I

[Manual Nº 01]

n8n

El 20% que te permite construir automatizaciones de nivel profesional

NODEVELOP.COM · MMXXVI · MUESTRA V1.0

MUESTRA PÚBLICA · SECCIONES 0 + 1

Sobre esta muestra

Esto es la muestra pública del Manual Nº 01 — n8n. Contiene las dos primeras secciones completas, tal como aparecen en la edición vendida:

- **Sección 0 — Conceptos base.** El colchón de vocabulario funcional sobre el que se apoya el resto del manual: array, JSON, HTTP/REST, webhooks, Docker, expresiones de n8n, LLMs y prompts, deuda técnica y nodo.
- **Sección 1 — Setup & mental model.** Las tres formas reales de correr n8n en 2026 (Cloud, Docker self-hosted, npx local), el primer workflow Hello World, y la regla central de la plataforma: todo lo que fluye entre nodos es un array de items.

El manual completo continúa con siete secciones más —arquitectura modular, nodos esenciales, datos y transformaciones, manejo de errores, integración con LLMs, agentes y deployment— para un total de 84 páginas. Junto al PDF se entregan 5 workflows ejecutables en formato `.json`, 5 system prompts curados con sus variantes para Claude, GPT y Gemini, y los archivos de infraestructura (Docker Compose, Caddyfile, script de backup) del anexo.

Las páginas de esta muestra son idénticas a las del manual completo en densidad, tipografía y nivel de detalle. Si después de leerlas no sientes que el resto vale los $29.990 CLP, no lo compres.

---

paretoseries.com · Manual Nº 01 — n8n · v1.0 · MMXXVI

0 Conceptos base

Si manejas arrays, JSON, HTTP/REST, webhooks, Docker, expresiones de n8n y conceptos de LLMs, salta a §1.

Esta sección es un colchón de vocabulario. Existe para que un lector con base técnica heterogénea entre al resto del manual con el mismo piso conceptual. Cada entrada es una definición funcional, no un tutorial. Si dominas el concepto, no la leas.

0.1 Array

Un array es una colección ordenada de elementos accesibles por índice numérico que empieza en cero.

En n8n, todo lo que fluye entre nodos es un array de items, no un item suelto. Cuando un nodo en modo "una vez por item" recibe cien items, ejecuta su lógica cien veces; cuando recibe cero, simplemente no ejecuta esa lógica. Confundir "un item" con "un array de items" es la fuente del 80% de los errores de principiante en n8n: se escribe la expresión asumiendo un objeto cuando en realidad hay que iterar, o se intenta iterar cuando ya hay un solo elemento. Cuando leas `$input.all()` en una expresión, asume array hasta que la evidencia diga lo contrario; cuando leas `$json`, asume el item actual de esa iteración.

0.2 JSON

JSON es un formato de intercambio de datos en texto plano, con cuatro tipos primitivos (string, number, boolean, null) y dos estructuras (objeto `{}` y arreglo `[]`).

Es la lengua franca de las APIs modernas y la moneda de cambio entre nodos en n8n: cada item que circula por un workflow es un objeto JSON, y cada expresión opera sobre esa estructura. Las expresiones acceden a los campos con notación de punto, `$json.user.email`, y si los datos vienen anidados, encadenas: `$json.payload.items[0].id`. JSON no tiene fechas como tipo nativo (vienen como string ISO 8601) ni números enteros separados de decimales (todo es `number`), detalle que importa cuando una API devuelve un timestamp y tu lógica espera un objeto `Date`. Si te equivocas en una clave, n8n devuelve `undefined` sin avisar y la lógica río abajo silenciosamente trabaja sobre vacío; verificarlo en el panel del nodo es parte del oficio, no un paso opcional.

Si quieres entrar a fondo en este concepto, ver: Base Nº 02 — JSON profundo (planeado).

0.3 HTTP y APIs REST

HTTP es el protocolo de petición y respuesta sobre el que se comunican navegadores, servidores y APIs. REST es un estilo arquitectónico que organiza una API por recursos accesibles vía URL, manipulados con verbos HTTP: GET para leer, POST para crear, PUT y PATCH para modificar, DELETE para eliminar.

