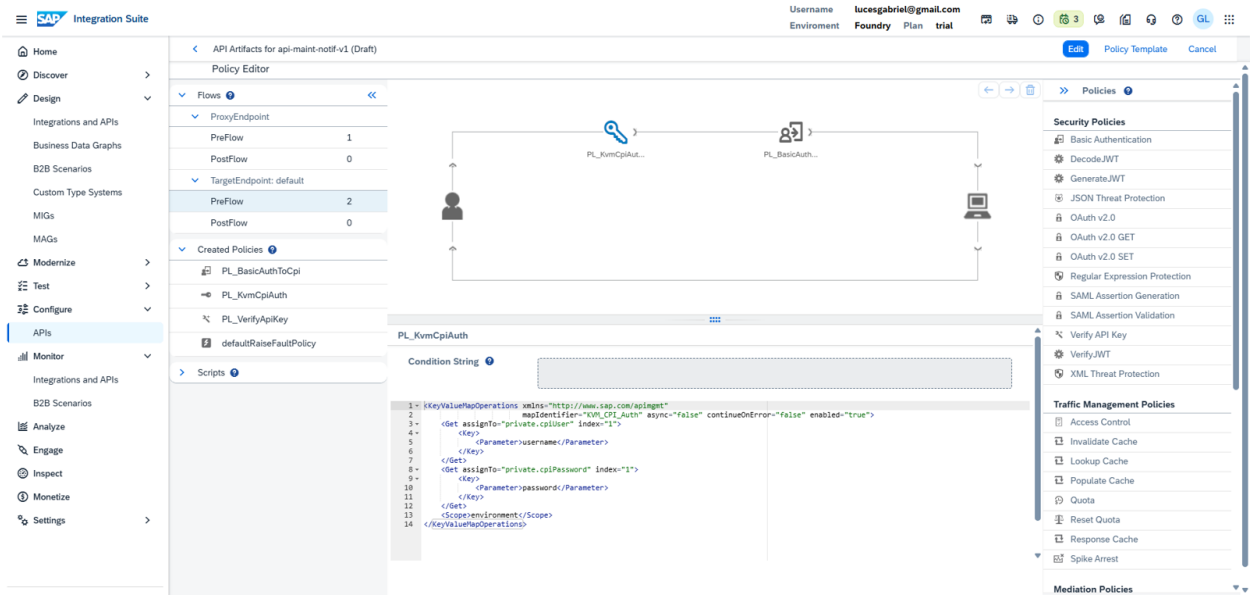
Proxy api-maint-notif-v1 sobre el API Provider de Cloud Integration, base path /acme/notifications/v1, **sin resources** (así POST y GET pasan sin declarar nada). Tres policies:

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

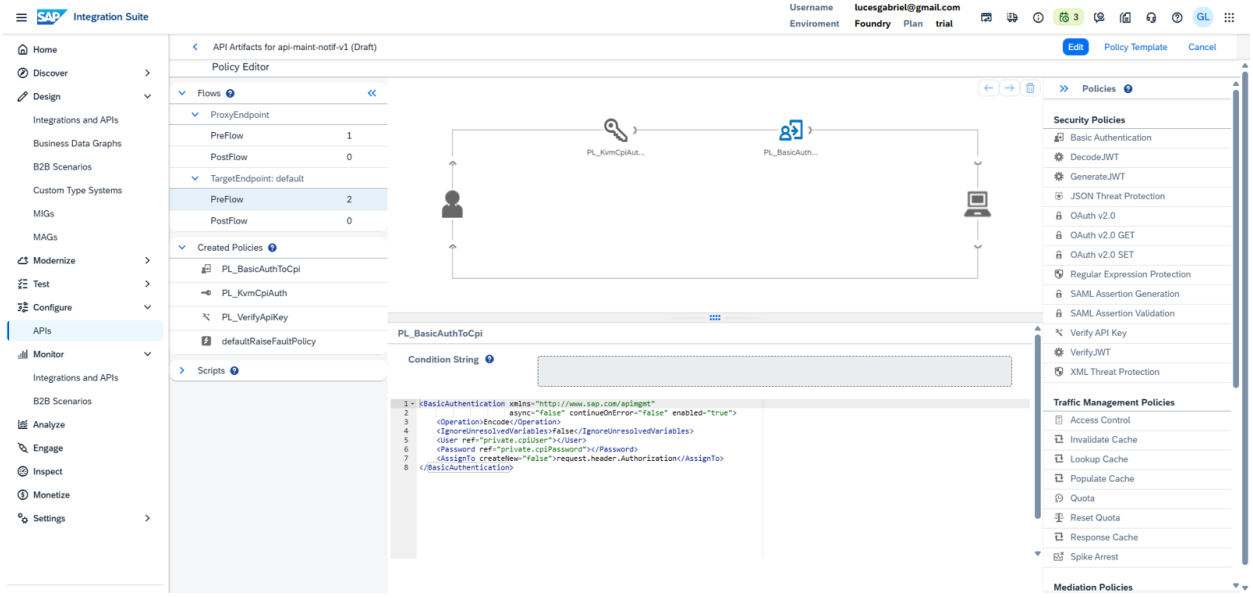
El consumidor nunca ve la credencial de CPI; el proxy tampoco la lleva en su XML.



