

JSON-RPC en Odoo

Guía completa del protocolo, la API externa
y la migración a JSON-2 en Odoo 20



```
POST /json/2/res.partner/search_read
```

```
Authorization: bearer .....
```

```
{  
  "domain": [["is_company", "=", true]],  
  "fields": ["name", "email"],  
  "limit": 20  
}
```

```
← 200 OK [{"id": 25, "name": "Deco Addict"}]
```

Marlon Falcón Hernández

Implemento BIM 2.0 · Valencia

Octubre 2026

Índice

Prólogo

Capítulo 1. Qué es JSON-RPC 2.0

Capítulo 2. Las APIs RPC de Odoo y su estado en Odoo 20

Capítulo 3. Arquitectura interna de JSON-RPC en Odoo

Capítulo 4. La API externa /jsonrpc

Capítulo 5. CRUD completo por JSON-RPC

Capítulo 6. JSON-2: la API externa de Odoo 20

Capítulo 7. Controladores JSON-RPC propios

Capítulo 8. JSON-RPC en el frontend OWL

Capítulo 9. Clientes en varios lenguajes

Capítulo 10. Errores, seguridad y rendimiento

Capítulo 11. Caso práctico: mediciones de obra desde una app externa

Apéndices

Prólogo

En Odoo 20, JSON-RPC sigue vivo, pero ya no es la puerta de entrada recomendada para integraciones externas: esa función pasa a la API JSON-2. Este libro explica las dos caras de JSON-RPC en Odoo —el protocolo interno que usa el cliente web y la API externa `/jsonrpc`— y cómo migrar a JSON-2 sin romper nada.

Va dirigido a desarrolladores e implantadores de Odoo que integran sistemas externos (apps móviles, ERPs, BIM, n8n, scripts) o que escriben módulos con controladores y componentes OWL.

Cómo leerlo:

- **Capítulos 1–3:** fundamentos. Qué es JSON-RPC 2.0 y cómo lo implementa Odoo por dentro.
- **Capítulos 4–5:** la API externa clásica `/jsonrpc`, con ejemplos de CRUD. Sigue funcionando en Odoo 20 salvo el servicio `db`.
- **Capítulo 6:** JSON-2, el reemplazo oficial. Lectura obligatoria para cualquier integración nueva.
- **Capítulos 7–8:** JSON-RPC dentro de tus módulos (controladores y frontend OWL).
- **Capítulos 9–11:** clientes en varios lenguajes, buenas prácticas y un caso práctico completo.

Los ejemplos usan una base de datos ficticia `miempresa` en `https://miempresa.example.com`. Sustituye host, base de datos y credenciales por los tuyos.

Capítulo 1. Qué es JSON-RPC 2.0

JSON-RPC 2.0 es un protocolo de llamada a procedimiento remoto: el cliente envía un objeto JSON que nombra un método y sus parámetros, y el servidor devuelve otro objeto JSON con el resultado o con un error. No impone URLs, verbos ni códigos HTTP; todo va en el cuerpo.

1.1 La petición

```
{
  "jsonrpc": "2.0",
  "method": "call",
  "params": { "name": "Marlon" },
  "id": 1
}
```

Campo	Obligatorio	Significado
<code>jsonrpc</code>	Sí	Siempre la cadena <code>"2.0"</code> .
<code>method</code>	Sí	Nombre del procedimiento. En Odoo casi siempre es <code>"call"</code> .
<code>params</code>	No	Objeto (parámetros con nombre) o array (posicionales). Odoo usa objeto.
<code>id</code>	No	Identificador para casar petición y respuesta. Sin <code>id</code> es una notificación.

