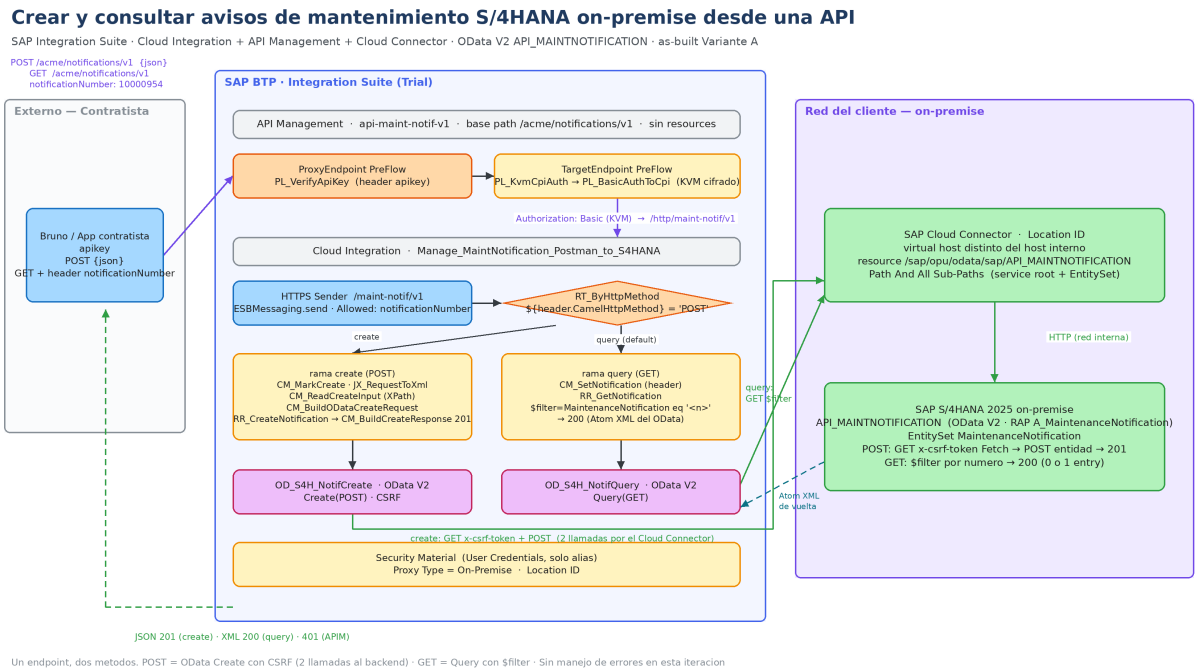
Creating and querying maintenance notifications in on-premise S/4HANA through an API

SAP Integration Suite · Cloud Integration + API Management + Cloud Connector · Hands-on case MIX-003

One REST endpoint: POST **creates** a malfunction notification in S/4HANA (OData V2 Create with CSRF token, through Cloud Connector) and GET **retrieves** a notification by number. A single iFlow with a Router on the HTTP method, governed by API Management with an API Key. Built and tested end to end on a Trial tenant against a real on-premise S/4HANA 2025 system.



End-to-end architecture

---

1. The problem

Context (fictional company). Acme Industrial Services maintains plant DT01. In the previous case (MIX-002) external contractors got the ability to **read** maintenance orders from their mobile app. The next pain point ran the other way: when a technician spots a malfunction in the field, it gets written on paper and dictated over the radio to a planner, who types it into SAP hours later. Data gets lost (which equipment, what priority, who reported it) and notifications end up **without a technical object**, which later blocks order planning.

What Maintenance asked for. Two operations behind one API:

1. Create a malfunction notification (M2) against a specific **piece of equipment**, with a description, a priority and the reporter, and get back the **notification number** SAP assigned.
2. Query a notification by number to confirm it was recorded.

What Security required. Nothing reaches the Cloud Integration runtime or S/4HANA directly; S/4HANA is **not exposed to the internet** (everything goes through Cloud Connector); each consumer authenticates with its own API Key.

Consumer contract.

```
POST /acme/notifications/v1
apikey: <application key>
Content-Type: application/json

{
  "notificationType": "M2",
  "description": "Abnormal bearing noise",
  "priority": "2",
  "equipment": "10000",
  "reportedBy": "CONTRACTOR1"
}
```

```
GET /acme/notifications/v1
apikey: <application key>
notificationNumber: 10000954
```

---

2. Before designing: look at the backend

Before drawing a single iFlow step I checked the service on the S/4HANA side. Three findings shaped the whole design:

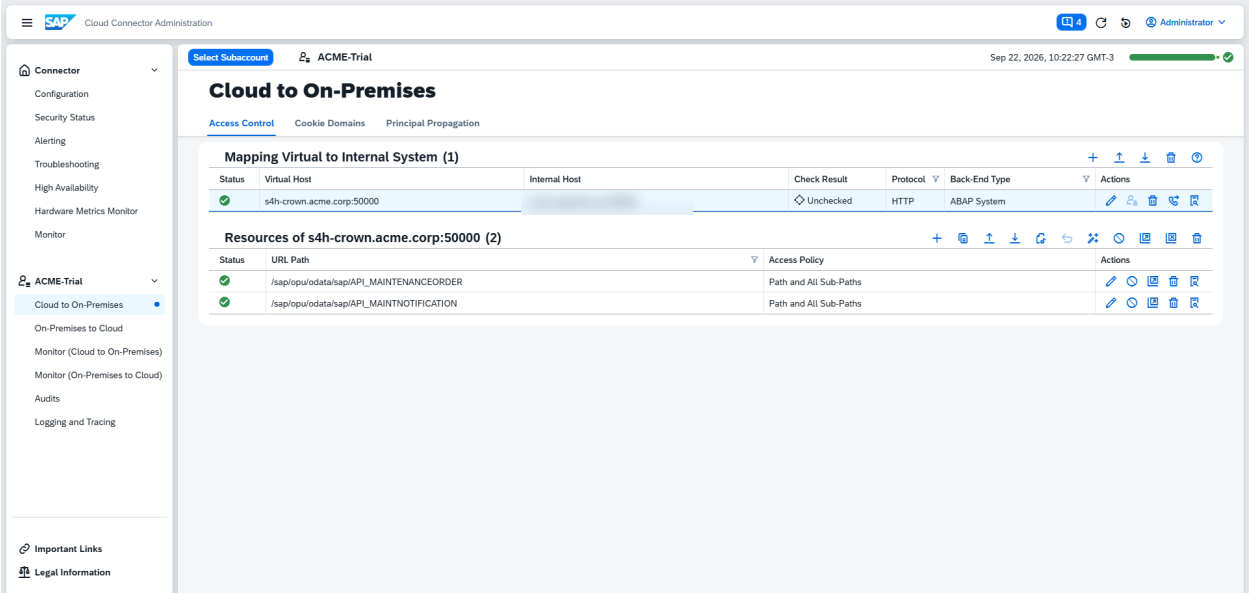
Finding	Consequence
API_MAINTNOTIFICATION is a **RAP** projection (A_MaintenanceNotification) exposed as OData V2 through SADL	Create requires the full **CSRF** handshake; numbers travel without leading zeros
In the Behavior Definition, FunctionalLocation is **read-only** and NotificationText is the only mandatory field	equipment is sent as TechnicalObject + TechObjIsEquipOrFuncnlLoc = EAMS_EQUI; functional location, plant and work center are **derived** by the backend
A POST with only type + text creates an **orphan** notification (no equipment, no plant)	the consumer contract makes equipment mandatory; a 201 does not prove the data is right