Policy Verify API Key

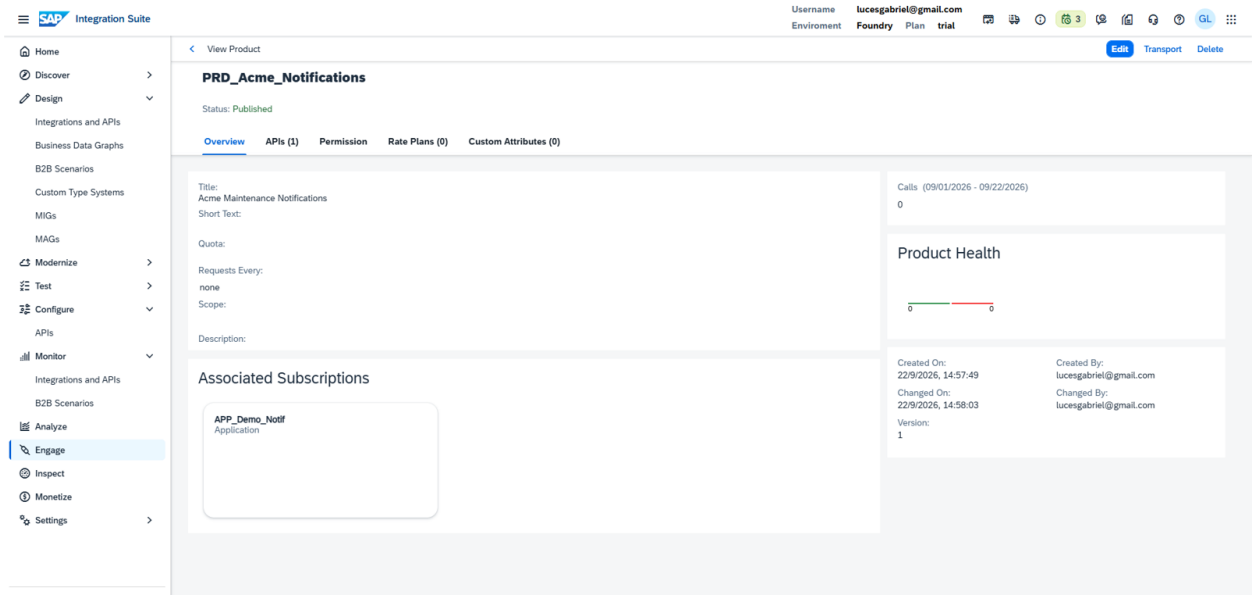


Policy Key Value Map Operations

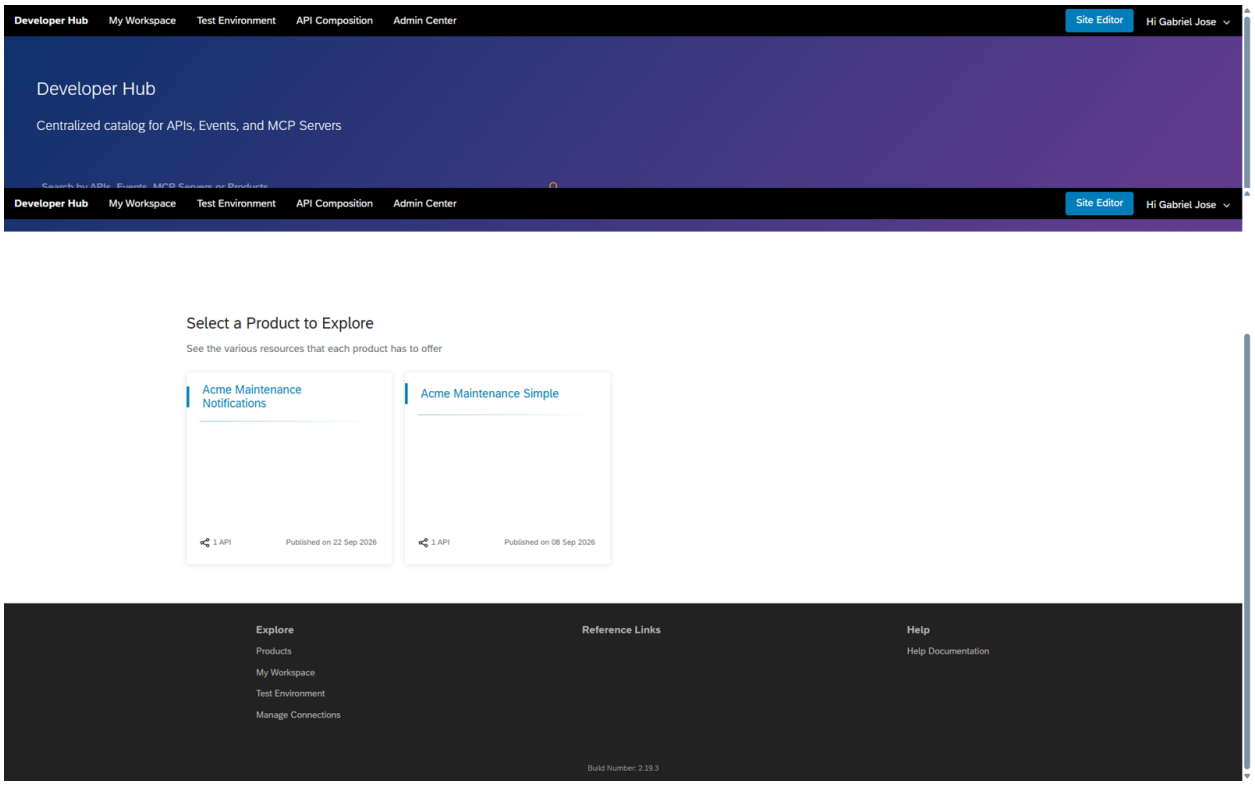


Policy Basic Authentication hacia CPI