Cada llamada a una API REST tiene método, URL, headers (autenticación, content-type, accept), opcionalmente body, y devuelve un status code más un body de respuesta. Los rangos del status code importan: 2xx es éxito (200 OK, 201 Created, 204 No Content), 3xx es redirección, 4xx es error del cliente (400 mal formada, 401 sin autenticar, 403 sin permiso, 404 no existe, 429 te pasaste de rate limit), 5xx es error del servidor (500 reventó, 503 caída temporal). Los esquemas de autenticación habituales son API key en un header custom, Bearer token (formato HTTP estándar usado por OAuth 2, JWTs y tokens personales) y Basic Auth con usuario y password codificados en base64; el nodo HTTP Request de n8n cubre los tres con configuración declarativa. Con este modelo mental en la cabeza dominas el 90% del nodo HTTP Request y puedes leer la documentación de cualquier API REST sin tutoriales intermedios.

Si quieres entrar a fondo en este concepto, ver: Base Nº 01 — HTTP y APIs sin folklore (planeado).

0.4 Webhooks

Un webhook es una URL pública que tu sistema expone para que un servicio externo te envíe datos por POST cuando ocurre un evento.

Es el patrón "ellos te llaman", opuesto al polling, donde tú preguntas cada cierto tiempo si pasó algo nuevo. Stripe te avisa de un pago con un webhook, GitHub te avisa de un push, Mercado Pago te avisa de un cambio de estado de una transacción; el servicio externo hace POST a tu URL con el payload del evento y tu sistema reacciona. En n8n, el nodo Webhook crea esa URL pública en el momento que activas el workflow y la dispara cada vez que llega un POST con el path configurado. Para producción real necesitas dos cosas además del trigger: responder rápido con 2xx (la mayoría de servicios reintentan si no reciben respuesta en pocos segundos) y verificar la autenticidad del webhook con HMAC y un signing secret, porque la URL es pública y cualquiera puede mandarte un POST falso. Ambos temas se cubren en la Sección 5.

0.5 Docker

Docker es una tecnología de containerización que empaqueta una aplicación con sus dependencias en una unidad portable, la imagen, que se ejecuta como un container.

Para correr n8n self-hosted necesitas cuatro piezas mínimas: la imagen (`docker.n8n.io/n8nio/n8n`, el registry oficial), el container (la instancia corriendo de esa imagen), el volume (almacenamiento persistente fuera del container, donde sobreviven tus credenciales y workflows) y el port mapping (cómo el puerto interno del container queda accesible desde fuera). No hace falta que domines `docker compose` para leer la Sección 8, pero sí que reconozcas qué hace cada pieza cuando aparezca en un comando.

0.6 Expresiones de n8n

Una expresión es sintaxis JavaScript embebida entre `{{ }}` que n8n evalúa en tiempo de ejecución para calcular el valor de cualquier campo de cualquier nodo.

Te dan acceso a variables especiales del workflow: `$json` es el item actual sobre el que se ejecuta el nodo, `$("NombreDelNodo").item.json` es el output del nodo referenciado en este run, `$input.all()` son todos los items entrantes en este nodo, `$input.item` es el item actual procesándose, y existen además `$workflow`, `$runIndex`, `$execution` y `$now` para metadata y tiempo. Soportan JavaScript moderno: ternarios, optional chaining (`?.`), nullish coalescing (`??`), spread operator y todos los métodos de array (`.map`, `.filter`, `.reduce`, `.find`); también tienes acceso a librerías incluidas como Luxon para fechas. El editor de expresión muestra el resultado evaluado en vivo contra el primer item del input, lo que convierte la prueba de una expresión en una operación de segundos: úsalo en lugar de adivinar y desplegar. Para lógica de más de tres

líneas conviene saltar al nodo Code, donde tienes un editor de verdad y puedes escribir funciones; las expresiones se vuelven ilegibles después de cierto largo.

