Connettere mondi con Odoo: webhook in azione

Odoo Days Italia

Napoli, 26 e 27 maggio 2025

Raffaele **Amalfitano** - Simone **Tullino**

UNITIVA OCA ASSOCIAZIONE ODOO ITALIA

Chi siamo

Unitiva è una **software house** con oltre 15 anni di esperienza, specializzata in **ERP e Digital Solutions**.

Come **Gold Partner Odoo,** ci impegniamo a offrire **valore** ai nostri clienti, adottando un approccio orientato all'ascolto e sviluppando **soluzioni su misura**.

Raffaele Amalfitano
Odoo Team Leader

Simone Tullino
Odoo Developer

Agenda

01
Cos'è un webhook

02
API vs webhook

03
Webhook in uscita

04
Webhook in ingresso

05
Sicurezza

06
Logging

07
Performance

01 Cos'è **un webhook?**

Un **webhook** è un meccanismo che consente a due sistemi di **comunicare in tempo reale**: quando in un sistema si verifica un **evento** (es. creazione di un ordine), **viene inviata automaticamente una notifica** all'altro, senza bisogno di richieste manuali o continue.

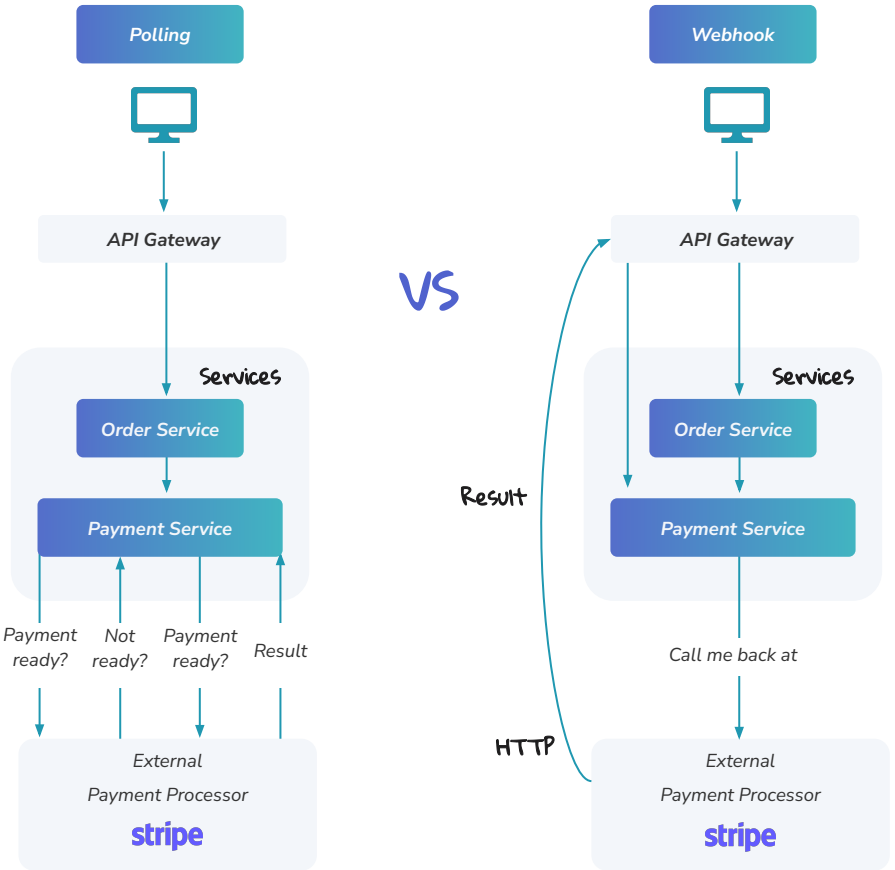
I webhook funzionano come una **notifica push**: trasmettono subito le informazioni appena accade qualcosa, in modo automatico ed istantaneo.

*É un po' come ricevere **una mail o un messaggio su Whatsapp**: non serve controllare ogni volta la casella di posta o l'app per vedere se ci sono novità, la notifica arriva da sola.*

02 API vs **Webhook**

Da un punto di vista operativo:

- **API:** modello basato su richiesta-risposta. Un sistema deve interrogare continuamente l'altro (polling) per sapere se ci sono novità, con un processo ripetitivo per ottenere informazioni.
- **Webhook**: modello basato su evento-risposta. Il sistema interessato alla ricezione di un determinato evento, comunica una callback-url per essere notificato.