---

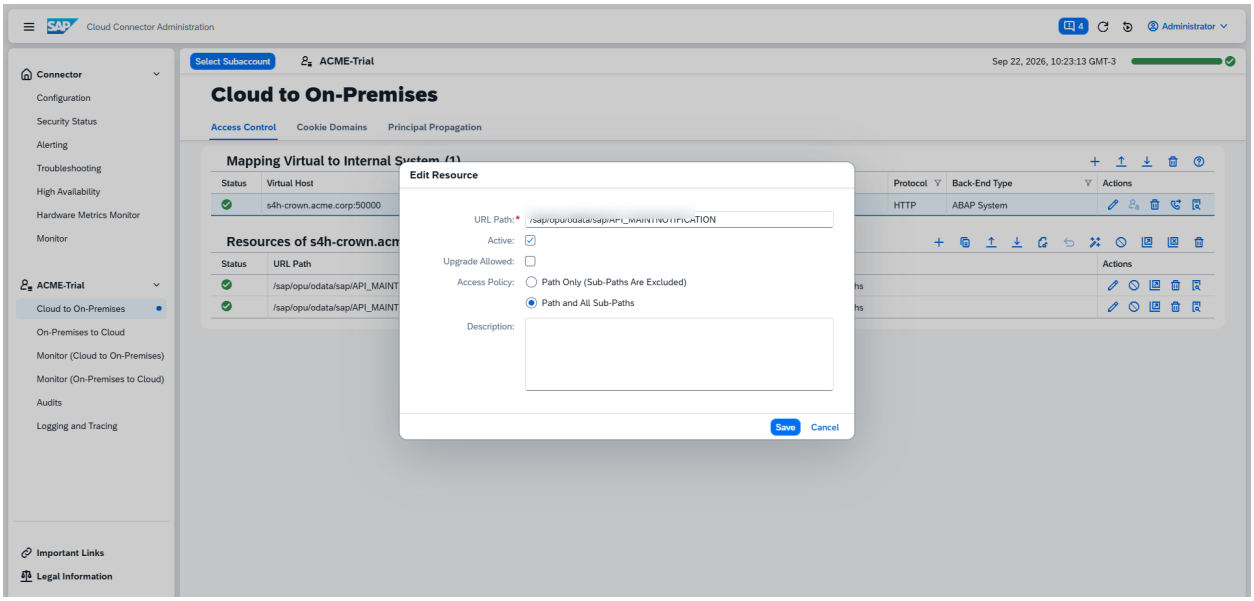
3. The solution

3.1 Cloud Connector: one new resource

On the existing mapping to S/4HANA (virtual host different from the internal one, Location ID S4H_CROWN), publish /sap/opu/odata/sap/API_MAINTNOTIFICATION with **Path And All Sub-Paths**. This is mandatory: Create makes two calls, a GET to the service root for the token and the POST to the EntitySet.