1.2 La respuesta correcta

```
{ "jsonrpc": "2.0", "id": 1, "result": { "greeting": "Hola, Marlon" } }
```

1.3 La respuesta con error

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": { "code": -32601, "message": "Method not found", "data": {} }
}
```

La especificación reserva los códigos -32700 (JSON mal formado), -32600 (petición inválida), -32601 (método inexistente), -32602 (parámetros inválidos) y -32603 (error interno). Odoo usa sobre todo **200** como código de servidor genérico y mete el detalle de la excepción Python en `error.data` (ver capítulo 10).

1.4 Ideas clave para Odoo

- La respuesta HTTP suele ser **200 OK aunque haya error**: hay que mirar si existe la clave `error`.
- El protocolo no tiene sesión: la autenticación la pone Odoo, por cookie de sesión (cliente web) o por usuario y contraseña en cada llamada (`/jsonrpc`).
- JSON no tiene fechas ni decimales nativos: Odoo envía fechas como cadenas `"YYYY-MM-DD HH:MM:SS"` en UTC y los importes como números de coma flotante.

Capítulo 2. Las APIs RPC de Odoo y su estado en Odoo 20

En Odoo 20 conviven tres cosas que se llaman parecido y no son lo mismo: la API externa `/jsonrpc` (obsoleta, pero `common` y `object` aún funcionan), los controladores internos `type='jsonrpc'` (vigentes) y la API externa JSON-2 (la recomendada).

2.1 Cronología

Versión	Qué cambió
Odoo 22 (otoño 2028) / Online 21.1 (invierno 2027)	Retirada prevista de los servicios <code>common</code> y <code>object</code> de <code>/xmlrpc</code> , <code>/xmlrpc/2</code> y <code>/jsonrpc</code> .
Odoo 20 (otoño 2026) / Online 19.1	Se elimina el servicio <code>db</code> (listar, crear, duplicar, borrar bases de datos por RPC).
Odoo 19 (2025)	Nace la API JSON-2 en <code>/json/2/<modelo>/<método></code> con API key <code>bearer</code> . <code>/xmlrpc</code> y <code>/jsonrpc</code> pasan a obsoletos. <code>type='json'</code> se renombra a <code>type='jsonrpc'</code> .
Odoo 14	Aparecen las API keys de usuario, utilizables en lugar de la contraseña en XML-RPC y JSON-RPC.
Odoo 8–18	XML-RPC y JSON-RPC conviven como API externa estable, con los servicios <code>common</code> , <code>db</code> y <code>object</code> .

Un matiz importante: la documentación de Odoo 19 anunciaba la retirada completa en Odoo 20, pero la de Odoo 20 la escalona. Solo `db` desaparece ahora; `common` y `object` aguantan hasta Odoo 22 ([doc. Odoo 20](#)).

2.2 Qué se rompe al subir a Odoo 20

- **Se rompe:** cualquier script que llame a `db.list`, `db.create_database`, `db.dump`, `db.drop`, `db.server_version`, etc. por `/jsonrpc` o `/xmlrpc`.
- **Sigue funcionando (con aviso en el log):** `common.version`, `common.login`, `common.authenticate` y `object.execute_kw`.
- **No le afecta la obsolescencia:** los controladores `@http.route(type='jsonrpc')`, incluido el `/web/dataset/call_kw` que usa el cliente web.

El log de Odoo avisa en cada llamada al endpoint antiguo con un *warning* del logger `odoo.addons.rpc.controllers.jsonrpc`. Si necesitas silenciarlo temporalmente: `--log-handler odoo.addons.rpc.controllers.jsonrpc:ERROR`.

2.3 Disponibilidad según plan

En Odoo Online, el acceso a la API externa (RPC o JSON-2) solo está incluido en el plan *Custom*; no está disponible en *One App Free* ni en *Standard*. En Odoo.sh y en instalaciones propias no hay esa restricción.

Capítulo 3. Arquitectura interna de JSON-RPC en Odoo

Toda petición HTTP a Odoo pasa por `odoo/http.py` : se busca la ruta, se elige un *dispatcher* según el `type` de la ruta, se autentica según su `auth` y se ejecuta el método del controlador dentro de una transacción SQL.

En Odoo 20, tres puertas JSON llegan al mismo ORM

Quién llama, por qué endpoint y en qué estado está cada uno

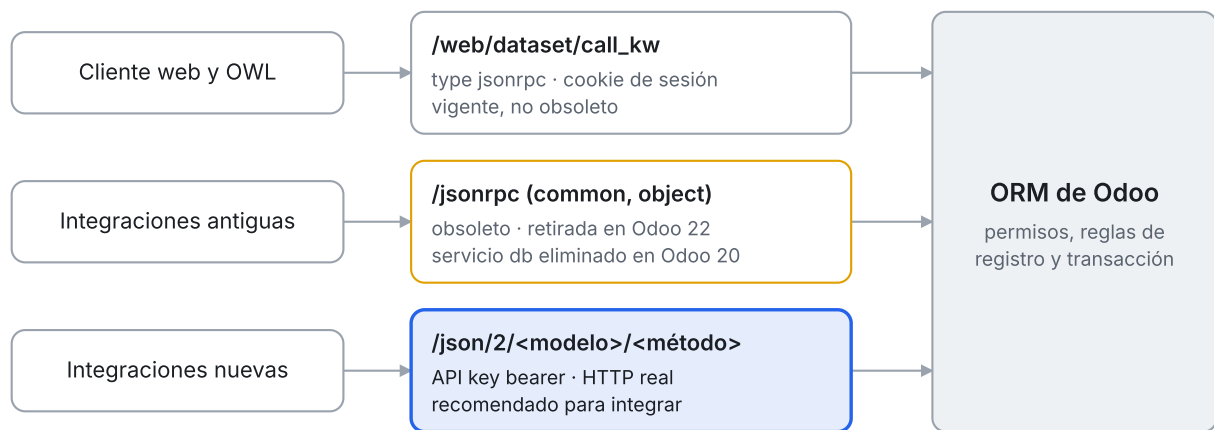


Figura 1. Vías de acceso JSON a Odoo 20

Las tres vías acaban en el mismo ORM, con los mismos permisos; lo que cambia es cómo se autentican y si tienen fecha de caducidad.

3.1 Tipos de ruta en Odoo 20

type	Formato de entrada	Formato de salida	Uso típico
http	Formulario, query string, multipart	HTML, ficheros, cualquier <code>Response</code>	Páginas web, descargas
jsonrpc	Sobre JSON-RPC 2.0 (<code>params</code>)	Sobre JSON-RPC 2.0 (<code>result</code> / <code>error</code>)	Cliente web, OWL, controladores propios
json	Igual que <code>jsonrpc</code>	Igual que <code>jsonrpc</code>	Alias obsoleto desde Odoo 19; emite un <i>warning</i>
json2	JSON plano con argumentos con nombre	JSON plano + códigos HTTP reales	Familia de la API JSON-2

3.2 Modos de autenticación (`auth`)

- `user` : exige sesión iniciada (cookie `session_id`). Es lo que usa el backend.

- `public` : acepta visitantes anónimos con el usuario público; si hay sesión, usa al usuario real.
- `none` : sin usuario ni, a menudo, base de datos. Para utilidades de bajo nivel.
- `bearer` : autentica con API key en la cabecera `Authorization: bearer <clave>` . Es el que usa JSON-2 y el que conviene en endpoints propios para máquinas.

3.3 El ciclo de una llamada del cliente web

1. Un componente OWL llama a `orm.searchRead(...)` o a `rpc("/ruta", params)` .
2. El navegador hace `POST` a `/web/dataset/call_kw/<modelo>/<método>` con un sobre JSON-RPC y la cookie de sesión.
3. El dispatcher `jsonrpc` deserializa `params` , autentica por sesión y llama a `call_kw` .
4. `call_kw` instancia el modelo con el usuario y el contexto, y ejecuta el método del ORM respetando permisos y reglas de registro.
5. El resultado se serializa en `result` ; si se lanza una excepción, Odoo hace *rollback* y devuelve `error` .

3.4 Dónde vive cada cosa en el código

- `odoo/http.py` : clases `Request` , dispatchers y el decorador `route` .
- `odoo/service/model.py` : `call_kw` , `execute_kw` y la lógica de reintentos ante conflictos de concurrencia.
- `addons/rpc/controllers/` : los endpoints externos `/xmlrpc` , `/xmlrpc/2` y `/jsonrpc` , separados en su propio módulo desde Odoo 19.
- `addons/web/controllers/dataset.py` : `/web/dataset/call_kw` y compañía.
- `addons/web/static/src/core/network/rpc.js` : la función `rpc` del frontend.

Capítulo 4. La API externa /jsonrpc

El endpoint `/jsonrpc` expone tres servicios — `common`, `db` y `object` — mediante el método único `call`. En Odoo 20 funcionan `common` y `object`; `db` ya no existe. Úsalo solo para mantener integraciones existentes.

4.1 Forma general de la llamada