0.7 LLMs y prompts

Un LLM (Large Language Model) es un modelo entrenado para predecir el siguiente token a partir de un contexto de texto. Un prompt es la entrada de texto que le das.

Cinco conceptos mínimos. **Token**: la unidad mínima que procesa el modelo. En español, una palabra promedio consume entre 1 y 2 tokens (la regla de "tres cuartos de palabra" es de inglés; el español paga más por la morfología flexiva de plurales, conjugaciones y sufijos). Los precios de las APIs se cuentan por millón de tokens de input y output, separados. **Ventana de contexto**: cuántos tokens caben en una llamada (input + output), hoy entre 8K y más de 1M según el modelo; cuando te pasas, la API rechaza o trunca silenciosamente. **System prompt vs user prompt**: el system instruye comportamiento estable (rol, tono, formato esperado, restricciones), el user es la entrada variable que cambia en cada invocación; mantenerlos separados es la diferencia entre un prompt mantenible y un string monolítico. **Temperature**: un parámetro entre 0 y 2 que regula la aleatoriedad de la salida; cerca de 0 conviene para extracción y clasificación, más alto para escritura creativa. **Function calling o tool use**: el modelo puede decidir invocar funciones que le defines en lugar de solo responder texto plano, devolviendo un objeto JSON con el nombre de la función y sus argumentos; es la base sobre la que se construyen los AI Agents que aparecen en la Sección 7.

0.8 Deuda técnica

Deuda técnica es el costo futuro de mantenimiento y cambio que se acumula cuando tomas decisiones de implementación que funcionan ahora pero no escalan en complejidad, claridad o robustez.

Un workflow con cuarenta nodos y lógica entremezclada es deuda técnica desde el primer cambio. Como toda deuda, paga intereses: cada bug toma más tiempo en diagnosticarse, cada extensión rompe algo no relacionado, cada handover requiere reverse engineering. La estrategia de este manual es prevenirla con disciplina temprana (Sección 2 sobre arquitectura modular, Sección 5 sobre manejo de errores). Reconocerla cuando aparece importa más que evitarla siempre.

0.9 Nodo (en n8n)

Un nodo es la unidad ejecutable mínima de un workflow. Cada nodo tiene una función (trigger, transformación, integración con un servicio o control de flujo) y opera sobre el array de items que recibe del nodo anterior.

Es distinto al "nodo" de grafos o estructuras de datos en programación general. En n8n un nodo siempre tiene inputs y outputs visibles en el canvas, parámetros configurables en el panel lateral, y un schema de datos de salida inspeccionable. Punto clave: la mayoría de los nodos se ejecutan una vez por cada item que reciben, no una vez por ejecución del workflow. Algunos (notablemente el nodo Code en su modo por defecto) se ejecutan una sola vez sobre el array completo; la elección del modo es parte de la configuración del nodo. Este detalle se profundiza en la Sección 1 (mental model) y se vuelve crítico en la Sección 4 (trampas de datos).

1 Setup & mental model

Todo en n8n son arrays de items que fluyen entre nodos. Internalizar esa regla primero te ahorra el 80% de los errores de cualquier workflow.

Esta sección no enseña a "usar n8n" de cabo a rabo. Hace dos cosas: te deja con una instancia corriendo donde puedas seguir el resto del manual, y te instala el modelo mental correcto antes de que aprendas un solo nodo. El orden importa. La mayoría de los problemas que un principiante atribuye a "n8n es raro" no son rarezas del producto; son consecuencias predecibles de una intuición equivocada sobre cómo viajan los datos. Diez minutos en la subsección 1.3 valen más que diez horas de tutoriales sueltos.

1.1 Instalación: Cloud, Docker self-hosted, comparativa rápida

Tres formas reales de correr n8n en 2026: **n8n Cloud** (instancia administrada), **Docker self-hosted** (producción seria), y **npx n8n** (Node.js local, evaluación rápida sin configuración). Cloud es la versión gestionada: pagas una suscripción mensual, el equipo de n8n se ocupa del servidor, las actualizaciones y los backups. Docker self-hosted es un container que corres en tu propia infraestructura (un VPS, un servidor on-premise o tu Mac para desarrollo); pagas la infraestructura y haces el mantenimiento. `npx n8n` levanta una instancia local con Node.js sin instalar nada permanente, persistiendo los datos en `~/.n8n`; sirve para probar la herramienta en cinco minutos antes de decidir qué opción real adoptar, no para producción.