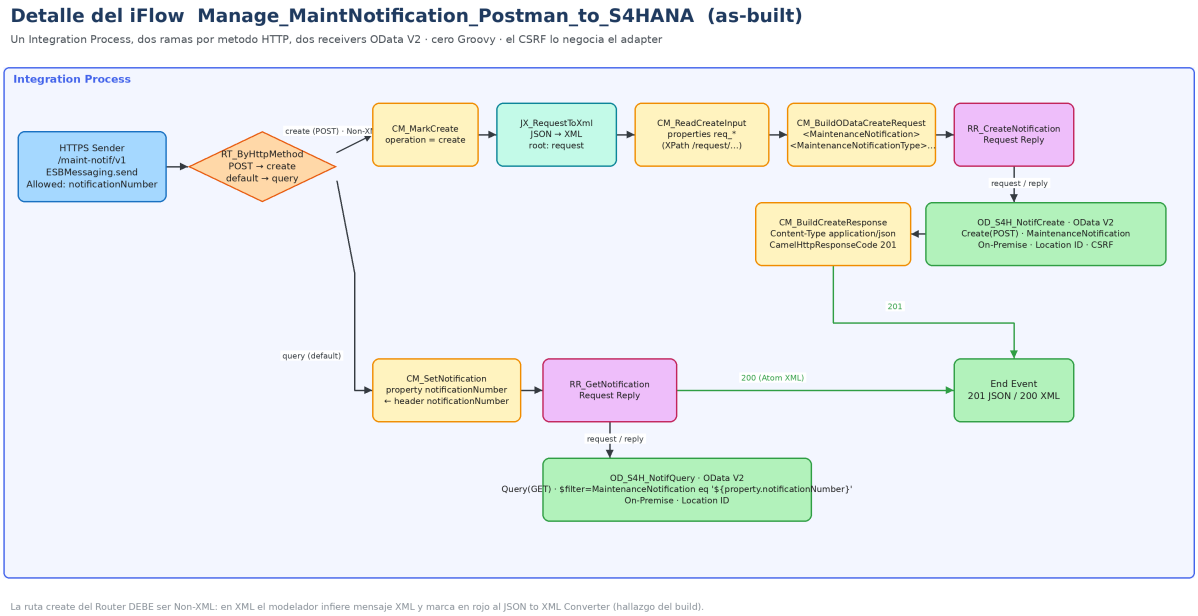
Cloud Connector: mapping and resources



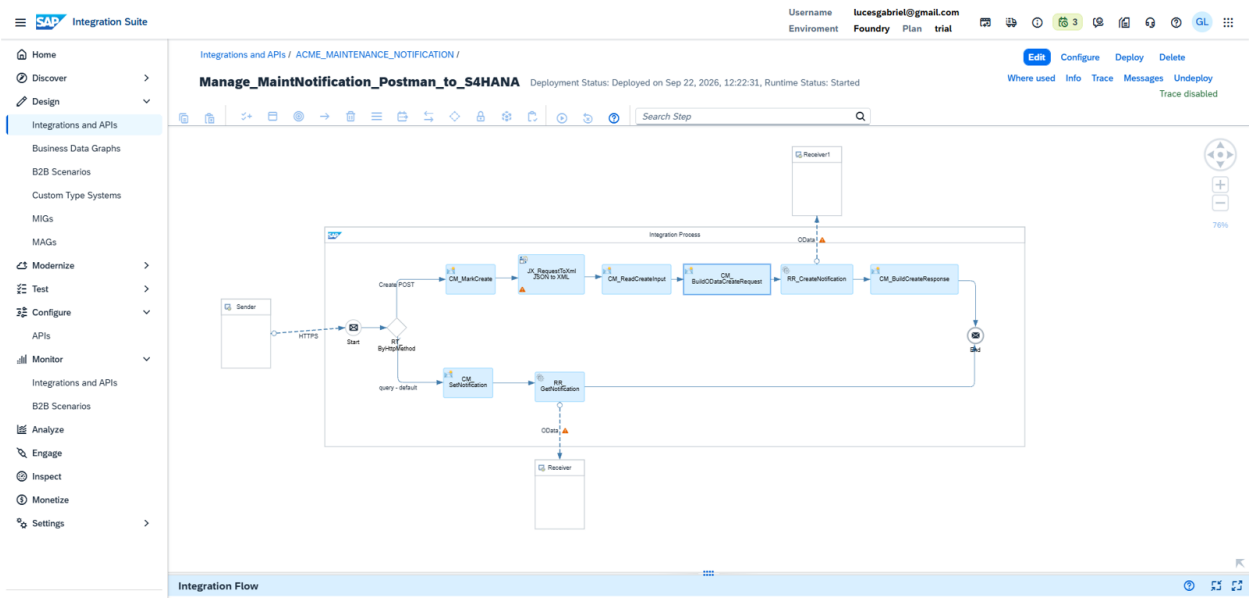
Resource with Path And All Sub-Paths

3.2 The iFlow: one endpoint, two branches

Manage_MaintNotification_Postman_to_S4HANA. HTTPS Sender on /maint-notif/v1; a Router on ${header.CamelHttpMethod} splits create (POST) from query (GET). No Groovy: a JSON to XML Converter, Content Modifiers with XPath and two OData V2 receivers.



iFlow detail

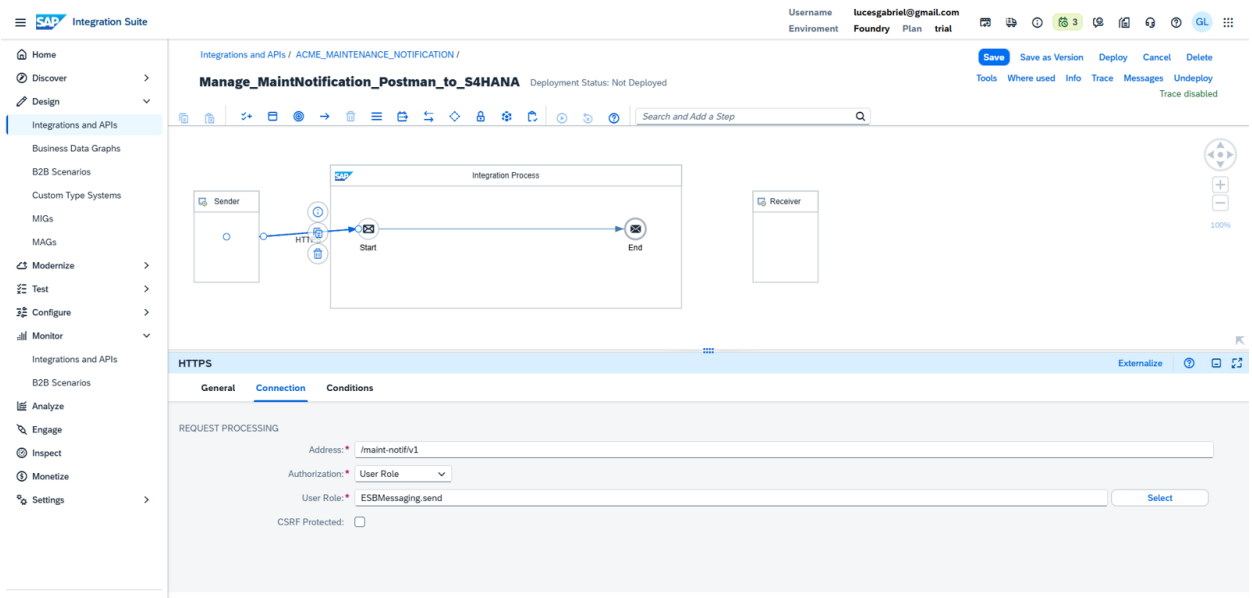


iFlow deployed and Started

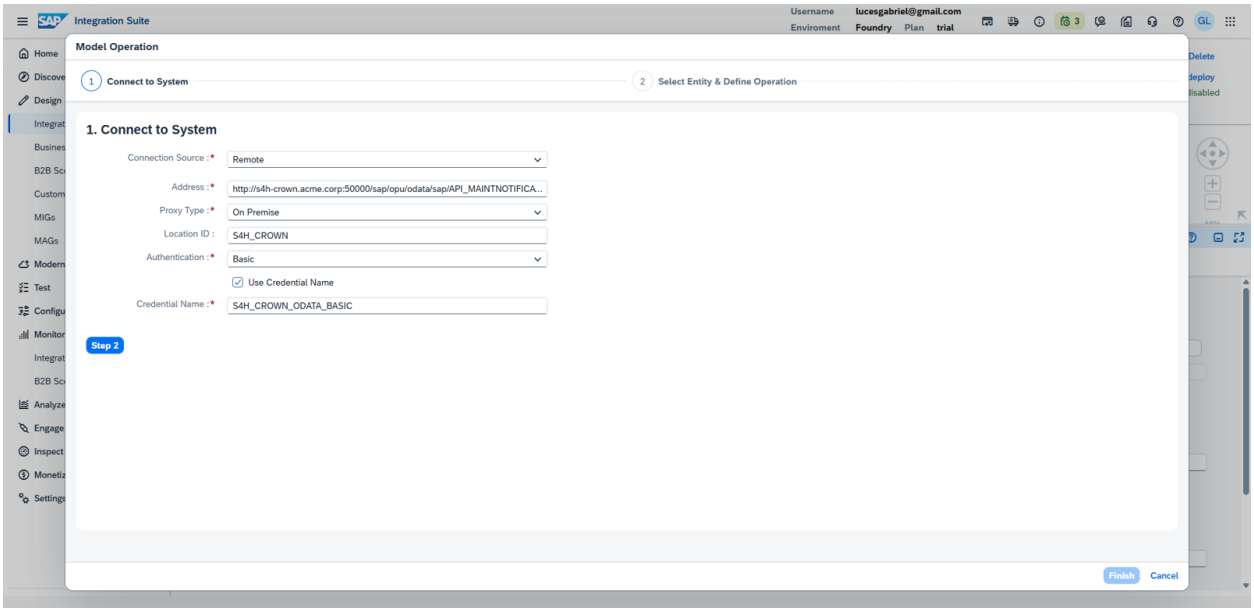
Create branch. The consumer's JSON is converted to XML, five properties are read via XPath, and a Content Modifier builds the body in the shape the adapter expects: root = EntitySet, child = EntityType.

```
<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>
```