```
POST /jsonrpc
Content-Type: application/json

{
  "jsonrpc": "2.0",
  "method": "call",
  "params": {
    "service": "object",
    "method": "execute_kw",
    "args": ["miempresa", 2, "API_KEY_O_PASSWORD", "res.partner", "search_count", [[]]]
  },
  "id": 1
}
```

`params.service` elige el servicio, `params.method` la función y `params.args` sus argumentos posicionales.

4.2 Servicio `common`

Función	Argumentos	Devuelve	Reemplazo en Odoo 20
<code>version</code>	—	Diccionario con <code>server_version</code> , <code>server_version_info</code> , <code>protocol_version</code>	<code>GET /web/version</code>
<code>login</code>	<code>db</code> , <code>login</code> , <code>password</code>	<code>uid</code> (entero) o <code>false</code>	API key + <code>res.users/context_get</code> en JSON-2
<code>authenticate</code>	<code>db</code> , <code>login</code> , <code>password</code> , <code>user_agent_env</code>	<code>uid</code> o <code>false</code>	Igual que <code>login</code>

```
{"jsonrpc": "2.0", "method": "call", "id": 1,
 "params": {"service": "common", "method": "login",
            "args": ["miempresa", "api@miempresa.com", "API_KEY"]}}
```

En el campo contraseña puedes —y debes— poner una **API key** en vez de la contraseña real del usuario.

4.3 Servicio `object`

- `execute(db, uid, password, model, method, *args)`: argumentos posicionales sueltos.
- `execute_kw(db, uid, password, model, method, args, kw)`: `args` es la lista posicional y `kw` un diccionario con argumentos con nombre, incluido `context`. Es la forma preferida.

Ambas son sin estado: base de datos, `uid` y contraseña viajan en cada llamada. Si el método no es `@api.model`, el primer elemento de `args` es la lista de IDs.

4.4 Servicio `db` (eliminado en Odoo 20)

Las funciones `list`, `create_database`, `duplicate_database`, `drop`, `dump`, `restore`, `rename`, `change_admin_password`, `db_exist`, `list_lang`, `list_countries`, `server_version` y `migrate_databases` ya no responden. Alternativas:

Necesidad	Alternativa en Odoo 20
Listar bases	<code>POST /web/database/list</code> (JSON-RPC, cuerpo <code>{}</code>)
Crear / duplicar / borrar	<code>POST /web/database/create</code> , <code>/duplicate</code> , <code>/drop</code> (formulario <code>x-www-form-urlencoded</code> , con <code>master_pwd</code>)
Copia de seguridad	<code>POST /web/database/backup</code> (<code>master_pwd</code> , <code>name</code> , <code>backup_format</code>)
Restaurar	<code>POST /web/database/restore</code> (<code>multipart/form-data</code> , fichero <code>backup_file</code>)
Versión del servidor	<code>GET /web/version</code>
Idiomas y países	JSON-2 sobre <code>res.lang</code> y <code>res.country</code>

Recuerda que en producción el gestor de bases debe estar protegido (`list_db = False` y una `admin_passwd` fuerte en el fichero de configuración).

Capítulo 5. CRUD completo por JSON-RPC

Con `execute_kw` puedes llamar a cualquier método público del ORM. Este capítulo usa un pequeño cliente Python y muestra la equivalencia en JSON-2 para que la migración sea directa.

5.1 Cliente mínimo

```
import itertools, requests

URL, DB = "https://miempresa.example.com", "miempresa"
LOGIN, KEY = "api@miempresa.com", "API_KEY"
_ids = itertools.count(1)

def jsonrpc(service, method, *args):
    payload = {"jsonrpc": "2.0", "method": "call", "id": next(_ids),
              "params": {"service": service, "method": method, "args": args}}
    r = requests.post(f"{URL}/jsonrpc", json=payload, timeout=30)
    r.raise_for_status()
    data = r.json()
    if "error" in data:
        raise RuntimeError(data["error"]["data"]["message"])
    return data["result"]

uid = jsonrpc("common", "login", DB, LOGIN, KEY)

def call(model, method, args=(), **kw):
    return jsonrpc("object", "execute_kw", DB, uid, KEY, model, method, list(args), kw)
```

5.2 Leer

