



TECHNICAL WHITE PAPER

An ERPNext Message Bus

Real-time updates between an ERP, several online stores and the systems around them — what we built, how it behaves, and the open-source version that follows.

For technical leads and operations owners running ERPNext alongside eCommerce.

1. Summary

A retailer runs ERPNext as its system of record and sells through several online stores. We connected them with a RabbitMQ message bus so that a price or stock change made in the ERP reaches every store without anyone exporting, importing or waiting for a nightly sync — and so that every order placed in a store becomes a sales order in the ERP.

- **ERP → stores, as it happens.** Prices, wholesale prices, dated pricing rules, stock levels, product and image changes, product removals, categories and shipment tracking are published when the ERP record changes and applied by each store when the message arrives. There is no polling interval.
- **Stores → ERP.** Each order arrives as a message and becomes a customer, addresses, a sales order with the right taxes and shipping, and a payment. Later events about the same order, such as a fraud-screening decision, update it.
- **One event, many subscribers.** Every store has its own queue and keeps only what applies to it. Other systems subscribe to the same events the same way; the client's point-of-sale system, a separate project, is one of them.
- **Operable by non-developers.** Message-to-handler mappings are edited in an ERPNext screen, which store events get published is chosen in a store admin screen, and both sides keep logs that staff can read and act on.

HOW TO READ THIS PAPER

Sections 2–6 describe the system as it runs in production. Where we give timings or effort (sections 4 and 8) they are **engineering estimates**, labelled as such. Section 7 introduces the open-source version we are building from this work.

2. The problem with the usual integrations

An ERP surrounded by sales channels is usually integrated in one of two ways, and both wear badly as the business grows.

Scheduled sync

A job runs every 15 minutes, hour or night and copies prices and stock across. Between runs the store is wrong: it sells stock that is gone, or shows yesterday's price. Shortening the interval makes every run compete with real work on the ERP database.

Direct API calls

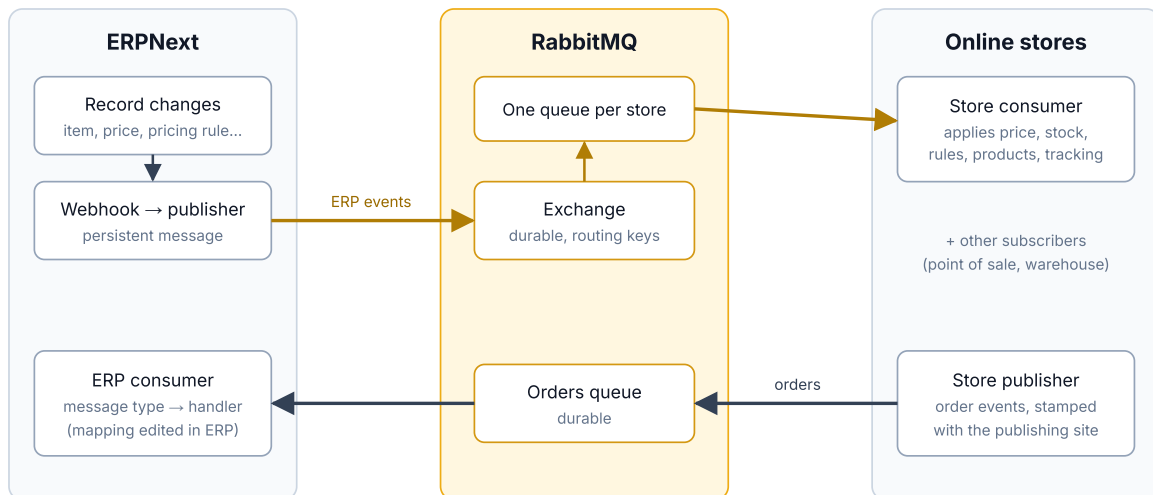
Each store calls the ERP when something happens. Now the ERP must be up and fast at the moment a customer checks out, every store needs its own integration code, and a call that fails halfway leaves no trace to retry from.

What the business actually needs

- A change made once in the ERP shows up everywhere it matters, quickly, without anyone remembering to push it.
- Stores keep taking orders when the ERP is restarting, upgrading or simply busy.
- Adding a store, or another system that cares about the same data, does not mean another round of point-to-point integration.
- When something fails, a person can see what failed and run it again.

