LO ESENCIAL · **Módulo 9 · Automatización y orquestación**

Flujos deterministas donde se pueda, modelo solo donde haya ambigüedad. Detectar la ruptura silenciosa.

9.1 · Herramientas de flujo: fortalezas y límites reales

Objetivos

- Decidir entre flujo visual y código
- Montar la ingesta nocturna

Flujo visual o código

Las herramientas de flujo (n8n, Node-RED y similares `[VOLÁTIL — verificado: 2026-09]`) permiten armar automatizaciones arrastrando cajas: «cuando llegue un PDF a esta carpeta → extraer campos → guardar → avisar». Son útiles y tienen límites que conviene decir antes de que el cliente se enamore de la interfaz.

	Herramienta de flujo	**Código (Python)**
Quién lo entiende	el cliente ve las cajas	solo quien programa
Velocidad de armado	alta para lo típico	media
Versionado	exportación JSON grande; el diff es ilegible	Git línea a línea
Pruebas	manuales, en la interfaz	automáticas
Errores	silenciosos si no se configuran avisos	excepciones que se registran
Lógica compleja	se vuelve un plato de espagueti	funciones
Mantenimiento a 3 años	depende de la herramienta y su versión	depende de Python

Criterio: flujo visual para **pegar** sistemas (carpeta → cola → correo) y para que el cliente vea el proceso; **código** para todo lo que tenga lógica, esquemas o pruebas. Y la frontera entre los dos es un contrato JSON (módulo 10.4).

PROCEDIMIENTO · **Laboratorio 9.1 · la ingesta nocturna**

Un flujo que a las 2 a. m. toma los PDF nuevos de la carpeta del NAS, llama al extractor (7.3) por HTTP, guarda el JSON, mueve el PDF a `procesados/` o `revision/`, y a las 6 a. m. manda un resumen al gerente: cuántos, cuántos a revisión, cuáles fallaron.
Se construye en la herramienta de flujo elegida **y** se exporta a `labs/09-01/ingesta.json`,

> versionado. El extractor sigue siendo código: el flujo solo lo llama.
> **Entregable:** el flujo exportado, y la captura del resumen de las 6 a. m. Commit: «Capa 9.1: ingesta nocturna».

Qué se degrada en cada perfil

A — El lote nocturno es **la forma natural** de usar el perfil A: el prefill lento no molesta de madrugada

B — Igual, y además en vivo

Cuándo NO usar esto **Herramientas de flujo: fortalezas y límites reales**

- No meta la lógica del extractor dentro de cajas del flujo: no se prueba ni se versiona.
- No use un flujo para algo que corre cada minuto con carga: el motor de flujos añade latencia y memoria.
- No confíe en que «si falla, se ve en la interfaz»: se ve si alguien mira. Lección 9.3.

PRÁCTICA · **Práctica**

1. Dibuje el flujo con 6 cajas. Marque cuáles son deterministas y cuáles llaman al modelo.
2. ¿Qué pasa si el extractor tarda 30 minutos una noche? Diseñe el tope.

Referencias

1. n8n — https://docs.n8n.io/ · Node-RED — https://nodered.org/docs/ `[VOLÁTIL — verificado: 2026-09]`
2. Módulo 11.3 — cómo se versiona la exportación del flujo.

9.2 · La regla determinista / probabilístico

> Objetivos
>
> - Refactorizar un flujo dejando el modelo solo donde hace falta

> - Explicar el costo de cada paso probabilístico

La regla

> Nota **El modelo solo donde hay ambigüedad real. Todo lo demás, código determinista.**

Un modelo de lenguaje es caro, lento y **no repetible**: la misma entrada puede dar salidas distintas. Una función de Python es barata, rápida y da siempre lo mismo. Cada paso de un flujo se clasifica antes de construirlo:

Paso	¿Ambiguo?
Detectar que llegó un PDF	no
Saber si tiene capa de texto	no
Decidir el nivel de OCR	poco
Extraer campos de un escaneo	**sí**
Verificar `subtotal + iva = total`	no
Verificar el dígito del NIT	no
Decidir si «Tornillería Santander S.A.S.» y «TORNILLERIA SANTANDER» son el mismo proveedor	**sí**
Mover el archivo a `procesados/`	no
Redactar el resumen para el gerente	**sí** (es redacción)
Enviar el correo	no

De diez pasos, **tres** son del modelo. Cada uno de los otros siete, si se le diera al modelo, sería más lento, más caro y a veces equivocado.

Ejemplos de los dos lados

Determinista que la gente pone en el modelo por costumbre: formatear fechas, sumar, validar correos, ordenar, deduplicar exactos, convertir unidades, enrutar por extensión de archivo.

Probabilístico que la gente intenta hacer con reglas y fracasa: normalizar nombres de proveedores con variantes, clasificar el asunto de un correo, extraer campos de un layout nuevo, resumir, decidir si dos descripciones de producto son la misma.

> PROCEDIMIENTO · **Laboratorio 9.2 · refactorizar**
>
> Se entrega un flujo de 6 pasos donde **todos** llaman al modelo. Reescríbalo dejando el modelo solo donde hace falta. Mida antes y después: tiempo total, tokens gastados, y **repetibilidad** (corra tres veces: ¿mismo resultado?).
> **Entregable:** el flujo refactorizado y la tabla antes/después. Commit: «Capa 9.2: determinista donde se puede».