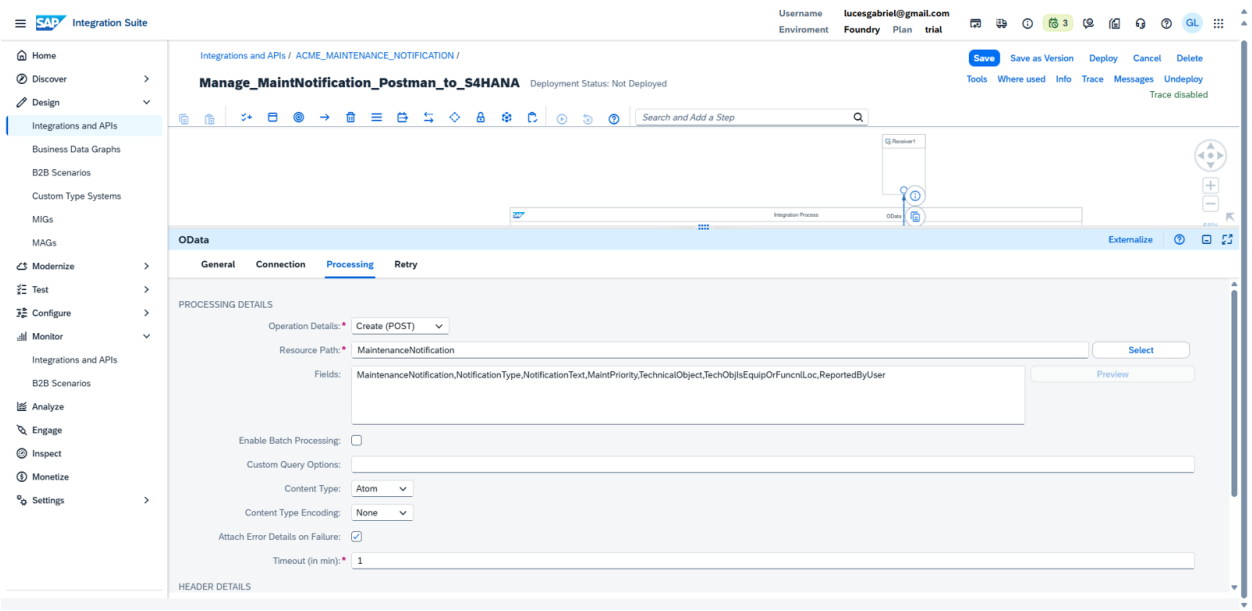
The OD_S4H_NotifCreate receiver (OData V2, Create (POST), Proxy Type = On-Premise, credential referenced by alias) is modeled with the Query Modeler, selecting only those six fields.



HTTPS Sender

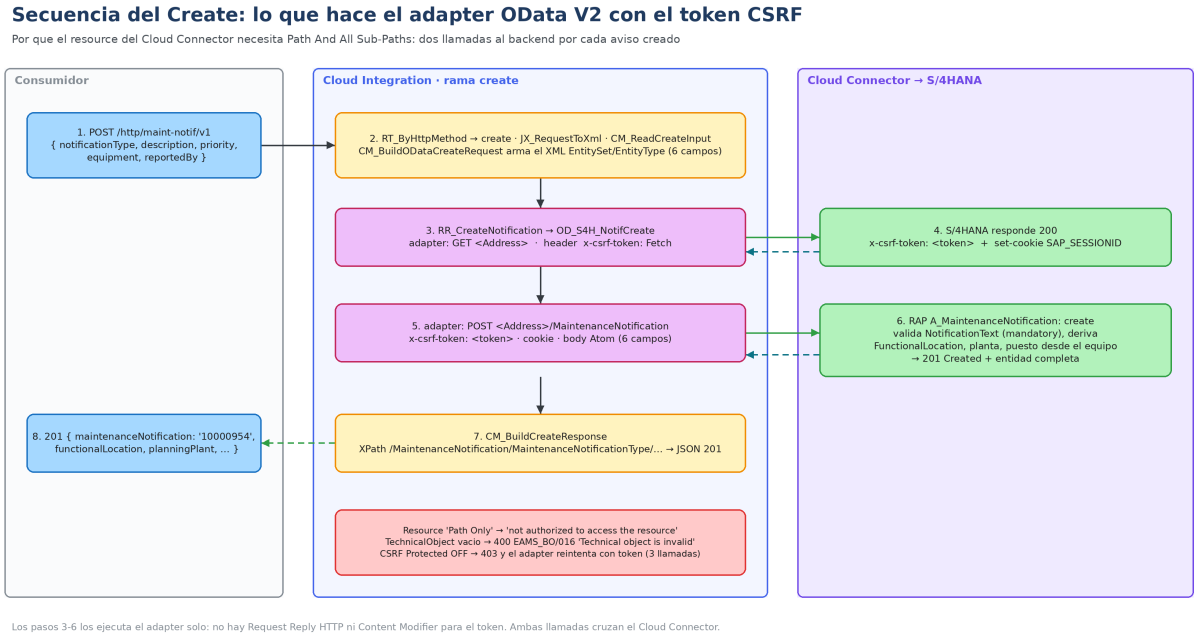


Query Modeler: remote connection through Cloud Connector



OData receiver: Create (POST) with the 6 fields

What the adapter does for you. With CSRF Protected enabled the adapter fetches the token and reuses it on the POST. Both calls cross the Cloud Connector.