Dimensión	Cloud	Self-hosted (Docker)	npx local
Precio inicial	Desde €20/mes (plan Starter)	Desde €5/mes (VPS chico)	Gratis (solo tu máquina)
Control sobre datos	Limitado al panel y políticas n8n	Total: filesystem, base, red	Total: vive en `~/.n8n` local
Data residency	Región fija del plan	Donde sea que esté tu servidor	Tu máquina
Mantenimiento	n8n lo gestiona	Tú: updates, backups, monitoring	Inexistente: efímero
Soporte	Oficial del equipo n8n	Comunidad (foro, GitHub, Discord)	Comunidad

La recomendación práctica es simple. Si quieres probar n8n en cinco minutos sin instalar nada, corre `npx n8n` y abre `http://localhost:5678`; sirve para acompañar la lectura del manual sin compromiso. Para uso real, empieza con Cloud para evaluar la herramienta y para proyectos chicos donde la factura mensual no duela. Migra a self-hosted cuando se cumpla al menos una de tres condiciones: el plan Cloud supera los €50/mes, necesitas

control sobre dónde viven los datos por requisito regulatorio o de cliente, o quieres correr nodos community que Cloud no permite. No te vayas a self-hosted "porque suena más profesional"; te vas cuando hay un motivo concreto.

Para arrancar una instancia local con Docker (la opción recomendada para acompañar el manual si vas a iterar en serio), este comando alcanza:

```
docker run -it --rm \
  -p 5678:5678 \
  -v n8n_data:/home/node/.n8n \
  docker.n8n.io/n8nio/n8n
```

Con eso tienes n8n escuchando en `http://localhost:5678`, con los datos persistidos en el volume `n8n_data`. La imagen es la oficial publicada en el registry de n8n; otras imágenes que circulan por Docker Hub son builds antiguos o de terceros. Si prefieres no usar Docker, `npx n8n` levanta la misma UI con Node.js, persistiendo en `~/.n8n` en tu home.

> NOTA
>
> Para producción seria no se usa este `docker run` suelto. Se usa `docker compose` con Postgres como base de datos, un reverse proxy con HTTPS y `N8N_ENCRYPTION_KEY` configurada de forma persistente. El stack completo se cubre en §8.

1.2 Tour de la interfaz

La pantalla principal de n8n tiene cuatro zonas que vas a usar todo el tiempo. Aprender dónde queda cada una ahorra fricción real; muchos atascos del principio son simplemente no saber dónde buscar.

El **editor canvas** ocupa el centro. Es un lienzo donde arrastras nodos desde la sidebar y los conectas dibujando líneas de output a input. Cada conexión representa flujo de datos: el array de items del nodo A entra como input del nodo B. Las conexiones tienen dirección; el handle del lado derecho de un nodo es output, el del lado izquierdo es input. Doble click sobre un nodo abre su panel de configuración.

La **sidebar de nodos** aparece a la derecha cuando haces click en el botón + del canvas o sobre un nodo conectable. Tiene buscador por nombre y categorías (Triggers, Action Nodes, Core Nodes, AI). El buscador es la vía rápida: escribe "webhook", "http", "code" o el nombre del servicio que necesitas y aparece. Las categorías sirven cuando aún no sabes qué buscar.

El **panel de output** es la pieza más usada del workflow y la que más se subestima. Aparece a la derecha del panel de configuración del nodo seleccionado y muestra los datos que ese nodo produjo en la última ejecución. Tiene tres vistas conmutables: **Table** (formato tabular cuando los items comparten estructura), **JSON** (el array crudo, tal cual lo ven los nodos río abajo) y **Schema** (un árbol con los nombres de campos y tipos, útil para entender estructuras anidadas sin leer JSON entero). El contador "N items" en la cabecera te dice cuántos elementos produjo el nodo; si dice "0 items", la lógica que viene después no se va a ejecutar.