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

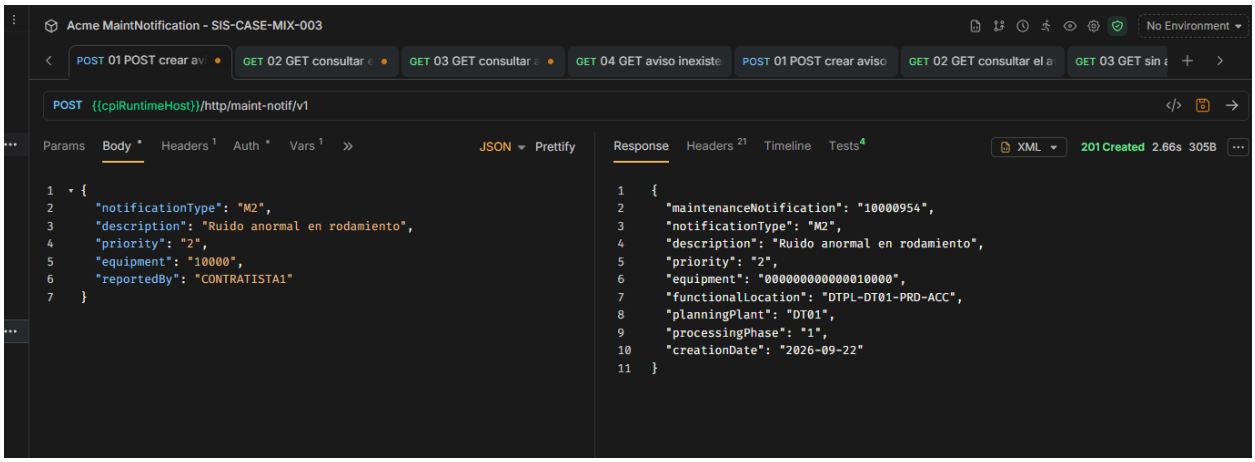
---

4. Las pruebas

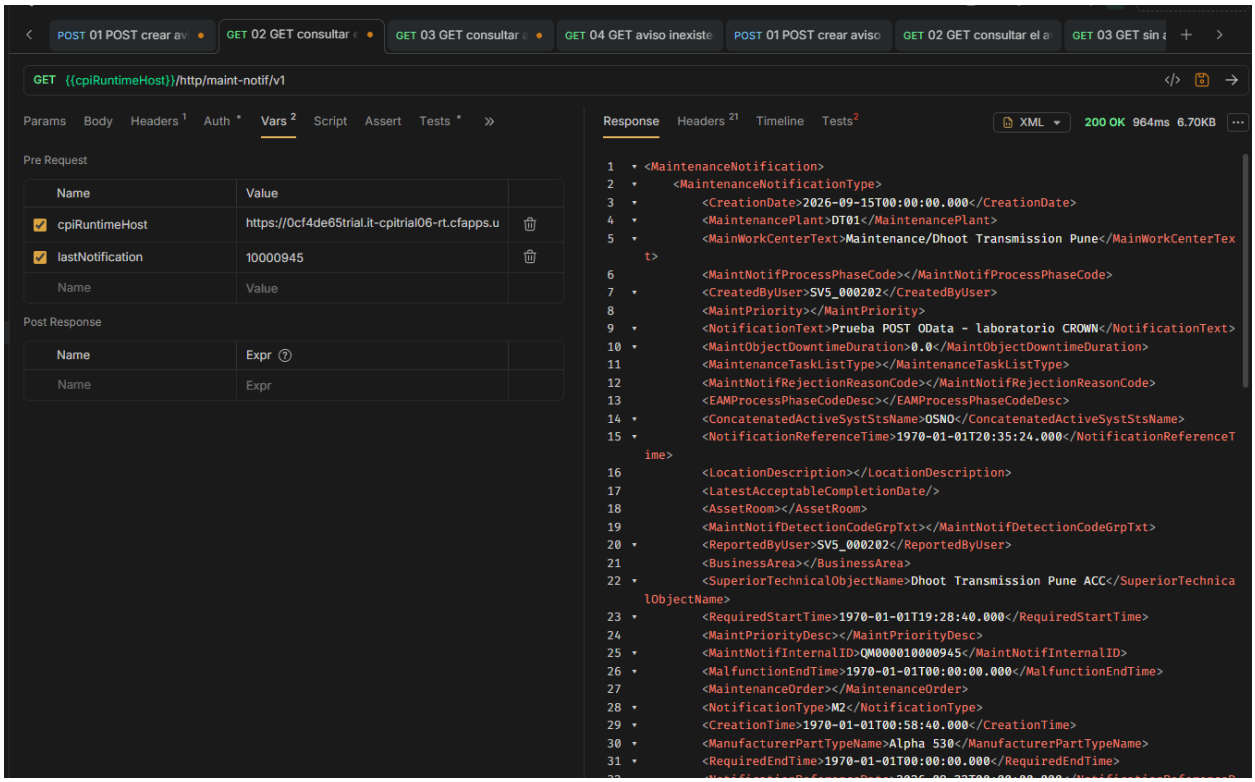
Colección Bruno con dos carpetas: 01 CPI directo (Basic con service key) y 02 APIM (solo apikey). El POST guarda el número devuelto en una variable y el GET siguiente lo consume.

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.