Create sequence with CSRF

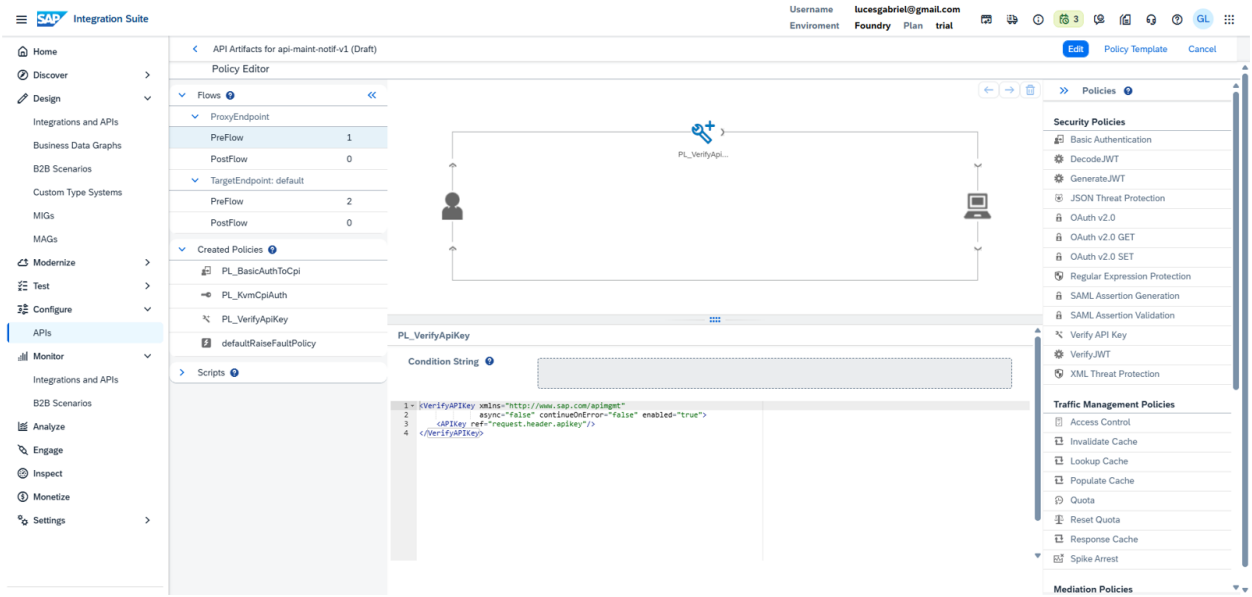
Query branch. A Content Modifier reads the notificationNumber header and the OD_S4H_NotifQuery receiver runs Query (GET) with $filter=MaintenanceNotification eq '${property.notificationNumber}'. A filter rather than key access: a non-existent notification returns 200 with an empty feed instead of an exception, which leaves the 404 decision to the iFlow in the next iteration.

3.3 API Management: minimal governance

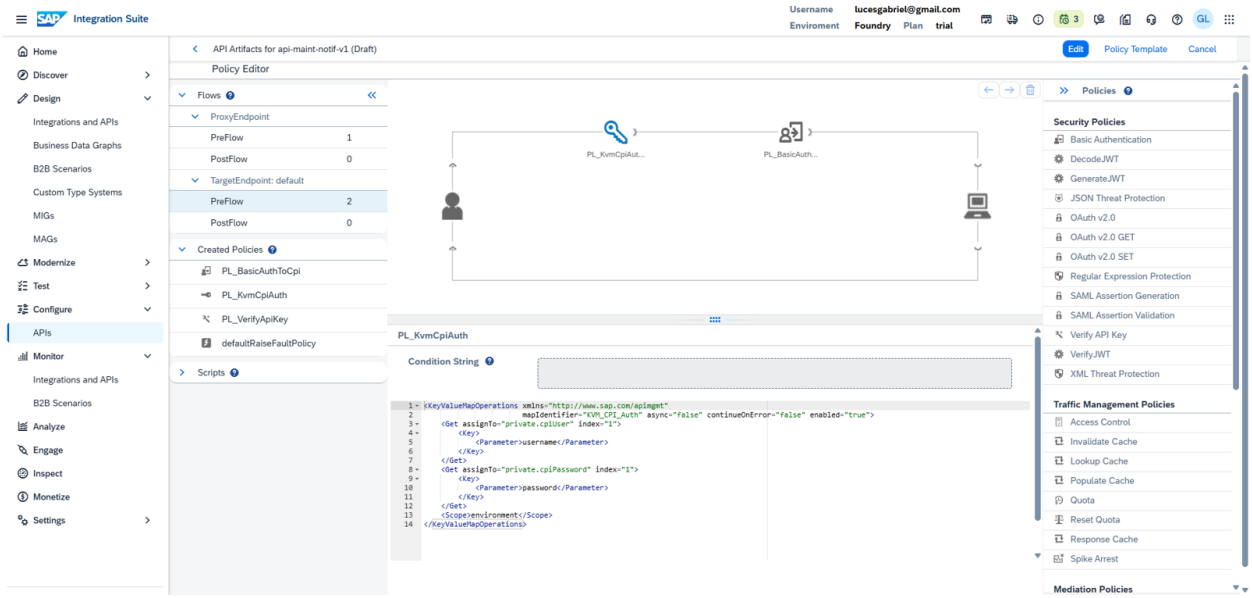
Proxy api-maint-notif-v1 on top of the Cloud Integration API Provider, base path /acme/notifications/v1, **no resources** (so POST and GET pass through without declaring anything). Three policies:

Where	Policy	What it does
ProxyEndpoint · PreFlow	PL_VerifyApiKey	rejects any request without a valid apikey with 401
TargetEndpoint · PreFlow	PL_KvmCpiAuth	reads user and password from an encrypted Key Value Map
TargetEndpoint · PreFlow	PL_BasicAuthToCpi	builds the Authorization: Basic header towards Cloud Integration

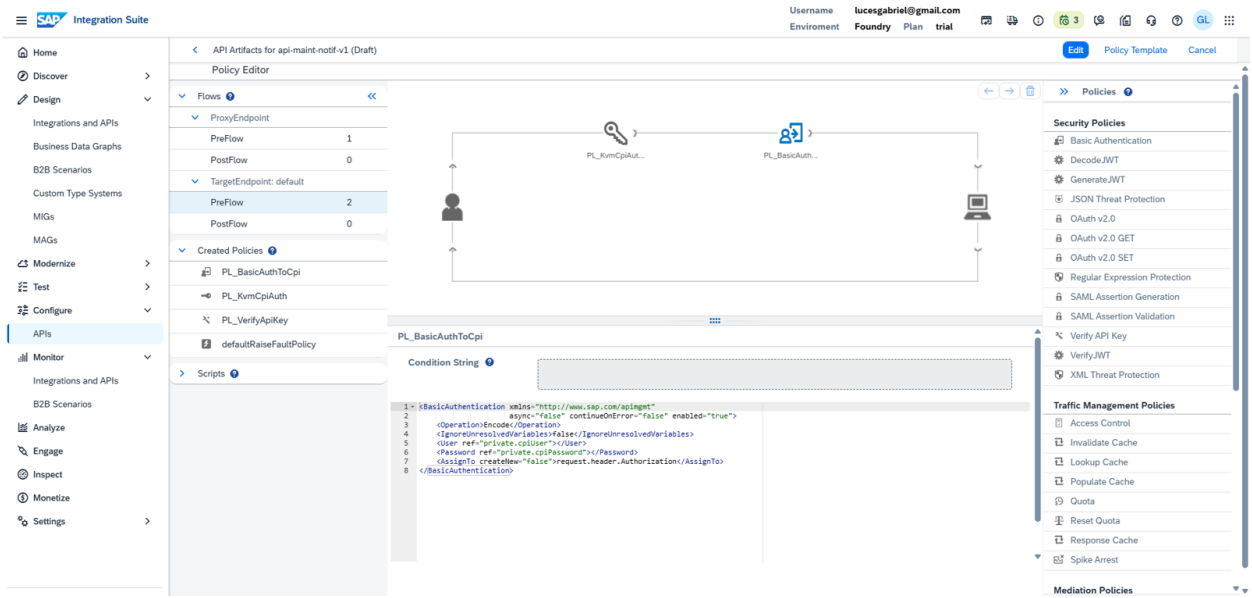
The consumer never sees the CPI credential, and the proxy XML never carries it either.