> NOTA
>
> Cuando trabajes con un workflow que tarda en correr (depende de APIs externas, espera webhooks reales), usa **pin data** sobre el nodo del trigger: fija un output de ejemplo y los nodos posteriores ejecutan contra

> ese pin sin volver a llamar al origen. El uso disciplinado de pin data acelera el desarrollo en órdenes de magnitud. Se profundiza en §5.5.

La pestaña **Executions** lista el historial de ejecuciones del workflow: cuándo corrió, si tuvo éxito o falló, cuántos items procesó cada nodo, cuánto tardó. Click sobre cualquier ejecución abre una vista que reproduce el estado del canvas en ese momento, con los datos reales que pasaron por cada conexión. Es la herramienta principal para debuggear qué pasó en producción.

La pestaña **Credentials** es donde guardas las llaves para conectarte a servicios externos (API keys, OAuth tokens, conexiones SQL). Las credenciales se definen una vez, se nombran y luego se referencian desde múltiples nodos sin volver a pegar valores. Se almacenan cifradas en la base de n8n con la `N8N_ENCRYPTION_KEY`; perder esa key vuelve ilegibles todas las credenciales guardadas. El manejo correcto en producción se cubre en §8.

1.3 El mental model: items y arrays como moneda

Esta es la subsección que decide si los próximos siete capítulos te van a costar el doble o la mitad. Léela despacio.

La moneda de cambio entre nodos en n8n es siempre la misma: **un array de items**. Nunca un objeto suelto, nunca un valor primitivo. Si un nodo produce un solo resultado, te lo entrega como un array de longitud uno. Si no produce nada, te entrega un array vacío. Un **item** es un objeto JSON con dos claves canónicas, `json` y opcionalmente `binary`. La carga útil del item vive bajo `json`; los archivos adjuntos (imágenes, PDFs, CSV crudos) viajan bajo `binary`.

Esta regla parece obvia escrita, pero la intuición de quien viene de programar funciones tradicionales es exactamente la inversa: "el nodo recibe los datos, los procesa y devuelve un resultado". En n8n el nodo recibe **muchos** datos, los procesa **uno por uno o todos juntos** (según su modo) y devuelve **muchos** resultados. Cada vez que escribas una expresión, pregúntate: ¿esto se evalúa una vez sobre un array, o se evalúa N veces sobre cada item? La respuesta a esa pregunta es lo que el modo del nodo decide por ti, y casi siempre con un default razonable, pero saber cuál es el default es lo que separa al usuario que entiende lo que está haciendo del que va por prueba y error.

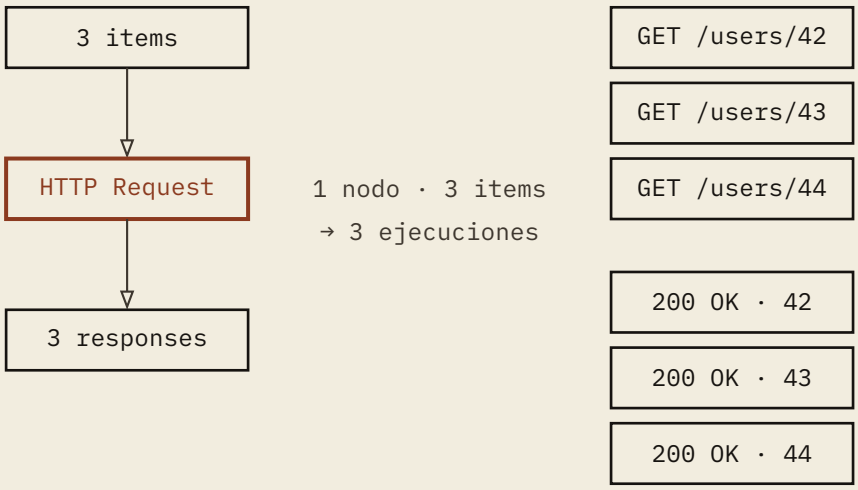
```
{
  "json": {
    "id": 42,
    "email": "ana@example.com",
    "status": "active"
  }
}
```