3. Architecture

Systems do not call each other. They publish messages to RabbitMQ and consume messages from it. The broker holds whatever has not been processed yet.



Gold: changes flowing out of the ERP. Dark: orders flowing in.

Out of the ERP

ERPNext's own webhooks fire when a record such as an item, an item price or a pricing rule changes. They call a small publishing endpoint in our ERPNext app, which puts the event on a durable exchange as a persistent message with a routing key. Nothing about the ERP's normal behaviour is modified; the bus listens to events ERPNext already raises.

Into each store

Every store consumes from its own queue. A message carries its type, its payload, an id and who made the change. The store picks the processor for that type and applies it. Because the same event is delivered to every store, each store filters for itself: it applies stock only for its own sales channel and country, and prices only from its own price list.

Out of each store

The store's publisher is configured, not coded: an administrator picks which store events to publish (for example, an order being placed) and gives each one an exchange and routing key. Messages are stamped with the publishing site so the ERP knows where an order came from.

Into the ERP

A long-running worker consumes the ERP's queue one message at a time and looks up each message type in a mapping table kept in ERPNext: message type on one side, handler function on the other. If the broker connection drops, the worker reconnects on its own and tells the team's chat channel.

4. Real-time updates

“Real time” here means event-driven: a change is published when it is saved and applied when it is received. Nothing waits for a schedule.

Change made in the ERP	What each store does on receipt
Item price	Sets the price of that variant, if the price list is the store's own.
Wholesale price	Sets the price and the matching wholesale price rule.
Pricing rule or discount	Replaces the rule by its id — for a product, a category or a brand — with its start date, end date and priority. Rules are stored with their dates rather than baked into prices, so a promotion starts and ends on time without a second message.
Stock level	Updates quantity on hand per warehouse and per country, for that store's sales channel only.
Product created or edited	Creates or updates the product and its variants, custom attributes and specification labels.
Product images	Attaches the images to the product on the sites they were published for.
Product discontinued	Removes it from the store, if it belongs to that store's price list.
Category or tag renamed or merged	Applies the same rename or merge in the store's catalogue.
Bulk re-categorisation	Moves a list of products to a new category or brand, in batches of 500.
Order shipped	Adds the shipment, carrier and tracking number to the customer's order.

The same events serve other subscribers. A point-of-sale system, for instance, can take price, stock, pricing-rule, tax and product events from the bus and push them straight on to its terminals; in this client's case that system is a separate project with its own consumer.

How long does a change take to arrive? (estimate)

The design tells us where the time goes: there is no polling interval anywhere in the path, so the delay is the sum of a few short hops.

Hop	What happens	Estimated time
1. ERP save → webhook	ERPNext hands the webhook to its background workers. This is the largest and most variable part; it grows when the ERP's job queue is busy.	1 – 5 s
2. Publish	The publisher connects to the broker and publishes a persistent message.	0.1 – 0.5 s
3. Broker	RabbitMQ routes the message to each bound queue.	< 0.05 s
4. Store applies it	The store's consumer parses the message and writes the change to its database.	0.1 – 1 s
Typical end to end, ERP save to visible in the store		about 2 – 7 s

WHAT THAT ESTIMATE MEANS

For comparison, a scheduled sync is wrong for half its interval on average: about 7 minutes for a 15-minute job, and 12 hours for a nightly one. Moving from schedules to events turns “within the hour” into “within seconds”. If a store or the ERP is down, the figure above stops applying: messages wait in their durable queue and are applied when the consumer returns.

Estimates are derived from the architecture and from typical figures for ERPNext background jobs and RabbitMQ on a local network. Bulk operations that change thousands of records at once take proportionally longer, because each change is its own message.

5. Orders into the ERP

When a customer places an order in any of the stores, the store publishes it. The ERP handler then does, in order:

1. **Logs the raw message first**, in an orders log inside ERPNext, together with the site it came from. Whatever happens next, the original is kept.
2. **Works out the context from the publishing site**: which company, price list and warehouse this store sells from.
3. **Finds or creates the customer**. Existing customers, contacts and addresses are matched on name, phone, email and address details, so a returning buyer is not duplicated. Addresses carry the store's address-verification result.
4. **Builds and submits the sales order**: the store's order number as the reference, each line looked up by SKU, shipping and long-distance charges as their own lines, discounts reconciled against the store's grand total, and the tax template chosen for the destination.
5. **Records the payment** according to how the customer paid: card payments are captured through the payment gateway, wallet payments are recorded and held until confirmed, and bank transfers are flagged for follow-up.
6. **Links the log entry to the sales order** and runs a verification pass over what was created.