```
# Contar y buscar IDs
n = call("res.partner", "search_count", [[["is_company", "=", True]])
ids = call("res.partner", "search", [[["is_company", "=", True]], limit=10, order="name")

# Leer campos de IDs concretos
rows = call("res.partner", "read", [ids], fields=["name", "email", "country_id"])

# Buscar y leer en una sola transacción (lo recomendado)
rows = call("res.partner", "search_read",
            [[["name", "ilike", "deco"]],
            fields=["name", "email"], limit=20, offset=0,
            context={"lang": "es_ES"})

# Descubrir campos de un modelo
meta = call("res.partner", "fields_get", attributes=["string", "type", "required"])
```

Los campos `many2one` llegan como `[id, "nombre"]`; los `one2many` y `many2many`, como listas de IDs.

5.3 Crear, modificar y borrar

```
new_id = call("res.partner", "create", [{"name": "Constructora Turia", "is_company":
True}])
call("res.partner", "write", [[new_id], {"phone": "+34 960 000 000"}])
call("res.partner", "unlink", [[new_id]])
```

`create` acepta un diccionario o una lista de diccionarios (creación masiva) y devuelve el ID o la lista de IDs.

5.4 Relaciones x2many: los comandos

Comando	Tupla	Efecto
CREATE	<code>(0, 0, {valores})</code>	Crea un registro hijo y lo enlaza
UPDATE	<code>(1, id, {valores})</code>	Modifica un hijo enlazado
DELETE	<code>(2, id, 0)</code>	Borra el hijo de la base de datos
UNLINK	<code>(3, id, 0)</code>	Quita el enlace sin borrar el registro
LINK	<code>(4, id, 0)</code>	Enlaza un registro existente
CLEAR	<code>(5, 0, 0)</code>	Quita todos los enlaces
SET	<code>(6, 0, [ids])</code>	Sustituye todos los enlaces por esa lista

```
call("sale.order", "create", [{
    "partner_id": 7,
    "order_line": [(0, 0, {"product_id": 12, "product_uom_qty": 3})],
}])
```

5.5 Dominios y contexto

- Un dominio es una lista de condiciones `[campo, operador, valor]` unidas por `&` implícito; usa `"|"` y `"!"` en notación prefija: `["|", ["city", "=", "Valencia"], ["city", "=", "Madrid"]]`.
- Operadores habituales: `=`, `!=`, `>`, `>=`, `<`, `<=`, `in`, `not in`, `like`, `ilike`, `=like`, `child_of`, `parent_of`.
- El `context` cambia el comportamiento: `lang`, `tz`, `active_test: False` (incluir archivados), `allowed_company_ids` (multiempresa).

5.6 Llamar a métodos de negocio

Cualquier método público (que no empiece por `_`) es invocable: `call("sale.order", "action_confirm", [[order_id]])`. Es mejor llamar a un método de negocio que encadenar varios `write`, porque todo ocurre en una sola transacción.

Capítulo 6. JSON-2: la API externa de Odoo 20

Toda integración nueva con Odoo 20 debe usar JSON-2: un `POST` a `/json/2/<modelo>/<método>` con una API key `bearer` y un cuerpo JSON con argumentos con nombre. No hay sobre JSON-RPC, no hay `uid` ni contraseña, y los errores vuelven con códigos HTTP reales ([doc. JSON-2](#)).

6.1 Anatomía de la petición

```
POST /json/2/res.partner/search_read HTTP/1.1
Host: miempresa.example.com
Authorization: bearer 6578616d706c65206a736f6e20617069206b6579
X-Odoo-Database: miempresa
Content-Type: application/json; charset=utf-8
User-Agent: implemento-bim/1.0

{
  "context": {"lang": "es_ES"},
  "domain": [{"name", "ilike", "%deco%"}, {"is_company", "=", true}],
  "fields": ["name", "email"],
  "limit": 20
}
```

Elemento	Obligatorio	Notas
<code>Authorization</code>	Sí	<code>bearer</code> + API key del usuario
<code>Content-Type</code>	Sí	<code>application/json</code> , mejor con <code>charset</code>
<code>X-Odoo-Database</code>	Solo si el dominio no basta	Necesario con varias bases en el mismo servidor sin <code>dbfilter</code> por host
<code>User-Agent</code>	Recomendado	Nombre de tu software, útil en los logs
<code>ids</code> (cuerpo)	Según método	Lista de IDs; se omite en métodos <code>@api.model</code>
<code>context</code> (cuerpo)	No	Igual que en el ORM
Resto del cuerpo	Según método	Cada parámetro por su nombre exacto en Python

6.2 Respuesta

- **Éxito:** HTTP 200 y en el cuerpo directamente el valor devuelto por el método (sin `result`).
- **Error:** HTTP 4xx/5xx y un objeto con `name` (clase de la excepción), `message`, `arguments`, `context` y `debug` (traza).