Un array de items se ve así:

```
[
  { "json": { "id": 42, "email": "ana@example.com" } },
  { "json": { "id": 43, "email": "luis@example.com" } },
  { "json": { "id": 44, "email": "sofia@example.com" } }
]
```

La consecuencia de esta regla es directa: **la mayoría de los nodos se ejecutan una vez por cada item que reciben**. Si tres items entran a un nodo HTTP Request configurado para llamar a `https://api.example.com/users/`

`{{$json.id}}`, n8n hace tres requests. Si entran cien, hace cien. Si entran cero, no hace ninguno y todo lo que venga río abajo recibe el array vacío.



Tres items entran al nodo HTTP Request, salen tres requests independientes, vuelven tres respuestas. Una llamada por item.

Ese comportamiento "una ejecución por item" es el implícito de HTTP Request, Set, IF y la mayoría de nodos: no tienen un selector de modo, simplemente iteran sobre el array entrante. No lo confundas con un nombre formal del producto.

En n8n hay **dos mecanismos distintos** que controlan cuántas veces corre un nodo. Confundirlos es la fuente del 80% de los "¿por qué se ejecutó N veces?".

Mecanismo 1 – Dropdown del Code node (exclusivo de ese nodo). El nodo Code expone en su panel un selector explícito con dos opciones:

Opción del dropdown	Comportamiento
Run Once for All Items *(default)*	El código corre **una sola vez** sobre el array completo. Dentro del script recibes `$input.all()` (array de items) y devuelves un array. Útil cuando la operación es naturalmente agregada: sumar, ordenar, filtrar, deduplicar.
Run Once for Each Item	El código corre **N veces**, una por cada item entrante. Dentro del script accedes al item actual con `$input.item` (singular) y devuelves un único item. Útil cuando la transformación es local a cada item y depende de su contenido.

Mecanismo 2 – Toggle "Execute Once" en Settings (presente en casi todos los nodos: HTTP Request, Set, IF, Merge, Postgres, etc.). Cuando lo activas en **Settings → Execute Once**, el nodo procesa **solo** `items[0]` y **descarta el resto**. Es decir: si entran cien items y el toggle está activado, el nodo corre una vez sobre el primero y los items 2..N se pierden. Sirve cuando quieres usar el primer item como configuración (por ejemplo, leer un único registro y descartar los demás) y no iterar.

> ⚠ ADVERTENCIA

> "Execute Once" del toggle **no es lo mismo** que "Run Once for All Items" del dropdown del Code. En el toggle se descartan los items 2..N; en el Code en modo "Run Once for All Items" se procesan todos juntos en un solo array. Confundirlos lleva a perder datos silenciosamente.

En todos los nodos que no son Code y que no tienen el toggle activado, el comportamiento es el implícito: una ejecución por item, iterando sobre el array entrante. No hay nombre formal para este modo en la doc oficial; es simplemente lo que hacen los nodos por defecto.

Tipo de nodo	Default	Variables que usas
HTTP Request, Set, IF, Postgres	Una ejecución por item	`$json`, `$input.item`
Code (dropdown propio)	Run Once for All Items	`$input.all()`, `$json` (item 0)
Aggregate, Sort, Limit, Merge	Operan sobre el array completo por naturaleza	`$input.all()`

> ⚠ ADVERTENCIA
>
> **El gotcha del array vacío**. Si un nodo en modo "Run Once for Each Item" recibe un array de cero items (porque el trigger no encontró nada, porque un filtro descartó todo, porque la API devolvió `[]`), no se ejecuta nunca y los nodos río abajo también reciben cero items. Las expresiones que asumen `$json.algo` no fallan visiblemente: simplemente no hay corrida donde fallar. El workflow termina "exitoso" sin haber hecho nada y el log no muestra error. La defensa correcta es comprobar el conteo de items en el panel de output del primer nodo donde sospechas vacío, y agregar un nodo IF que detecte el caso `$input.all().length === 0` para ramificar a una notificación o a un branch de "nada que hacer".