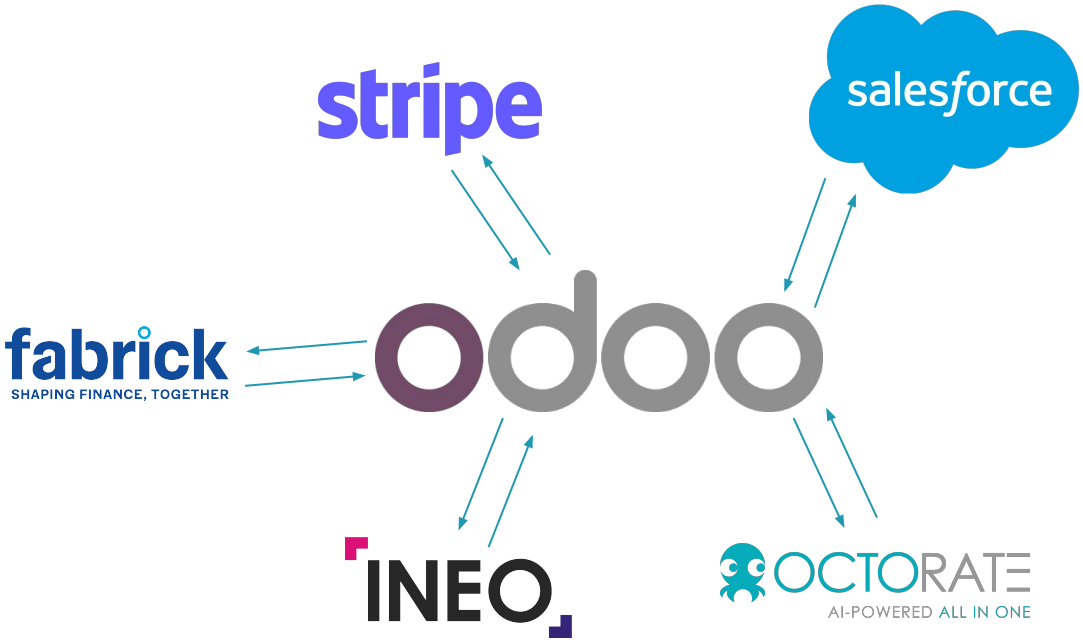
API vs **Webhook**

Da un punto di vista di efficienza dei costi:

- **API**: il polling continuo genera un carico costante sulla rete e sul processore, aumentando i costi operativi. Richieste ripetute anche in assenza di cambiamenti.
- **Webhook**: basati su eventi, riducono il numero di richieste, ottimizzando costi e risorse. Comunicazione immediata e mirata.

Le nostre **integrazioni**

- Booking engine & Channel Manager
- Client onboarding & AML compliance
- Open banking & financial services
- Payment gateway
- CRM & Sales automation



03 Webhook in uscita: **Odoo genera eventi**

Quando si verifica un evento, come la conferma di un ordine o la creazione di una fattura, **Odoo notifica automaticamente** altri sistemi.

I vantaggi?

- *automazione senza interventi manuali*
- *aggiornamenti in tempo reale*
- *reattività ed efficienza del sistema*

```python
WEBHOOK_URL = "https://unitiva.com/receive"
class SaleOrder(models.Model):
    _inherit = 'sale.order'

    @api.model
    def create(self, vals):
        order = super(SaleOrder, self).create(vals)
        self._send_order_webhook(order)
        return order

    def _send_order_webhook(self, order):
        try:
            payload = {
                'id': order.id,
                'name': order.name,
                'partner': order.partner_id.name,
                'amount_total': order.amount_total,
                'date_order': order.date_order.isoformat() if order.date_order else None,
                'lines': [
                    {
                        'product': line.product_id.name,
                        'quantity': line.product_uom_qty,
                        'price_unit': line.price_unit,
                    } for line in order.order_line
                ]
            }
            headers = {'Content-Type': 'application/json'}
            response = requests.post(WEBHOOK_URL, data=json.dumps(payload), headers=headers)

            if response.status_code != 200:
                logger.warning("Webhook POST failed with status %s: %s", response.status_code, response.text)
        except Exception as e:
            logger.error("Error sending webhook: %s", str(e))
```

04 Webhook in ingresso: **Odoo riceve eventi**

Immaginiamo di voler registrare una fattura su Odoo, integrato con la piattaforma di pagamento Stripe.