```
HTTP/1.1 401 Unauthorized
{"name": "werkzeug.exceptions.Unauthorized", "message": "Invalid apikey",
 "arguments": ["Invalid apikey", 401], "context": {}, "debug": "Traceback..."}
```

6.3 API keys

- Se crean en *Preferencias del usuario* → *Seguridad de la cuenta* → *Nueva API key*, o desde el icono de llave en la página `/doc`.
- Requieren descripción y duración. La duración máxima es de **tres meses**: las claves de larga vida deben rotarse.
- La clave se muestra una sola vez. Guárdala en un gestor de secretos, nunca en el código.
- Desde Odoo 19 se pueden generar y revocar por API: `res.users.apikeys/generate` (con `key`, `scope`, `name`, `expiration_date`) y `res.users.apikeys/revoke`. El `scope` `rpc` es el genérico para controladores con `auth='bearer'`.

Usa **usuarios bot** dedicados para integraciones: permisos mínimos, contraseña vacía (login deshabilitado) y trazabilidad clara en los campos de auditoría.

6.4 Transacciones: una llamada, una transacción

Cada llamada JSON-2 se ejecuta en su propia transacción: *commit* si va bien, *rollback* si falla. No se pueden encadenar varias llamadas en una misma transacción. Por eso, para operaciones que deben ser atómicas (reservas, pagos, confirmaciones), crea un método de negocio en un módulo propio que lo haga todo y llámalo una sola vez. `search_read` es mejor que `search` + `read` por la misma razón.

6.5 La página /doc

Cada base de datos Odoo 19+ publica en `https://<host>/doc` su documentación dinámica: modelos, campos y métodos realmente disponibles con los módulos instalados, incluidos los tuyos. Es el primer sitio que hay que mirar antes de escribir una llamada.

6.6 Tabla de migración desde /jsonrpc

Antes (/jsonrpc)	Ahora (JSON-2)
<code>common.version()</code>	<code>GET /web/version</code>
<code>common.login(db, login, pwd) → uid</code>	No hace falta. Si necesitas tu <code>uid</code> : <code>POST /json/2/res.users/context_get</code> sin <code>ids</code>
<code>object.execute_kw(db, uid, pwd, "res.partner", "search_read", [dominio], {"fields": [...]})</code>	<code>POST /json/2/res.partner/search_read</code> con <code>{"domain": ..., "fields": [...]}</code>
<code>execute_kw(..., "read", [[1,2,3]], {"fields": ["name"]})</code>	<code>POST /json/2/res.partner/read</code> con <code>{"ids": [1,2,3], "fields": ["name"]}</code>

Antes (/jsonrpc)	Ahora (JSON-2)
<code>execute_kw(..., "write", [[7], {"phone": "..."}])</code>	POST /json/2/res.partner/write con {"ids": [7], "vals": {"phone": "..."} }
<code>execute_kw(..., "create", [{...}])</code>	POST /json/2/res.partner/create con {"vals_list": [{...}] }
<code>execute_kw(..., "unlink", [[7]])</code>	POST /json/2/res.partner/unlink con {"ids": [7]}
Contexto en <code>kw["context"]</code>	Clave <code>context</code> del cuerpo
Base de datos en cada llamada	Cabecera <code>X-0doo-Database</code> , solo si hace falta
Comprobar <code>error</code> con HTTP 200	Comprobar el código HTTP

La trampa principal: JSON-2 no admite argumentos posicionales. Hay que usar el nombre real del parámetro en la firma Python (`vals_list` en `create` , `vals` en `write` , `domain` en `search`). Compruébalo en `/doc` .

6.7 Ejemplo completo en Python

```
import os, requests

BASE = "https://miempresa.example.com/json/2"
H = {"Authorization": f"bearer {os.environ['OD00_API_KEY']}",
     "X-0doo-Database": "miempresa",
     "User-Agent": "implemento-bim/1.0"}

def odoo(model, method, **body):
    r = requests.post(f"{BASE}/{model}/{method}", headers=H, json=body, timeout=30)
    if not r.ok:
        err = r.json()
        raise RuntimeError(f"{r.status_code} {err['name']}: {err['message']}")
    return r.json()

partners = odoo("res.partner", "search_read",
                domain=[["is_company", "=", True]], fields=["name"], limit=5)
new_ids = odoo("res.partner", "create", vals_list=[{"name": "Obra Ruzafa"}])
odoo("res.partner", "write", ids=new_ids, vals={"city": "Valencia"})
```

Capítulo 7. Controladores JSON-RPC propios

En tus módulos, los endpoints JSON se declaran con `@http.route(type='jsonrpc')`. Este tipo no está obsoleto en Odoo 20; lo único que cambió en Odoo 19 es el nombre: `type='json'` sigue funcionando como alias, pero emite un aviso y conviene renombrarlo.

7.1 Un controlador básico

```
# mi_modulo/controllers/main.py
from odoo import http
from odoo.http import request

class ObraController(http.Controller):

    @http.route("/bim/obra/resumen", type="jsonrpc", auth="user", methods=["POST"])
    def resumen(self, obra_id, incluir_partidas=False):
        obra = request.env["project.project"].browse(int(obra_id))
        obra.check_access("read")
        datos = {"id": obra.id, "nombre": obra.name}
        if incluir_partidas:
            datos["partidas"] = obra.task_ids.read(["name", "stage_id"])
        return datos
```

Los parámetros del método se rellenan con las claves de `params`. Lo que devuelvas se serializa en `result`.

7.2 Cómo se llama

```
curl -X POST https://miempresa.example.com/bim/obra/resumen \
-H "Content-Type: application/json" \
-b "session_id=..." \
-d '{"jsonrpc":"2.0","method":"call","params":
{"obra_id":12,"incluir_partidas":true},"id":1}'
```

Error típico al migrar a Odoo 19/20: enviar un JSON plano `{"obra_id": 12}` a una ruta `jsonrpc`. El controlador no recibe los parámetros. Hay que envolverlos en el sobre `{"jsonrpc":"2.0","method":"call","params":{...}}`.

7.3 Endpoints para máquinas: `auth='bearer'`

Para que un sistema externo llame a tu endpoint sin cookie de sesión, usa autenticación por API key:

