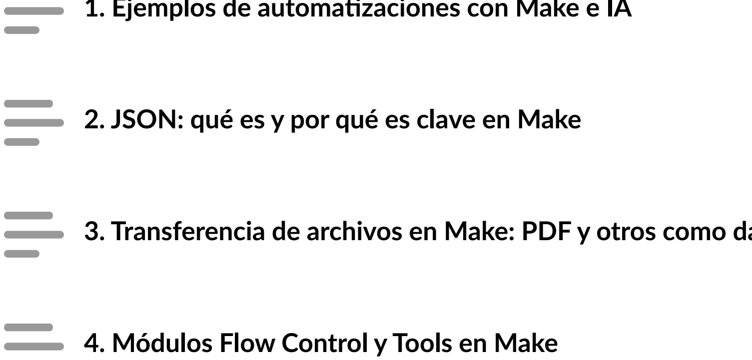
Módulo 3. Flujos avanzados con PDF, JSON y clasificadores



1. Ejemplos de automatizaciones con Make e IA

2. JSON: qué es y por qué es clave en Make

3. Transferencia de archivos en Make: PDF y otros como datos

4. Módulos Flow Control y Tools en Make

Referencias

Lección 1 de 5

1. Ejemplos de automatizaciones con Make e IA

Para entender el potencial de Make + IA, veremos tres escenarios donde Make se utiliza en automatizaciones complejas con ayuda de GPT u otras IA. Estos ejemplos se basan en casos reales documentados en comunidades técnicas, foros y blogs especializados.

Ejemplo 1: Generación masiva de contenido con ChatGPT y hojas de cálculo

Un uso avanzado de Make con IA es la creación automatizada de contenido a gran escala. Imaginemos que una empresa necesita generar descripciones de producto, ideas de palabras clave SEO o definiciones de un glosario empresarial para decenas de ítems. En lugar de pedir manualmente a ChatGPT cada respuesta, podemos automatizar el proceso con Make:

- Se preparan todas las solicitudes de contenido en una fuente de datos estructurada, por ejemplo, filas en Google Sheets (o Airtable). Cada fila contiene un tema o prompt.
- Make monitorea la hoja (módulo Google Sheets > Watch Rows) y cuando detecta nuevas filas, envía cada petición al módulo de OpenAI para obtener una respuesta.
- GPT genera la salida para cada solicitud, y Make toma esas respuestas y las coloca nuevamente en la hoja u otro sistema.

Figura 1. Generación masiva de contenido con ChatGPT y hojas de cálculo

Fuente: Make Community, 2023, https://bit.ly/4amXZat

Este método aprovecha la capacidad generativa de la IA para producir contenido nuevo de forma masiva a partir de un conjunto de solicitudes. En un caso práctico, Make permitió generar ideas de palabras clave SEO para títulos de artículos: se introdujeron los títulos en una hoja de Google, Make envió cada título a GPT, y las palabras clave sugeridas por la IA se escribieron de vuelta en la hoja, todo sin intervención manual (Make Community, 2023). Así, en pocos segundos la hoja quedó poblada con listas de *keywords* por cada artículo, demostrando una automatización potente y escalable.

Ejemplo 2: Clasificación automática de tickets de soporte con IA

Otro escenario avanzado es usar IA para clasificar y enrutar tickets de soporte al cliente de manera más inteligente. Las plataformas de atención (como Zendesk, Freshdesk, etc.) generan numerosos tickets que suelen etiquetarse o asignarse según palabras clave. Con Make y GPT, podemos llevar esto al siguiente nivel:

- Un módulo Zendesk > Watch Tickets detecta nuevos tickets de soporte ingresando.
- Make envía el contenido del ticket a un modelo de IA mediante el módulo OpenAI > Create a Completion o Create a Chat Completion, con un prompt diseñado para clasificar el caso según su intención.
- La IA aplica procesamiento de lenguaje natural para analizar el texto completo del ticket y devuelve una categoría o etiqueta sugerida (por ejemplo: “Reembolso”, “Consulta técnica”, “Sugerencia de funcionalidades”, etc.), basándose en el contexto y no solo en palabras aisladas (Tan, 2023).
- En función de la categoría devuelta, el escenario de Make utiliza rutas condicionales

para buscar al agente o equipo más adecuado. Esto podría implicar un módulo de búsqueda (ej. buscar en una tabla de agentes quién maneja “Reembolsos”).