Quando **emetti una fattura in Odoo**, parte tutto in automatico: viene creato un **payment intent su Stripe** con l'importo, il cliente e il riferimento alla fattura.

Non appena il pagamento va a buon fine, Stripe invia un **webhook** a Odoo.

Odoo riceve la notifica e **registra il pagamento** in tempo reale, senza bisogno di interventi manuali.

```python
@http.route('/webhook/stripe/payment_intent_management', type='json', auth='none', csrf=False, methods=['POST'])
def stripe_webhook(self, **kwargs):
    try:
        payload = request.httprequest.get_data()
        data = json.loads(payload)
        _logger.info("Webhook received: %s", data)
    except Exception:
        return http.Response("Invalid payload", status=400)

    # Custom processing based on event type
    event_type = data.get('type')
    if event_type == 'payment_intent.succeeded':
        payment_intent = data.get('data', {}).get('object', {})
        payment_intent_id = payment_intent.get('id')
        amount_received = payment_intent.get('amount_received')
        currency = payment_intent.get('currency')
        customer_email = payment_intent.get('receipt_email')
        metadata = payment_intent.get('metadata', {})

        _logger.info("PaymentIntent succeeded: %s", payment_intent_id)

        # Creating an internal log or custom record
        request.env['stripe.payment.log'].sudo().create({
            'stripe_payment_intent': payment_intent_id,
            'amount': amount_received / 100.0,
            'currency': currency,
            'customer_email': customer_email,
            'raw_data': json.dumps(data),
        })


    return {"status": "ok"}
```

05 Sicurezza
Gestione IP

Prima di gestire un evento, è essenziale verificare che la richiesta arrivi davvero da Stripe.

Una semplice chiamata HTTP **può essere simulata**, quindi il sistema controlla l'indirizzo IP del mittente per evitare accessi non autorizzati.

Cosa succede qui?
- Recuperiamo l'**IP reale del client**
- Lo **confrontiamo con la lista** ufficiale di Stripe
- Se non è presente → **HTTP 403: Forbidden**

Un meccanismo semplice, ma efficace: blocca chi non è autorizzato e protegge l'integrità del sistema.

```python
# Stripe official webhook IP addresses
WHITELISTED_IPS = [
    '3.18.12.63', '3.130.192.231', '13.235.14.237',
    '13.235.122.149', '18.211.135.69', '35.154.171.200',
    '52.15.183.38', '54.88.130.119', '54.88.130.237',
    '54.187.174.169', '54.187.205.235', '54.187.216.72'
]

# Check if the request comes from a trusted IP
client_ip = request.httprequest.remote_addr
_logger.info("Client IP: %s", client_ip)
if client_ip not in WHITELISTED_IPS:
    return http.Response("Unauthorized IP address", status=403)
```

Sicurezza

Firma

Per verificare la firma del webhook - garantendo che il messaggio provenga davvero da Stripe - usiamo una **chiave segreta** per ricostruire e confrontare la firma.

```python
# Get raw payload and Stripe signature header
payload = request.httprequest.get_data()
sig_header = request.httprequest.headers.get('Stripe-Signature')
_logger.info("Stripe-Signature Header: %s", sig_header)
webhook_secret = request.env['ir.config_parameter'].sudo().get_param('WEBHOOK_SIGNING_SECRET')
if not webhook_secret:
    _logger.error("Missing webhook signing secret in config parameters.")
    return http.Response("Server misconfiguration", status=500)
webhook_secret = webhook_secret.encode()  # Convert string to bytes

# Verify the Stripe signature
try:
    parts = dict(x.split('=') for x in sig_header.split(','))
    timestamp = parts.get('t')
    signature = parts.get('v1')
except Exception:
    _logger.warning("Malformed Stripe-Signature header.")
    return http.Response("Malformed signature header", status=400)
```

Cosa succede qui?

- **Verifica della firma**:
  usiamo la chiave segreta salvata in Odoo per validare il messaggio.
- **Error handling**:
  se la chiave è assente → errore 500;
  se la firma è sbagliata → messaggio scartato.