Verify API Key policy



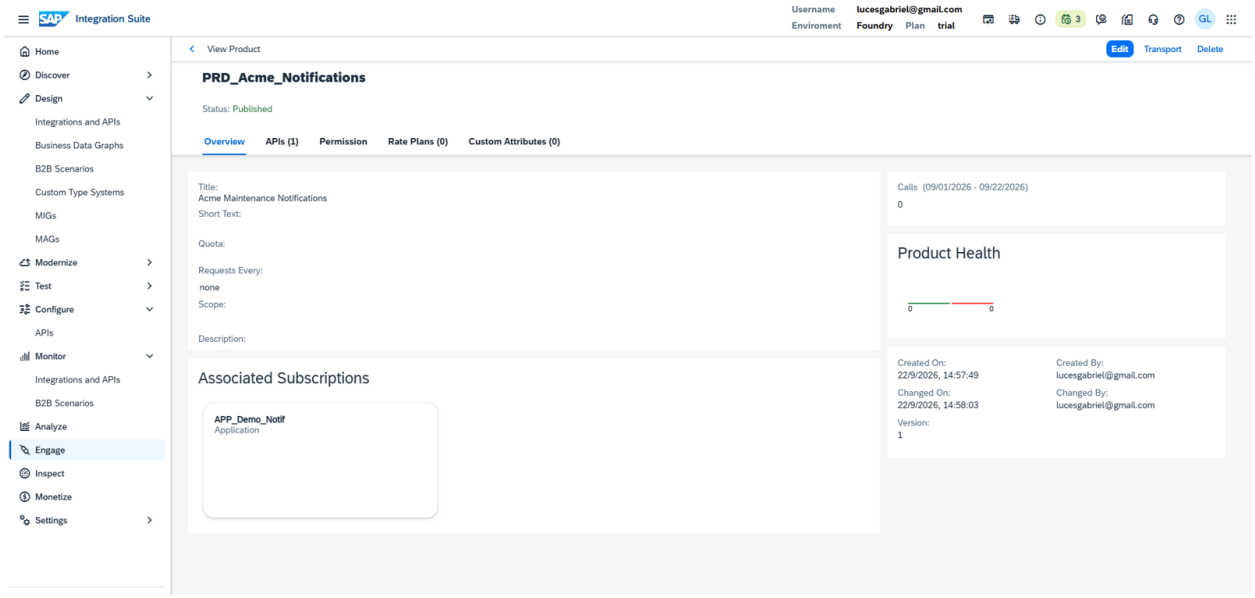
Key Value Map Operations policy



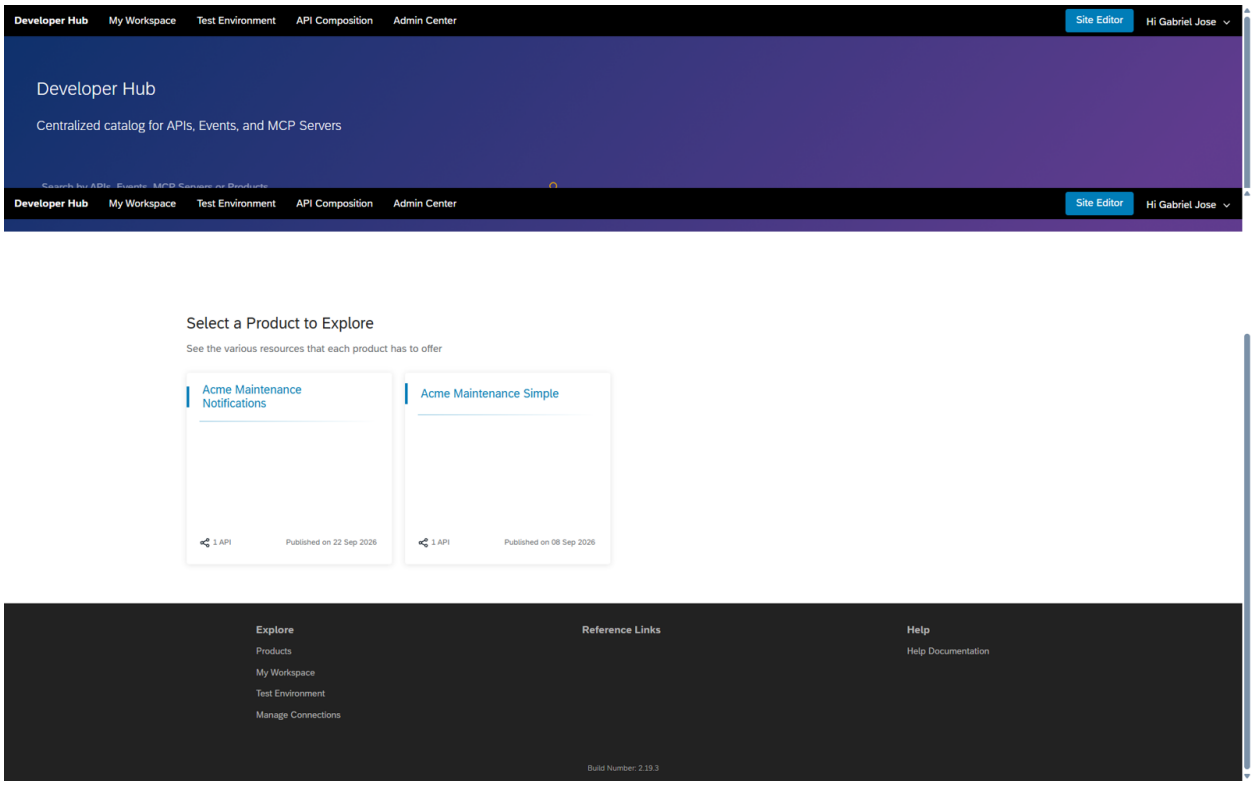
Basic Authentication policy towards CPI

Product PRD_Acme_Notifications published in the Developer Hub and an App subscribed to it:

that is where the apikey comes from.