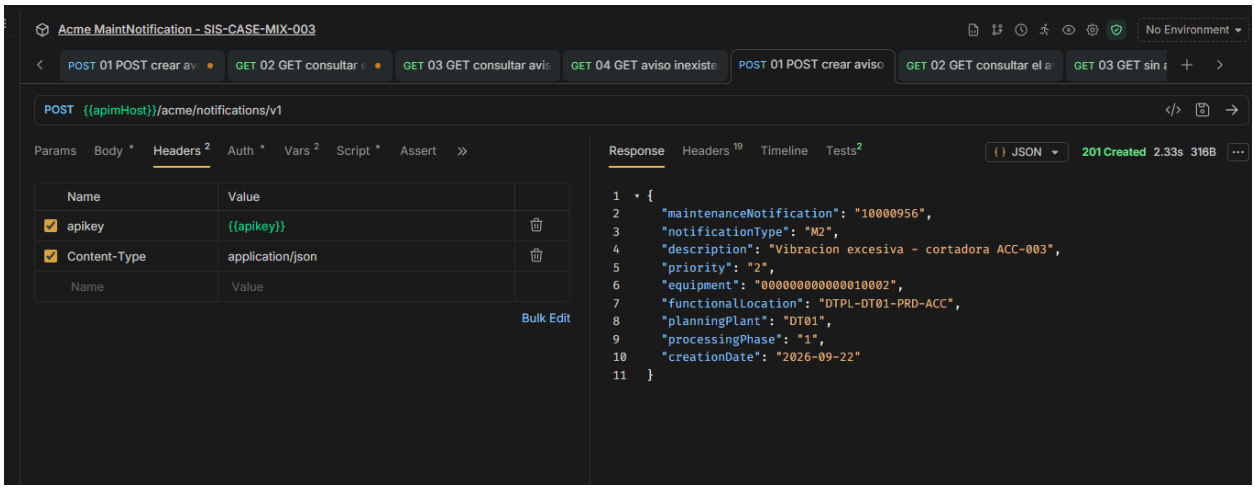


POST directo a CPI: 201

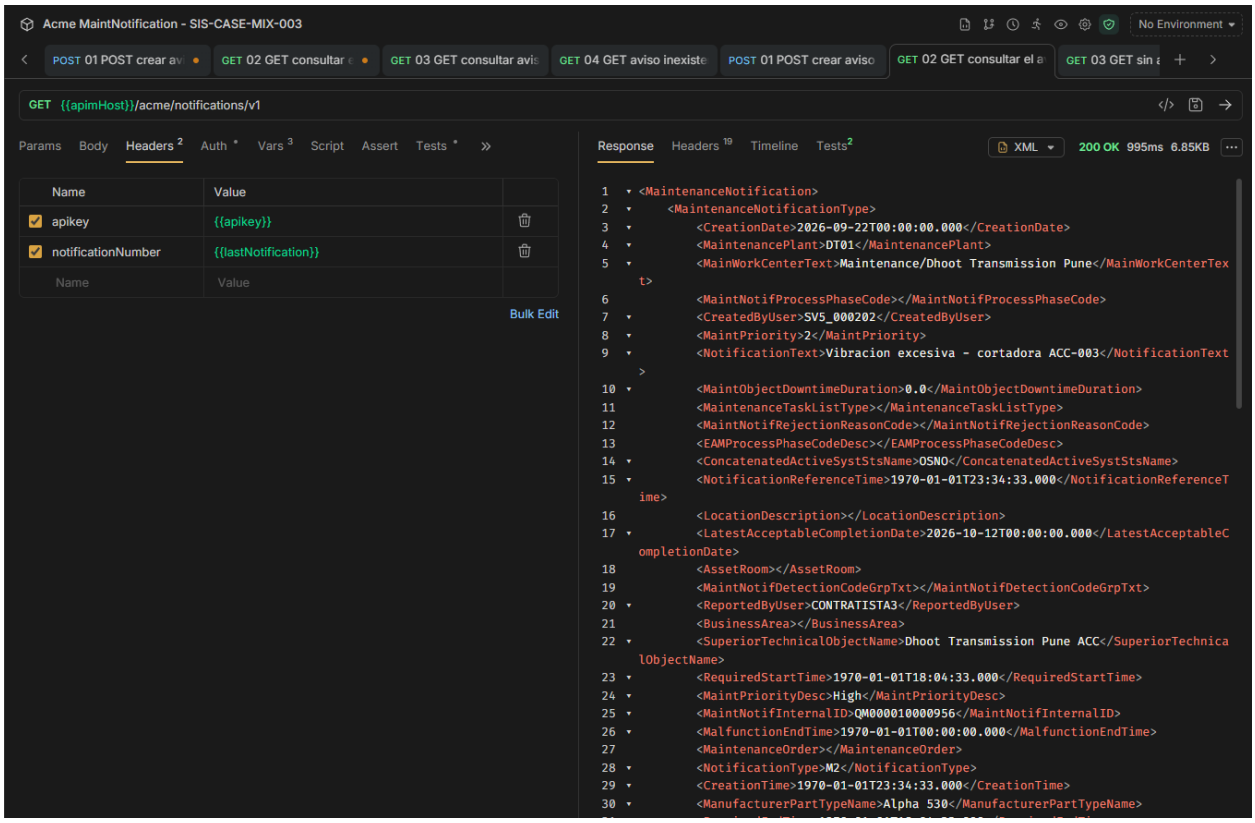


GET directo a CPI: 200

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: *"cannot process the message type passed by element Http Sender Component"*, 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 + ${header.CamelHttpMethod} = 'POST'
POST → 500; el backend responde EAMS_BO/016 Technical object is invalid (con &1 vacío)	typo en el nombre de una Exchange Property (req_equipmen vs ${property.req_equipment}): la property no existía y SAP recibió TechnicalObject en blanco	corregir el nombre. En el MPL con Trace, setProperty[...] muestra los nombres reales
El Query Modeler abre en *Local EDMX File* y dice *"Unable to get edmx file content"*	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.