In sintesi: solo i messaggi firmati correttamente entrano nel sistema!

Sicurezza

Replay attack

Anche un messaggio firmato può essere intercettato e reinviato: si tratta di un replay attack.

Per evitarlo, Stripe firma anche un timestamp (t=) che ci permette di verificare se il messaggio è troppo vecchio.

```
# Check for replay attack (5 min validity)
if abs(time.time() - int(timestamp)) > 300:
    _logger.warning("Timestamp expired: %s", timestamp)
    return http.Response("Timestamp expired", status=400)
```

Cosa succede qui?

- Stripe firma anche l'**orario di invio**
- Se sono passati **più di 300 secondi**, lo scartiamo
- Questo evita che i **messaggi vecchi** vengano processati di nuovo!

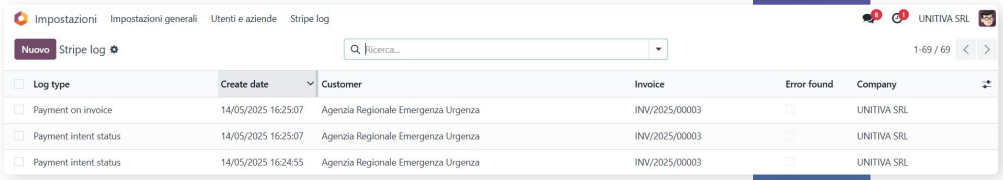
Una firma valida non è sufficiente: se il messaggio è troppo vecchio, viene considerato non affidabile.

06 Logging

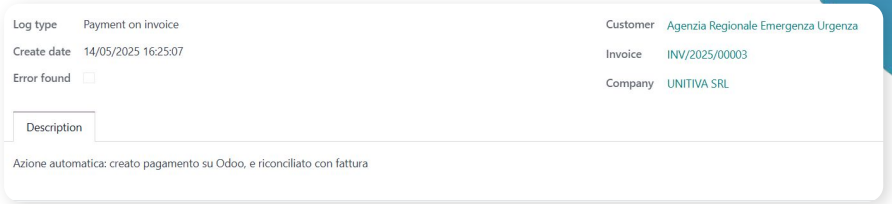
Quando riceviamo un webhook da Stripe, Odoo lo elabora e registra l'evento nello **Stripe Log**: un menù realizzato per tracciare gli eventi.

Ogni riga mostra:

- tipo di evento
- data/ora
- cliente
- fattura (se presente)
- azienda
- errori riscontrati
- una descrizione chiara dell'azione eseguita, come: "**Pagamento creato su Odoo e riconciliato con fattura.**"