Hay un detalle más que conviene aprender ahora porque va a aparecer en capítulos posteriores: el **linaje de items**. Cuando un nodo en modo "Run Once for Each Item" produce output, n8n asocia cada item de salida con el item de entrada que lo generó, vía un campo interno llamado `pairedItem`. Eso permite que dos ramas paralelas del workflow (por ejemplo, una rama que valida y otra que enriquece) puedan converger después en un nodo Merge y reunir los datos del mismo item original, no de items distintos cruzados al azar. La mayoría de los nodos nativos mantienen el linaje sin intervención. En el nodo Code que tú escribes a mano, el linaje se preserva automáticamente sólo en el modo "Run Once for Each Item"; en "Run Once for All Items" tienes que devolverlo explícitamente con `{ json: {...}, pairedItem: { item: i } }` para que los Merge río abajo sigan funcionando. Es de las pocas mecánicas internas que vale la pena conocer desde el principio porque ahorra debugging futuro inexplicable.

Conviene también nombrar el **orden de ejecución**. Cuando un nodo recibe N items, los procesa secuencialmente en el orden en que llegaron, no en paralelo. Es decir: cien llamadas HTTP no se disparan al mismo tiempo; se disparan una tras otra, esperando la respuesta de cada una antes de mandar la siguiente. Eso tiene dos consecuencias prácticas: una integración con cien items que tarden un segundo cada una toma cien segundos, no uno; y nunca te encuentras con condiciones de carrera dentro del mismo nodo. Para paralelizar de verdad hay que usar nodos específicos con configuración de batching (`Split In Batches` o el modo de paralelismo del HTTP Request), tema de §3.

Resumen mental, en una línea: **cada nodo es una función que se aplica secuencialmente al array de items entrante; el modo del nodo decide si la función opera ítem por ítem o sobre el array completo, y el linaje se preserva para que las ramas paralelas converjan limpio**. Con esta frase memorizada, los próximos siete capítulos son consecuencia.

1.4 Tu primer workflow en 5 minutos

El workflow más corto que demuestra el modelo end-to-end tiene tres nodos: un trigger que recibe HTTP, una transformación que agrega un campo, una respuesta que devuelve JSON al llamante.

Paso 1 — Trigger. En un workflow nuevo, click en el `+` y busca "Webhook". El nodo se llama exactamente **Webhook**. En el panel, en el campo `HTTP Method` deja `POST`, en `Path` escribe `test`. En el campo **Respond**, selecciona **Using 'Respond to Webhook' node**. Sin esto, n8n devolverá un mensaje genérico de inicio de workflow (`{"message": "Workflow was started"}`) en lugar del JSON que vas a construir río abajo. n8n genera dos URLs visibles en el panel: una para **Test** (escucha hasta que ejecutas el nodo manualmente) y una para **Production** (escucha siempre que el workflow esté activo). Para esta prueba usas la URL de Test.

Paso 2 — Transformación. Conecta el output del Webhook a un nodo **Edit Fields** (también conocido como **Set**). En el panel, agrega un campo nuevo llamado `timestamp` con valor `={{ $now.toISO() }}`. Mantén el toggle "Include other input fields" activado para que los datos originales del webhook (name, email, lo que sea) pasen junto al timestamp agregado.

Paso 3 — Respuesta. Conecta el output del Set a un nodo **Respond to Webhook**. En el panel, deja `Respond With` en `First Incoming Item` y `Response Code` en `200`. Ese nodo es el que cierra el ciclo y devuelve el item al cliente HTTP.