- Finalmente, el escenario asigna el ticket al agente correspondiente en la plataforma de soporte.

Figura 2: Clasificación automática de tickets de soporte con IA

Fuente: elaboración propia.

Este flujo implementa una clasificación inteligente: a diferencia de reglas estáticas por palabras clave (que pueden fallar por falta de contexto), la IA comprende la intención del cliente. Por ejemplo, un ticket que menciona “cancelé mi suscripción y quiero mi dinero de vuelta” será categorizado

por GPT en “Reembolsos” aunque no contenga exactamente la palabra “reembolso”.

Ejemplo 3: Resumen automatizado de documentos PDF con IA

El tercer ejemplo combina la manipulación de archivos PDF y la IA, mostrando un flujo complejo pero muy útil: resumir automáticamente un documento PDF escaneado usando Make, un servicio de PDF y GPT. Supongamos que una empresa recibe reportes o contratos en PDF y quiere obtener un resumen rápido de su contenido:

- El escenario inicia cuando un PDF llega a una carpeta (por ejemplo, en Google Drive o Dropbox). Un módulo Google Drive > Watch Files podría detectar el nuevo archivo y luego Google Drive > Download a file obtener su contenido.

- Los PDFs escaneados primero deben convertirse a texto. Para ello, se integra un servicio como PDF.co dentro de Make: usando un módulo de PDF.co (por ejemplo, PDF.co > Make Searchable PDF u otro endpoint), se envía el archivo PDF descargado para que PDF.co le aplique OCR y lo convierta en un PDF con texto seleccionable o extraiga

directamente el texto (PDF.co., 2024). Básicamente, el PDF pasa a formato textual.

- Una vez obtenido el texto del documento, Make lo alimenta a la IA. Usamos el módulo OpenAI > Create a Chat Completion, proporcionándole como prompt el texto (o los fragmentos relevantes) precedido de una instrucción como “Resumir el siguiente documento en 3 párrafos”. GPT procesará el texto extenso y generará un resumen en lenguaje natural.

- El resultado devuelto por la IA, que consiste en uno o varios párrafos sintetizando lo esencial del PDF, puede entonces enviarse por email, guardarse en un archivo de texto o almacenarse en alguna base de datos según las necesidades.

Figura 3. Resumen automatizado de documentos PDF con IA

Fuente: elaboración propia.

Al finalizar, la salida que produce GPT es un párrafo con los puntos clave, dando al usuario una rápida comprensión del documento sin tener que leerlo completo. Este flujo avanzado demuestra cómo Make puede encadenar múltiples servicios: primero manejar un archivo binario, transformarlo a datos utilizables (texto) y finalmente aplicar IA para obtener información de alto nivel.

CONTINUAR

2. JSON: qué es y por qué es clave en Make

En el contexto de automatizaciones y API, JSON es uno de los formatos de datos más importantes. A continuación veremos qué es JSON y por qué Make lo utiliza ampliamente, en especial para interactuar con respuestas de IA.

¿Qué es JSON?

JSON (*JavaScript Object Notation*) es un formato de texto plano para representar datos estructurados, muy utilizado en intercambios de información entre aplicaciones web y servicios. Se basa en la sintaxis de objetos de JavaScript, pero es independiente del lenguaje y legible tanto por humanos como por máquinas. En JSON, los datos se organizan en pares clave-valor (por ejemplo: "nombre": "Juan") y estructuras como ***arrays*** (listas ordenadas de valores, por ejemplo: "productos": ["lapiz", "cuaderno", "borrador"]). Es un estándar abierto y ligero; gracias a su sencillez, se ha convertido en el formato predilecto para API y almacenamiento simple de datos. En resumen, un archivo o cadena JSON luce similar a un diccionario: las llaves encierran objetos, cada campo tiene un

nombre y un valor (que puede ser número, texto, booleano u otra estructura), y múltiples elementos pueden anidarse.
Un ejemplo breve de JSON podría ser:

```
{
 "usuario": "alice",
 "edad": 30,
 "intereses":
                    ["música",
                     "deportes",
                     "IA"]
}
```

Este formato representa claramente información etiquetada, fácil de generar y de *parsear* (analizar) por programas.