Published product



Developer Hub: product catalog

---

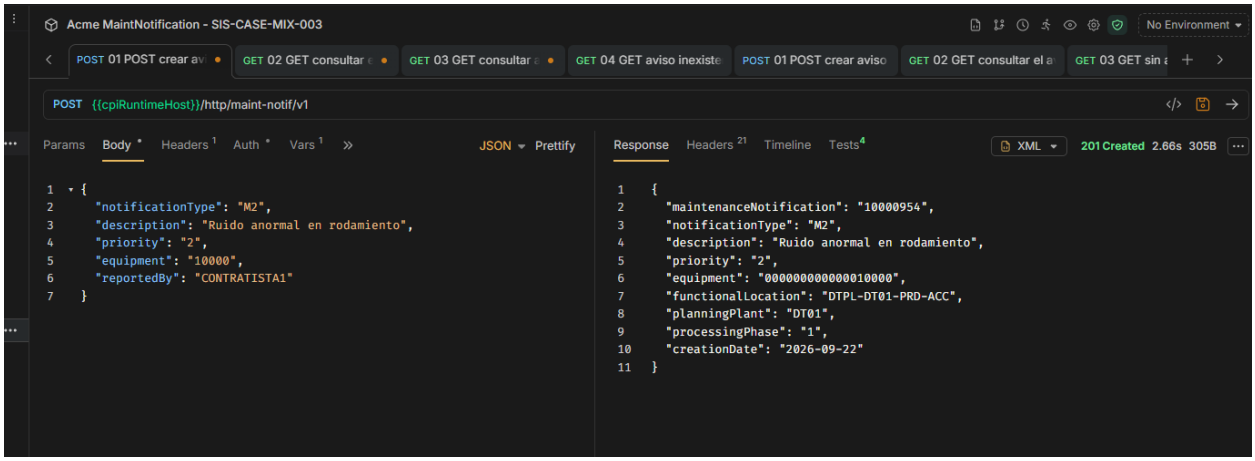
4. Testing

A Bruno collection with two folders: 01 CPI direct (Basic with the service key) and 02 APIM

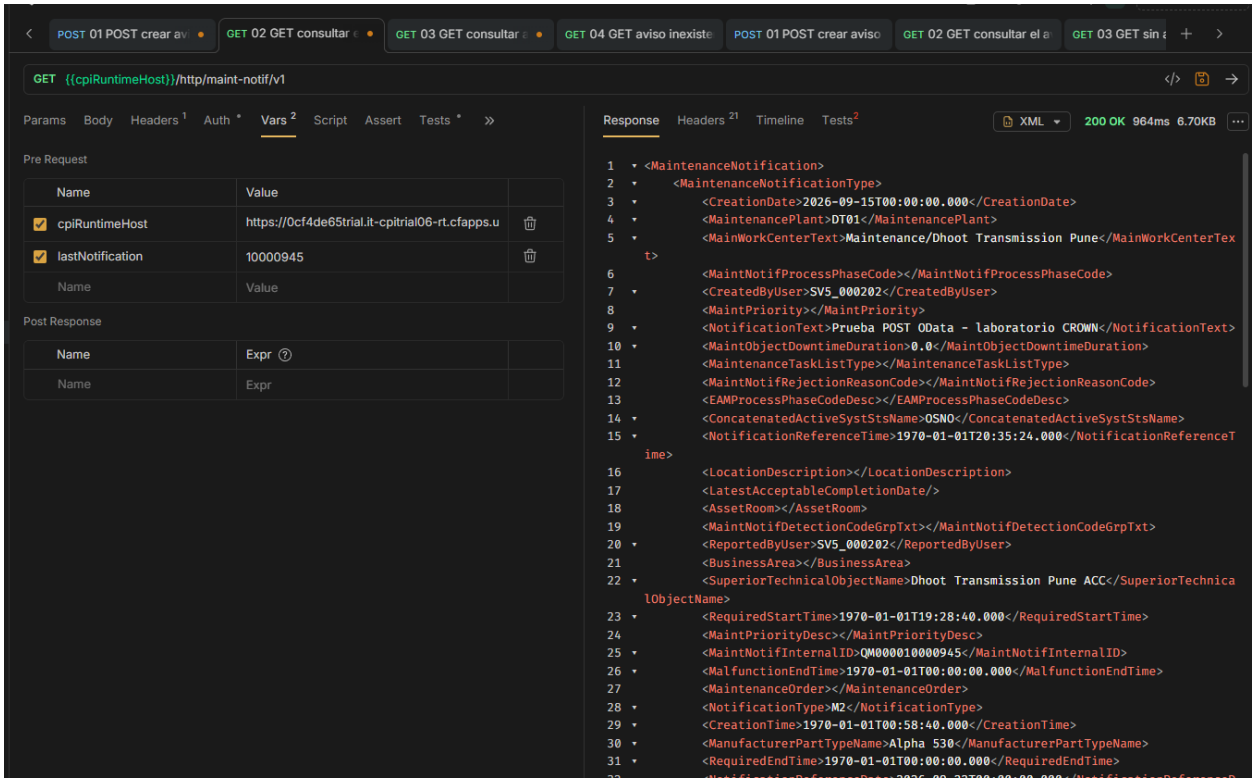
(apikey only). The POST stores the returned number in a variable and the next GET consumes it.

Straight to Cloud Integration. POST → **201** with notification 10000954; functional

location DTPL-DT01-PRD-ACC and plant DT01 were **not sent**: S/4HANA derived them from the equipment.