Para probar el workflow, haz click en **Execute workflow** (queda escuchando el webhook), y desde una terminal lanza:

```
curl -X POST https://tu-instancia.app.n8n.cloud/webhook-test/test \
  -H "Content-Type: application/json" \
  -d '{"name": "Ana", "email": "test@test.cl"}'
```

La respuesta esperada tiene la forma:

```
{
  "name": "Ana",
  "email": "test@test.cl",
  "timestamp": "2026-05-26T18:42:13.521-04:00"
}
```

Cuatro notas sobre lo que acaba de pasar. Primero, el path del webhook en el comando es `/webhook-test/test`, no `/webhook/test`: la URL de Test va por un prefijo distinto a la de Production (`/webhook/test`). El panel del nodo te muestra ambas; copiar la equivocada es el error más frecuente. Segundo, `$now` es una variable global de n8n que devuelve un objeto Luxon DateTime con la fecha y hora actuales; `.toISO()` lo serializa a string ISO 8601. Tercero, el formato de la respuesta lo decide **Respond to Webhook**: si lo configuras como `JSON` en lugar de `First Incoming Item`, puedes devolver un body custom escrito como expresión, útil cuando la API contra la que respondes espera un envelope específico. Cuarto, después de la primera ejecución exitosa el canvas se "pinta": el panel de output de cada nodo queda con los datos reales que pasaron por ahí, y puedes inspeccionar el item tal como llegó al Set y tal como salió hacia Respond to Webhook. Esa inspección post-ejecución es el ciclo de feedback más rápido que vas a tener para iterar sobre lógica.

Con esos tres nodos tienes un endpoint HTTP funcionando que enriquece la entrada con un timestamp del servidor. Es trivial, pero contiene el patrón completo del 70% de las integraciones reales: alguien te llama, transformas, respondes. Lo que sigue del manual son variaciones más interesantes sobre la misma estructura: triggers más complejos, transformaciones con condicionales y múltiples ramas, respuestas con manejo de errores, integraciones con servicios externos donde "transformar" significa enriquecer con una llamada a una API, validar contra un esquema, o pasar por un LLM. Pero la forma es la misma: trigger, lógica, salida.

✕ LO QUE NO HACER

Tratar cada nodo como una función que corre una vez por ejecución del workflow. En n8n, un nodo se ejecuta una vez **por cada item** que recibe, con la excepción notable del nodo Code que por defecto corre una sola vez sobre el array completo. Si cien items entran a un HTTP Request, son cien requests salientes. Si cien items entran a un nodo Code en modo "Run Once for Each Item", el código corre cien veces. Internalizar esto antes de armar tu primer workflow real evita el 80% de los "¿por qué se ejecutó cincuenta veces?" que todos cometen al inicio, y te ahorra facturas inesperadas cuando el "cada item" se traduce en llamadas a APIs pagadas o tokens de LLM.

Continúa

Hasta aquí llega la muestra. El manual completo cubre el 80% restante del recorrido, en seis secciones temáticas y una de cierre:

- **Sección 2 — Arquitectura modular.** Cómo evitar el workflow de cuarenta nodos sin perder velocidad de iteración.
- **Sección 3 — Nodos esenciales.** El 20% de nodos que aparece en el 80% de los workflows reales, con sus parámetros no obvios.
- **Sección 4 — Datos y transformaciones.** Trampas frecuentes con arrays, objetos anidados, paginación y normalización.
- **Sección 5 — Manejo de errores.** Workflows resilientes, reintentos estructurados, alertas accionables y verificación HMAC de webhooks.
- **Sección 6 — LLMs.** El mismo ejemplo de extracción estructurada implementado lado a lado en Claude, GPT y Gemini, con sus diferencias de sintaxis.
- **Sección 7 — Agents.** Tool use, memoria, control de bucles y el patrón de research agent del anexo de workflows ejecutables.
- **Sección 8 — Deployment.** Docker Compose productivo, Caddy como reverse proxy con TLS automático y la rutina de backup.

Junto al PDF van los 5 workflows ejecutables en `.json`, los 5 system prompts curados en `.md` con variantes para Claude, GPT y Gemini, y los archivos de infraestructura del anexo. Todo entregable se descarga en el mismo email, inmediatamente después del pago, sin formularios ni inicio de sesión.

[paretoseries.com]

Manual completo · $29.990 CLP · 84 páginas