JSON en los flujos de Make

En Make, JSON es fundamental porque muchos módulos envían o reciben datos en este formato. Cada vez que Make se comunica con un servicio web mediante API (lo cual es frecuente con las apps integradas), está manejando JSON bajo el capó. Por ejemplo, las respuestas de la API de OpenAI llegan en JSON, y Make las interpreta para que podamos mapear sus campos en los siguientes pasos. También al mandar datos a servicios vía módulos HTTP o similares, a menudo debemos formatearlos como JSON.

¿Por qué es clave manejar JSON correctamente en Make, especialmente con módulos de IA? Porque JSON permite estructurar las salidas de la IA de forma predecible y fácil de procesar. Imaginemos que pedimos a GPT que devuelva varias cosas a la vez (por ejemplo, a partir de un texto de entrada que nos dé un resumen, una lista de palabras clave y un sentimiento). Si no imponemos formato, la respuesta será texto libre, difícil de separar

en partes. En cambio, podemos incluir en el prompt de OpenAI algo como: “Devuelve la respuesta formateada en JSON con las claves resumen, keywords y sentimiento correspondientes”. De esta manera forzamos a la IA a generar una estructura JSON. Luego, en Make, simplemente añadimos un módulo Tools > Parse JSON para convertir esa respuesta textual JSON en campos separados que se pueden mapear fácilmente (crm_automator, 2024). Cada parte de la respuesta de GPT quedará en un elemento accesible (por ejemplo, resumen mapeable a un campo de Google Sheets, *keywords* como una lista para otro uso, etc.), sin necesidad de hacer complejas manipulaciones de texto.

En resumen, JSON se usa en Make para estructurar datos complejos y facilitar su manipulación en el flujo. Con las integraciones de IA, aprovechar JSON es una buena práctica para obtener salidas controladas (listas, objetos, etiquetas) que luego podemos mapear a donde necesitemos. De lo contrario, lidiaríamos con texto libre difícil de partir. Por eso es recomendable, instruir a GPT a responder en JSON cuando requerimos múltiples valores con distinto destino, y luego utilizar el módulo correspondiente (Parse JSON) para extraer esos valores de forma segura (crm_automator, 2024). Esta combinación garantiza que las automatizaciones con IA sean más robustas y profesionales.

CONTINUAR

3. Transferencia de archivos en Make: PDF y otros como datos

En flujos de automatización es común tener que mover archivos de un punto a otro (por ejemplo, recibir un PDF por email, enviarlo a un módulo de IA y luego guardar resultados). Make puede manejar archivos, pero es importante aclarar que no "pasa archivos" literalmente de un módulo a otro como si fueran adjuntos, sino que opera con datos codificados que representan esos archivos. Veremos cómo funciona esto y cómo aprovecharlo, incluyendo un ejemplo práctico con PDF.co y OpenAI.

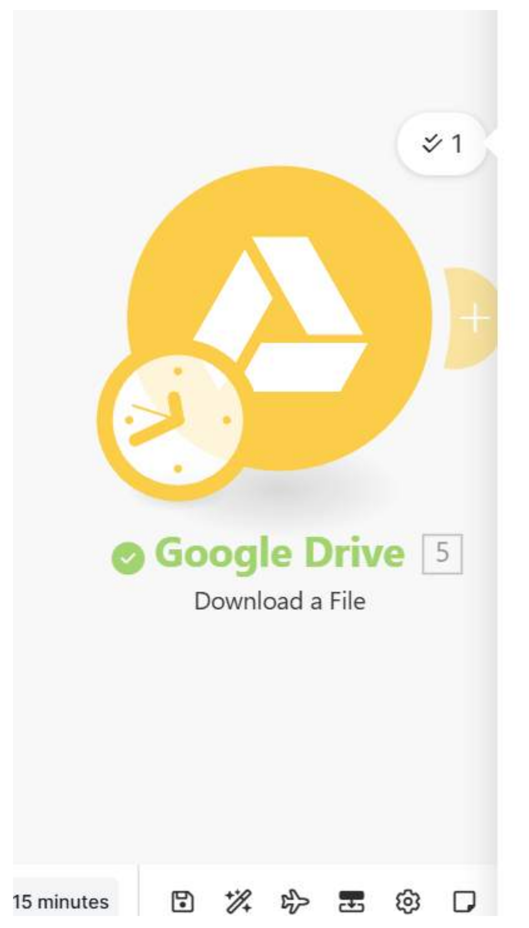
Archivos como datos codificados (no objetos físicos)