```
@http.route("/bim/api/mediciones", type="jsonrpc", auth="bearer", methods=["POST"],  
csrf=False)  
def mediciones(self, obra_id, lineas):  
    return request.env["bim.medicion"].registrar_lote(int(obra_id), lineas)
```

El cliente envía la cabecera `Authorization: bearer <API_KEY>`. El usuario de la clave es el que ejecuta el código, con sus permisos.

7.4 Reglas de oro

- **Valida y convierte la entrada:** `int()`, listas permitidas de campos, longitudes máximas. Nunca construyas SQL con datos del cliente.
- **No uses `sudo()` por comodidad.** Si lo necesitas, limita el alcance al mínimo y comprueba antes los permisos del usuario real.
- **Pon la lógica en el modelo**, no en el controlador: así es reutilizable desde JSON-2, desde el cliente web y desde tests.
- `auth='public'` solo para datos realmente públicos, y con límites de tamaño en la respuesta.
- Las rutas `jsonrpc` no aplican CSRF como las `http`; aun así, protege las de escritura con `auth='user'` o `auth='bearer'`.

Capítulo 8. JSON-RPC en el frontend OWL

El cliente web de Odoo 20 habla con el servidor exclusivamente por JSON-RPC sobre sesión. Tienes dos herramientas: el servicio `orm` para métodos de modelos y la función `rpc` para rutas propias.

8.1 El servicio `orm`

```
/** @odoo-module */
import { Component, onWillStart, useState } from "@odoo/owl";
import { useService } from "@web/core/utils/hooks";

export class ResumenObras extends Component {
  static template = "bim.ResumenObras";

  setup() {
    this.orm = useService("orm");
    this.state = useState({ obras: [] });
    onWillStart(async () => {
      this.state.obras = await this.orm.searchRead(
        "project.project",
        [["active", "=", true]],
        ["name", "partner_id"],
        { limit: 50, order: "name" }
      );
    });
  }

  async cerrarObra(id) {
    await this.orm.call("project.project", "action_cerrar_obra", [[id]]);
  }
}
```

Métodos útiles del servicio: `searchRead`, `search`, `read`, `create`, `write`, `unlink`, `searchCount`, `readGroup`, `webSearchRead` y el genérico `call(modelo, método, args, kwargs)`. Por debajo, todos hacen `POST /web/dataset/call_kw/<modelo>/<método>`.

8.2 La función `rpc` para controladores propios

```
import { rpc } from "@web/core/network/rpc";

const resumen = await rpc("/bim/obra/resumen", { obra_id: 12, incluir_partidas: true });
```

`rpc` construye el sobre JSON-RPC, envía la cookie de sesión, desenvuelve `result` y, si llega `error`, lanza un `RPCError` que el cliente web muestra como diálogo de error.

8.3 Fuera del backend (website, portal, scripts sueltos)

En páginas públicas o en JavaScript sin el framework, hay que construir el sobre a mano:

```
const resp = await fetch("/bim/obra/resumen", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ jsonrpc: "2.0", method: "call", params: { obra_id: 12 }, id:
Date.now() }),
});
const data = await resp.json();
if (data.error) throw new Error(data.error.data.message);
console.log(data.result);
```

8.4 Consejos de rendimiento

- Pide solo los campos que vas a pintar: `fields` corto, `limit` siempre.
- Agrupa en el servidor: un `readGroup` o un método propio es mejor que diez llamadas desde el navegador.
- Carga en `onWillStart` lo imprescindible y el resto bajo demanda.

Capítulo 9. Clientes en varios lenguajes

Cada ejemplo hace lo mismo con JSON-2 (recomendado en Odoo 20): leer las cinco primeras empresas. El cliente Python está en el capítulo 6.7.

9.1 curl

```
curl https://miempresa.example.com/json/2/res.partner/search_read \
-X POST \
--oauth2-bearer "$OD00_API_KEY" \
-H "X-Odoo-Database: miempresa" \
-H "Content-Type: application/json" \
-d '{"domain": [{"is_company", "=", true}], "fields": ["name"], "limit": 5}' \
--fail-with-body
```

9.2 JavaScript / Node.js (18+)

```
const BASE = "https://miempresa.example.com/json/2";
const headers = {
  "Content-Type": "application/json",
  Authorization: `bearer ${process.env.OD00_API_KEY}`,
  "X-Odoo-Database": "miempresa",
};

async function odoo(model, method, body = {}) {
  const res = await fetch(`${BASE}/${model}/${method}`, {
    method: "POST", headers, body: JSON.stringify(body),
  });
  const data = await res.json();
  if (!res.ok) throw new Error(`${res.status} ${data.name}: ${data.message}`);
  return data;
}

const empresas = await odoo("res.partner", "search_read", {
  domain: [{"is_company", "=", true}], fields: ["name"], limit: 5,
});
```

Ojo: `body` debe ir serializado con `JSON.stringify`; un objeto directo no se envía como JSON.

9.3 PHP (8+)

```
<?php
function odoo(string $model, string $method, array $body = []) {
    $ch = curl_init("https://miempresa.example.com/json/2/$model/$method");
    curl_setopt_array($ch, [
        CURLOPT_POST => true,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_HTTPHEADER => [
            'Content-Type: application/json',
            'Authorization: bearer ' . getenv('ODOO_API_KEY'),
            'X-Odoo-Database: miempresa',
        ],
        CURLOPT_POSTFIELDS => json_encode($body),
    ]);
    $out = json_decode(curl_exec($ch), true);
    $code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    if ($code >= 400) throw new Exception("$code {$out['message']}");
    return $out;
}