Events that arrive later for the same order — most importantly the fraud-screening decision — find the sales order and update it.

When an order fails

A missing SKU or a locked record should not need a developer. The error is written to ERPNext's error log and posted to the team's chat channel. The orders log has a **Re-Sync Order** button: it re-runs the stored message, and if the sales order already exists it reconciles the contact, addresses, items and payment instead of creating a second one. From the store side, an administrator can also re-publish an order by its order number.

6. Operating it day to day

In the ERP

A settings screen holds the broker connection and the message-type-to-handler mapping. An orders log shows every incoming order message, the sales order it produced, and a re-sync button.

In each store

An admin page shows the message log with a reason for every entry (success, warning, error, "not sold on this site"), queue status, and tools to purge or inspect by SKU.

Feedback to the person who made the change

When a store cannot apply a change, it can publish a result message back saying what failed, whether it was a user error, and who requested the change — so the ERP user hears about it rather than a developer.

Alerts

The ERP worker posts to the team's chat when it starts, loses its connection, reconnects, or fails to process a message.

Adding a store

A new store gets its own queue bound to the same exchange, plus three settings: its sales channel, its country and its price list. From then on it receives every ERP event and keeps what applies to it. No change is made to the ERP side.

Adding a message type

On the ERP side: write the handler and add one mapping row. On the store side: add a processor for the new type. Existing flows are untouched.

7. The open-source version

The production system proved the pattern. We are now building it as a general ERPNext app, so that any ERPNext site can publish and consume messages the same way, with the business rules left to you.

Capability	What it gives you
Automatic retries	A message that cannot be processed is retried with back-off, then parked in a dead-letter queue for review.
Standard message envelope	Every message carries an id, type, version, source and timestamp, and is de-duplicated on its id.
Transactional outbox	Outbound events are written in the same transaction as the change they describe and confirmed by the broker.
One settings screen	Connection, exchanges, queues and routing are configured in one place inside ERPNext.
Topology from configuration	The app declares its own exchanges, queues and bindings; nothing is set up by hand.
Bring your own handlers	The bus ships with no business rules. You map each message type to your own handler.
Tests, docs and examples	A test suite, an install guide and worked examples for common flows.

Availability

The app is in active development and is intended for release under the MIT licence. If you would like one email when it is published, the case study page on wizardlabz.com has a sign-up.

8. What it takes to adopt (estimate)

For an organisation already running ERPNext with one online store, a first production rollout typically looks like this. The ranges are indicative; they depend mostly on how clean the product data is and how many order edge cases the store has.

Phase	What gets done	Estimate
1. Discovery and message contract	Agree which changes matter, what each message contains, and which system owns each piece of data.	1 – 2 weeks
2. Broker and ERP app	RabbitMQ with durable queues, the ERPNext app installed and configured, monitoring and alerts.	1 week
3. Real-time updates out	Price, stock and product events from the ERP applied in the store.	2 – 3 weeks
4. Orders in	Order messages into sales orders, customers, taxes and payments, with logging and re-sync.	2 – 4 weeks
5. Hardening and handover	Failure drills, replay, operator screens, documentation and training.	1 – 2 weeks
First store live		7 – 12 weeks
Each additional store on the same platform		1 – 2 weeks
Another kind of subscriber (point of sale, warehouse, reporting)		scoped separately

These are planning estimates for a typical case, not a quote. A store platform we have not integrated before adds time to phases 3 and 4.

Is a bus the right answer for you?

- **Probably yes** if you have more than one channel reading from the ERP, if stale prices or stock cost you sales or refunds, or if orders are re-keyed by hand when an integration fails.
- **Probably not yet** if you have one store, low order volume and a nightly sync nobody complains about. A broker is one more thing to run.

Talk to us about your setup

Wizard Labz builds, fixes and scales the systems businesses run on. Tell us what your ERP needs to talk to at wizardlabz.com and we will tell you honestly whether a message bus is worth it.