> Nada: la regla reduce el uso del modelo, y por tanto mejora más en A que en C.

Cuándo NO usar esto **La regla determinista / probabilístico**

- No convierta en reglas algo que tiene 40 casos especiales: llegó el momento del modelo.
- No use el modelo «para que sea flexible» en un paso que un auditor tiene que poder reproducir.

PRÁCTICA · **Práctica**

1. Clasifique los 12 pasos del flujo de ingesta de la ferretería.
2. Un cliente pide «que la IA decida si pagar la factura». Escriba en cinco líneas por qué no, y qué sí.

Referencias

1. Anthropic (2024). *Building effective agents* — «usar el patrón más simple que funcione».
2. Módulo 12 — cómo se mide la repetibilidad.

9.3 · Scraping responsable, tareas programadas y ruptura silenciosa

> Objetivos
>
> - Programar tareas con canario y alerta
> - Detectar que algo dejó de funcionar sin fallar

Lo que deja de funcionar sin fallar

Un proveedor cambia el formato de su factura. La página de precios cambia una clase CSS. El modelo se actualizó solo. Ninguno de esos casos lanza un error: el sistema **sigue corriendo y produce basura** con toda confianza. Se llama **ruptura silenciosa** y es el modo de fallo número uno de las automatizaciones en producción.

Scraping responsable

Si el asistente consulta listas de precios de proveedores en la web:

- Respetar `robots.txt` y los términos; identificarse con un `User-Agent` con correo de contacto.
- Frecuencia baja (una vez al día), en horario de poca carga, con caché.
- **Nunca** contra sitios con sesión o datos personales.
- Guardar la copia cruda con fecha: si el sitio cambia, hay evidencia de qué cambió.

Tareas programadas

`cron` (o el planificador del sistema) es determinista y aburrido: perfecto. Cada tarea tiene tope de tiempo, registro, y **canario**.

```
# /etc/cron.d/el-tornillo
0 2 * * *   app  timeout 3h /opt/tornillo/ingesta.sh   >> /var/log/tornillo/ingesta.log 2>&1
0 6 * * *   app  /opt/tornillo/resumen.sh              >> /var/log/tornillo/resumen.log 2>&1
30 3 * * *  app  /opt/tornillo/canario.sh              >> /var/log/tornillo/canario.log 2>&1
```

El canario

Un canario es una prueba pequeña con **respuesta conocida** que corre sola y avisa si cambia:

```
#!/usr/bin/env bash
# canario.sh — tres pruebas de respuesta conocida; si alguna cambia, alerta
set -u; fallos=0
# 1) el extractor sigue leyendo la factura patrón igual que ayer
python3 /opt/tornillo/extraer_factura.py /opt/tornillo/canario/patron.pdf | diff -q - /opt/
    tornillo/canario/patron.esperado.json >/dev/null || { echo "CANARIO extractor: cambió";
    fallos=$((fallos+1)); }
# 2) el RAG recupera la ficha conocida para la consulta conocida
python3 /opt/tornillo/consultar.py --usuario canario "¿cuál es la garantía del contrato 2026-014?
    " | grep -q "contrato-2026-014#7" || { echo "CANARIO rag: no recupera"; fallos=$((fallos+1));
     }
# 3) el modelo sigue siendo el mismo (hash del archivo) y responde en tiempo
[ "$(sha256sum /opt/modelos/actual.gguf | cut -c1-16)" = "$(cat /opt/tornillo/canario/modelo.hash
    )" ] || { echo "CANARIO modelo: cambió el archivo"; fallos=$((fallos+1)); }
[ $fallos -gt 0 ] && /opt/tornillo/alertar.sh "canario: $fallos fallos" && exit 1
echo "canario OK $(date -Is)"
```

La alerta va por el canal que el cliente **sí** mira (correo, mensaje). Un canario que escribe en un log que nadie lee es un pájaro muerto.

> PROCEDIMIENTO · **Laboratorio 9.3**
>
> Monte las tres tareas programadas, el canario, y **provoque una ruptura** (cambie el formato de la factura patrón). Lo que debe ver: la alerta al día siguiente a las 3:30. Anote cuánto tardó en enterarse **sin** el canario (probablemente: nunca).
> **Entregable:** `cron.d`, `canario.sh`, `alertar.sh`, y la captura de la alerta. Commit: «Capa 9: automatización con canario».
> Nada. El canario en A tarda más pero corre de madrugada.

Cuándo NO usar esto **Scraping responsable, tareas programadas y ruptura silenciosa**

- No haga scraping de un sitio que ofrece API o descarga de datos: use eso.
- No programe tareas sin `timeout`: una tarea colgada bloquea la siguiente noche.

PRÁCTICA · **Práctica**

1. Escriba un cuarto canario para el proxy y el SSO.
2. El proveedor cambió el formato y el canario avisó. ¿Cuáles son los siguientes tres pasos?

Referencias

1. `cron` — `man 5 crontab`; `systemd.timer` como alternativa.
2. robots.txt — RFC 9309 (2022): https://www.rfc-editor.org/rfc/rfc9309
3. Módulo 12.4 — canarios de evaluación, la versión completa de este.