$empresas = odoo('res.partner', 'search_read', [
    'domain' => [['is_company', '=', true]], 'fields' => ['name'], 'limit' => 5,
]);
```

Ventaja añadida en PHP: JSON-2 no necesita la extensión XML-RPC, que desde PHP 8 no viene activada por defecto.

9.4 Herramientas y librerías

- **n8n**: el nodo de Odoo v2 con credencial de API key ya usa JSON-2.
- **Postman / Bruno**: basta una colección con la cabecera `Authorization` y una variable para el host.
- **Python**: `requests` es suficiente; hay librerías comunitarias para JSON-2 (por ejemplo, `odoo-json2`), pero revisa su mantenimiento antes de usarlas en producción. `odoorpc` y `xmlrpc.client` usan los endpoints obsoletos.

Capítulo 10. Errores, seguridad y rendimiento

La diferencia práctica más grande entre `/jsonrpc` y JSON-2 está en cómo llegan los errores: en el primero hay que inspeccionar el cuerpo; en el segundo basta el código HTTP.

10.1 Errores en JSON-RPC (cliente web y `/jsonrpc`)

```
{
  "jsonrpc": "2.0", "id": 1,
  "error": {
    "code": 200,
    "message": "Odoo Server Error",
    "data": {
      "name": "odoo.exceptions.AccessError",
      "message": "No tiene permiso para acceder a 'Contacto' (res.partner)...",
      "arguments": [...],
      "context": {},
      "debug": "Traceback (most recent call last): ..."
    }
  }
}
```

Decide qué hacer leyendo `error.data.name`, no `error.message`, que es genérico.

10.2 Excepciones frecuentes y qué significan

Excepción (<code>name</code>)	Código HTTP en JSON-2	Causa habitual	Qué hacer
<code>werkzeug.exceptions.Unauthorized</code>	401	API key inválida, caducada o revocada	Rotar la clave
<code>odoo.exceptions.AccessDenied</code>	401/403	Usuario o clave incorrectos en <code>login</code>	Revisar credenciales
<code>odoo.exceptions.AccessError</code>	403	Faltan permisos o una regla de registro lo impide	Ajustar grupos del usuario bot
<code>odoo.exceptions.MissingError</code>	404	El registro no existe o fue borrado	Usar <code>search_read</code> en vez de <code>search + read</code>
<code>odoo.exceptions.ValidationError</code>	422	Restricción <code>@api.constrains</code> o SQL	Corregir los datos enviados
<code>odoo.exceptions.UserError</code>	422	Regla de negocio (p. ej., confirmar un pedido sin líneas)	Mostrar el mensaje al usuario

Excepción (<code>name</code>)	Código HTTP en JSON-2	Causa habitual	Qué hacer
<code>TypeError</code>	422	Argumento con nombre incorrecto en JSON-2	Mirar la firma en <code>/doc</code>
<code>psycpg2.errors.SerializationFailure</code>	409/500	Conflicto de concurrencia	Reintentar con espera exponencial

Los códigos HTTP de JSON-2 de esta tabla son los habituales; confírmalos en tu versión antes de programar lógica que dependa de un número concreto.

10.3 Seguridad

- Siempre **HTTPS**. La API key equivale a la contraseña del usuario.
- **Usuario bot** por integración, con permisos mínimos y contraseña vacía.
- **Rotación** de claves cada tres meses como máximo (límite de Odoo) y revocación inmediata si se filtra.
- No registres en logs cuerpos completos con datos personales ni la cabecera `Authorization`.
- En producción: `list_db = False`, `dbfilter` por host y el gestor de bases fuera de internet.
- Recorta lo que devuelve `debug`: no muestres trazas a usuarios finales.

10.4 Rendimiento

- **Paginación**: `limit` + `offset`, o mejor por ID (`["id", ">", ultimo_id]`) para tablas grandes.
- **Campos mínimos** en `fields`. Evita campos calculados no almacenados en lecturas masivas.
- **Lotes**: `create` con `vals_list` de 100–500 registros en vez de una llamada por registro.
- **Sincronización incremental**: filtra por `write_date > última_sincronización`.
- **Reintentos** solo en errores transitorios (409, 429, 502, 503, 504), con *backoff* exponencial.
- **Timeouts** explícitos en el cliente (30–60 s) para no dejar procesos colgados.

Capítulo 11. Caso práctico: mediciones de obra desde una app externa

Escenario: una app de campo envía a Odoo 20 las mediciones de una obra (partida, cantidad, fecha) y necesita que se registren todas o ninguna. La solución es un método de negocio en un módulo propio, llamado una sola vez por JSON-2. Los nombres de modelo (`bim.obra` , `bim.medicion`) son ilustrativos.

11.1 El modelo y el método atómico

```
# bim_mediciones/models/bim_medicion.py
from odoo import api, fields, models
from odoo.exceptions import UserError, ValidationError

class BimMedicion(models.Model):
    _name = "bim.medicion"
    _description = "Medición de obra"

    obra_id = fields.Many2one("bim.obra", required=True, index=True)
    partida = fields.Char(required=True)
    cantidad = fields.Float(required=True)
    fecha = fields.Date(required=True)
    ref_externa = fields.Char(index=True)

    _ref_unica = models.Constraint("unique(obra_id, ref_externa)", "Medición duplicada")