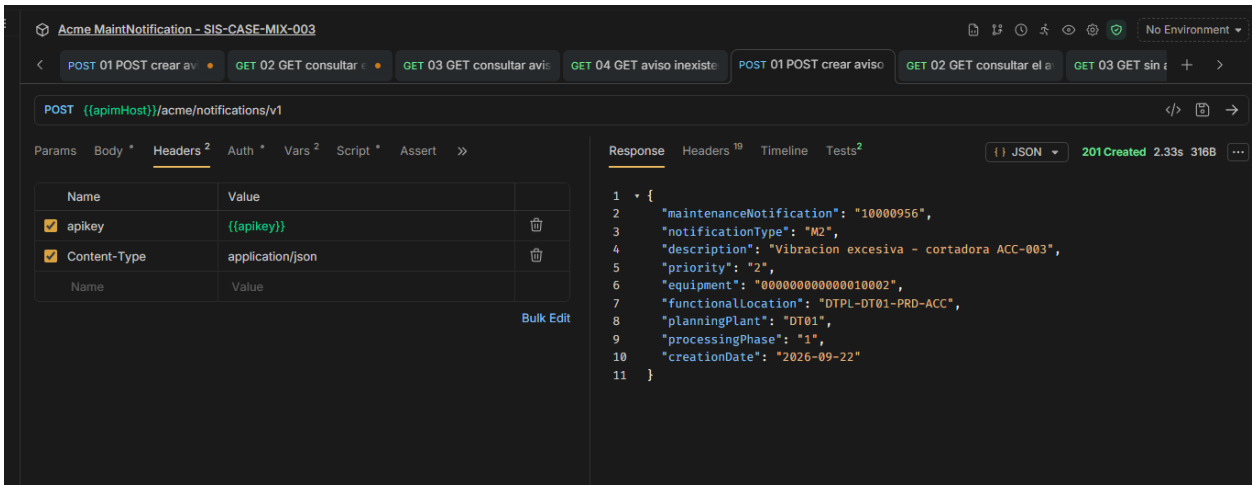


POST straight to CPI: 201

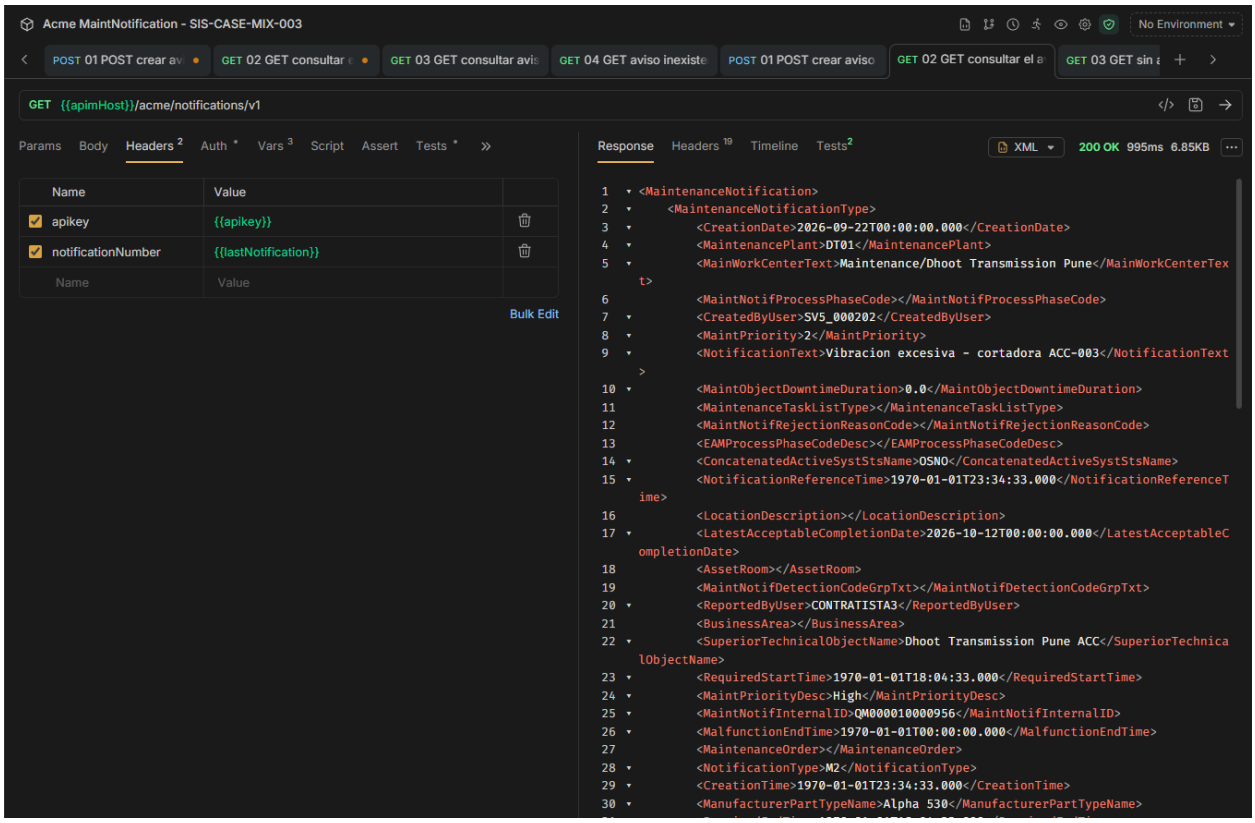


GET straight to CPI: 200

Through API Management. Same result, no Basic Auth on the client: POST → **201** with notification 10000956; GET → **200**.



POST via APIM: 201



GET via APIM: 200

---

5. What broke during the build (and how I fixed it)

Symptom	Actual cause	Fix
JSON to XML Converter flagged red: *"cannot process the message type passed by element Http Sender Component"*, Save fails	the Router's create route was left with **Expression Type = XML** (the default) and the modeler inferred an XML message on that branch	Non-XML + ${header.CamelHttpMethod} = 'POST'
POST → 500; the backend answers EAMS_BO/016 Technical object is invalid (with &1 empty)	typo in an Exchange Property name (req_equipmen vs ${property.req_equipment}): the property did not exist and SAP received an empty TechnicalObject	fix the name. In the MPL with Trace, setProperty[...] shows the real names
Query Modeler opens on *Local EDMX File* and says *"Unable to get edmx file content"*	orphan EDMX from an earlier attempt	Connection Source = Remote

Symptom	Actual cause	Fix
Create worked with **CSRF Protected unchecked**	the OData V2 adapter (1.30.2) receives the 403, fetches the token from the service document and retries the POST on its own	tick the box anyway: from 3 calls to 2 per notification

And one design lesson: **a 201 is not proof that the data is right.** The first notification I created through the API came out with no priority and no reporter. Checking the backend is part of the test.

---

6. What comes next

Input validation (400), an existence Router on the GET (404 instead of an empty 200), an Exception Subprocess with a {code, message, correlationId} error contract and 502 when the Cloud Connector is down, the GET response mapped to flat JSON, externalized parameters, and on the API Management side: Spike Arrest, Quota, stripping the apikey before the backend, and Fault Rules.

Tooling: SAP Integration Suite Trial (Cloud Integration, API Management, Developer Hub), SAP Cloud Connector, SAP S/4HANA 2025 on-premise, Bruno. Technical data comes from a lab system; the company is fictional. No credentials or internal hostnames in this document.