Un punto di accesso immediato per verificare cosa è successo, senza leggere i log di sistema e senza dubbi sul risultato.



```python
class StripeLog(models.Model):
    _name = "stripe.log"
    _description = "Stripe log"
    _rec_name = "create_date"

    log_type = fields.Selection([
        ('customer', 'Customer'),
        ('stripe_payment', 'Stripe payment'),
        ('payment_on_invoice', 'Payment on invoice'),
        ('payment_status', 'Payment intent status'),
        ('stripe_dispute', 'Stripe dispute'),
        ('stripe_dispute_rc', 'Stripe dispute RC task'),
        ('stripe_fee', 'Stripe fee')
    ], string="Log type", readonly=True, copy=False)

    partner_id = fields.Many2one('res.partner', string="Customer",
                                 readonly=True, copy=False)
    invoice_id = fields.Many2one('account.move',
                                 string="Invoice", readonly=True, copy=False)
    description = fields.Text(string="Description", readonly=True)
    error_found = fields.Boolean(string="Error found", default=False, readonly=True, copy=False)
    company_id = fields.Many2one('res.company', string="Company", readonly=True, copy=False)
```

```python
def _create_stripe_log(self, log_type, customer_obj, description, company_id, invoice_id=False, error_found=False):
    """
    Used to create stripe log
    """
    self.env['stripe.log'].sudo().create({
        'log_type': log_type,
        'partner_id': customer_obj.id,
        'invoice_id': invoice_id.id if invoice_id else False,
        'description': description,
        'company_id': company_id.id,
        'error_found': error_found
    })
```

```python
import logging
_logger = logging.getLogger(__name__)
```

```python
        if found_intent:
            # **6. update odoo invoice **
            invoice.write({
                'stripe_payment_intent_id': found_intent.id,
                'stripe_payment_intent_status': found_intent.status,
                'stripe_validation_status': 'waiting_stripe_payment_verification',
                'stripe_processing': False  # stripe processing flag reset
            })
            self.env.cr.commit()
            invoice.with_user(SUPERUSER_ID).message_post(body=f"Intent di pagamento recuperato da Stripe tramite procedura di allineamento: {found_intent.id}")
            _logger.info("Intent di pagamento {} recuperato da Stripe tramite procedura di allineamento per fattura {}".format(
                found_intent.id, invoice.name))
        else:
            # **7. payment intent not found, reset stripe processing flag to enable invoice reprocessing **
            invoice.write({'stripe_processing': False})
            self.env.cr.commit()
            invoice.with_user(SUPERUSER_ID).message_post(body="Nessun intent di pagamento trovato su Stripe tramite procedura di allineamento."\
                " Fattura sbloccata e pronta per essere riprocessata.")
            _logger.info("Nessun intent di pagamento trovato su Stripe tramite procedura di allineamento."\
                " Fattura {} - ID {} sbloccata e pronta per essere riprocessata.".format(invoice.name, invoice.id))
    except Exception as e:
        # self.env.cr.rollback()  # rollback
        invoice.with_user(SUPERUSER_ID).message_post(body="Errore '{}' nel recupero dell'Intent di Pagamento per fattura {}".format(
            e, invoice.name))
        _logger.info("Errore '{}' nel recupero dell'Intent di Pagamento per fattura {}".format(
            e, invoice.name))
```

07 Performance

- Spostamento logica di business all'interno del job
- Creazione code di lavoro
- Esecuzione asincrona


```python
def handle_stripe_event(self, event_type, webhook_object):

    _logger.info("Esecuzione del job asincrono per l'evento: %s", event_type)

    # Handle payment_intent events
    if webhook_object.get('object') == 'payment_intent':
        pi_id = webhook_object['id']
        move = self.env['account.move'].sudo().search(
            [('stripe_payment_intent_id', '=', pi_id)], limit=1
        )
        if move:
            self._stripe_payment_intent_status_mgmt(move, webhook_object)
    # Handle charge events
    elif event_type in ['charge.pending', 'charge.succeeded', 'charge.failed', 'charge.updated']:
        pi_id = webhook_object.get('payment_intent')
        move = self.env['account.move'].sudo().search(
            [('stripe_payment_intent_id', '=', pi_id)], limit=1
        )
        if move:
            self._stripe_charge_mgmt(move, webhook_object, event_type)
    # Handle charge disputes
    elif event_type == 'charge.dispute.created':
        charge_id = webhook_object.get('charge')
        move = self.env['account.move'].sudo().search(
            [('stripe_payment_intent_charge_id', '=', charge_id)], limit=1
        )
        if move:
            self._stripe_charge_mgmt(move, webhook_object, event_type)
```

```python
class StripeWebhookController(http.Controller):

    @http.route('/webhook/stripe/payment_intent_management', type='json', auth='none', methods=['POST'], csrf=False)
    def webhook_stripe_payment_intent(self, *args, **kwargs):
        """
        Secure Stripe webhook endpoint for events:
        - payment_intent.succeeded / canceled / requires_action
        - charge.pending / succeeded / failed / updated
        - charge.dispute.created
        """


        # # Get raw payload and Stripe signature header
        payload = request.httprequest.get_data()

        # # Parse the payload as JSON
        try:
            webhook_data = json.loads(payload)
        except Exception:
            _logger.warning("Invalid JSON payload.")
            return http.Response("Invalid JSON payload", status=400)

        # Extract event information
        event_type = webhook_data.get('type')
        webhook_object = webhook_data.get('data', {}).get('object', {})

        stripe_int_lib = request.env['stripe.integration.lib'].sudo()

        # Accoda il job asincrono
        stripe_int_lib.with_delay().handle_stripe_event(event_type, webhook_object)
```

Grazie per l’attenzione!

raffaele.amalfitano@unitiva.it
simone.tullino@unitiva.it

odoo-italia.org

associazioneodooitalia@gmail.com