@api.model
def registrar_lote(self, obra_id, lineas):
    """Registra todas las líneas o ninguna. Devuelve los IDs creados."""
    obra = self.env["bim.obra"].browse(obra_id).exists()
    if not obra:
        raise UserError(f"La obra {obra_id} no existe")
    if obra.state == "cerrada":
        raise UserError("La obra está cerrada")
    vals_list = []
    for l in lineas:
        if l["cantidad"] <= 0:
            raise ValidationError(f"Cantidad no válida en {l['partida']}")
        vals_list.append({"obra_id": obra.id, **{k: l[k] for k in
            ("partida", "cantidad", "fecha", "ref_externa")}})
    return self.create(vals_list).ids
```

Si alguna línea falla, la excepción provoca el *rollback* de toda la llamada: no quedan mediciones a medias. La restricción única sobre `ref_externa` hace la operación **idempotente**: si la app reintenta tras un corte de red, no se duplican datos. (En versiones anteriores a Odoo 18 la restricción se declara con `_sql_constraints`.)

11.2 La llamada desde la app

```
ids = odoo("bim.medicion", "registrar_lote",
    obra_id=12,
    lineas=[
        {"partida": "01.02 Excavación", "cantidad": 42.5,
         "fecha": "2026-10-30", "ref_externa": "app-7781"},
        {"partida": "02.01 Hormigón HA-25", "cantidad": 18.0,
         "fecha": "2026-10-30", "ref_externa": "app-7782"},
    ]
)
```

Es una sola petición `POST /json/2/bim.medicion/registrar_lote`. Al estar decorado con `@api.model`, no lleva `ids` y los argumentos van por nombre.

11.3 Permisos del usuario bot

1. Crear el usuario `app-campo@miempresa.com` sin contraseña.
2. Darle un grupo propio, p. ej. *BIM / Mediciones (API)*, con `ir.model.access` de lectura en `bim.obra` y de creación en `bim.medicion`. Sin permiso de borrado.
3. Añadir una regla de registro si solo debe ver ciertas obras.
4. Generar su API key con caducidad de 90 días y guardarla en el gestor de secretos de la app.
5. Programar un recordatorio para rotarla antes de que caduque.

11.4 Lectura incremental para el panel externo

```
cambios = odoo("bim.medicion", "search_read",
    domain=[["write_date", ">", "2026-10-29 00:00:00"]],
    fields=["obra_id", "partida", "cantidad", "fecha"],
    order="write_date asc", limit=500)
```

Guarda el `write_date` más alto recibido y úsalo en la siguiente sincronización. Las fechas viajan en UTC.

Apéndices

A. Chuletarío

Quiero...	Cliente web (OWL)	/jsonrpc (obsoleto)	JSON-2 (Odoo 20)
Buscar y leer	<code>orm.searchRead(m, dom, fields)</code>	<code>execute_kw(db, uid, key, m, "search_read", [dom], {"fields": f})</code>	POST <code>/json/2/m/search_read</code> <code>{domain, fields}</code>
Crear	<code>orm.create(m, [vals])</code>	<code>execute_kw(..., "create", [vals])</code>	POST <code>/json/2/m/create</code> <code>{vals_list}</code>
Modificar	<code>orm.write(m, ids, vals)</code>	<code>execute_kw(..., "write", [ids, vals])</code>	POST <code>/json/2/m/write</code> <code>{ids, vals}</code>
Borrar	<code>orm.unlink(m, ids)</code>	<code>execute_kw(..., "unlink", [ids])</code>	POST <code>/json/2/m/unlink</code> <code>{ids}</code>
Método de negocio	<code>orm.call(m, "action_x", [ids])</code>	<code>execute_kw(..., "action_x", [ids])</code>	POST <code>/json/2/m/action_x</code> <code>{ids}</code>
Ruta propia	<code>rpc("/ruta", params)</code>	—	<code>type='jsonrpc' +</code> <code>auth='bearer'</code>
Versión	—	<code>common.version()</code>	GET <code>/web/version</code>
Mi usuario	<code>user.userId</code>	<code>common.login(...)</code>	POST <code>/json/2/res.users/context_get</code>

B. Códigos de error JSON-RPC 2.0

Código	Significado
-32700	JSON mal formado
-32600	Petición inválida
-32601	Método no encontrado
-32602	Parámetros inválidos
-32603	Error interno
200	Código que usa Odoo para cualquier excepción de servidor; el detalle está en <code>error.data</code>

C. Checklist de migración a Odoo 20

- ☐ Inventariar todas las integraciones y qué endpoint usan (`/xmlrpc` , `/xmlrpc/2` , `/jsonrpc` , JSON-2).
- ☐ Buscar llamadas al servicio `db` y sustituirlas por `/web/database/*` o `/web/version` (bloqueante en Odoo 20).
- ☐ Renombrar `type='json'` a `type='jsonrpc'` en los controladores de tus módulos.
- ☐ Crear usuarios bot y API keys con caducidad; documentar la rotación.
- ☐ Migrar `execute_kw` a JSON-2 traduciendo argumentos posicionales a nombres (comprobar en `/doc`).
- ☐ Cambiar el control de errores: de mirar `error` en el cuerpo a mirar el código HTTP.
- ☐ Agrupar secuencias de llamadas que deben ser atómicas en un método de negocio.
- ☐ Probar todo en un staging de Odoo 20 y revisar los avisos del logger `odoo.addons.rpc.controllers.jsonrpc` .
- ☐ Planificar la retirada total de `/jsonrpc` antes de Odoo 22 (otoño 2028).

Fuentes

- [Odoo 20 — External RPC API](#)
- [Odoo 19 — External JSON-2 API y migración desde XML-RPC/JSON-RPC](#)
- [Odoo saas-19.3 — gestión programática de API keys](#)
- [Foro Odoo — qué se rompe en Odoo 20 y qué aguanta hasta Odoo 22](#)
- [Foro Odoo — type='json' obsoleto en Odoo 19](#)
- [Oduist — resumen de la charla «XMLRPC is dead. All Hail JSON-2»](#)