Cuando un módulo de Make obtiene un archivo (digamos un PDF, imagen, documento), internamente lo representa como datos binarios dentro de un *bundle*. Ese flujo de datos puede luego ser enviado a otro módulo. Pero no imagines que va

“arrastrando” un archivo .pdf por los cables; en realidad, lo que viaja es una serie de bytes codificados (generalmente en Base64 o similar) junto con metadatos. Base64 es un método de codificar datos binarios como texto (usando caracteres alfanuméricos), lo cual es útil porque muchos sistemas transmiten información en formato textual (JSON, XML, etc.). Make a veces convierte automáticamente el archivo en una URL interna o en un *string* base64 para pasarlo a la siguiente etapa.

Por ejemplo, si utilizamos un módulo Google Drive > Download a file para bajar un PDF, la salida contiene campos como el nombre, tipo MIME y los datos binarios. Esos datos pueden insertarse en otro módulo.

Figura 4. Archivos como datos codificados



Viewed By Me Time: 7 de octubre de 2025 12:00
Created Time: 3 de octubre de 2025 10:22
Modified Time: 30 de diciembre de 2025 9:51
Modified By Me Time: 7 de octubre de 2025 11:58
Modified By Me: true
Owners: (Array)
Shared: true
Owned By Me: true
downloadRestrictions: (Collection)
Capabilities: (Collection)
Viewers Can Copy Content: true
Copy Requires Writer Permission: false
Writers Can Share: true
Permissions: (Array)
Permission IDs: (Array)
Size: 52335
Quota Bytes Used: 52335
App Authorized: false
Export Links: (Collection)
Link Share Metadata: (Collection)
inheritedPermissionsDisabled: false
Data: (Buffer, codepage: binary)
50 4b 03 04 14 00 08 08 08 00 f3 34 28 5c 00 00
00 00 00 00 00 00 00 00 00 00 10 00 00 00 77 6f
72 64 2f 68 65 61 64 65 72 31 2e 78 6d 6c cd 57
db 6e db 38 10 fd 82 fd 07 41 ef 89 a4 c4 f5 da
42 ec a2 8d 9b 36 40 b1 35 9c 6c df 69 8a b2 88
f0 56 92 f2 a5 5f bf 43 51 92 2d 39 30 1c 67 81

Fuente: elaboración propia.

En la práctica, muchos módulos de Make ya hacen este manejo transparente. Por ejemplo, el módulo Email > Send an email permite adjuntar un archivo simplemente mapeando la salida de un Download file en el campo de adjunto, y Make se encarga de entregar el contenido. Similarmente, módulos de servicios de PDF o de almacenamiento *cloud* detectan ese contenido y lo utilizan. Pero siempre es bueno tener presente que lo

que circula es una representación del archivo y no el archivo en sí.

Entonces, a la hora de compartir un PDF, por ejemplo, con un módulo de OpenAI, no podemos enviarle a la IA como *prompt* los datos encriptados, ya que no podría interpretarlos. Por lo que ahí es donde resultan útiles los módulos de aplicaciones como PDF.co, I Love PDF, etc., que permiten, como vimos en el ejemplo anteriormente, extraer el contenido de los archivos como texto mapeable en Make y de este modo poder usarlo como *prompt*.

CONTINUAR

4. Módulos Flow Control y Tools en Make

Una de las razones por las que Make es tan poderoso es la existencia de módulos especiales que no se conectan a apps externas, sino que sirven para controlar la lógica interna del escenario o para manipular datos. Estos módulos se agrupan principalmente en dos categorías: *Flow Control* (control de flujo) y *Tools* (herramientas). A continuación, definiremos cada grupo y revisaremos detalladamente algunos de sus módulos más importantes, con ejemplos de uso.

Módulos de *Flow Control*

Los módulos de *Flow Control* permiten ramificar, iterar o reconducir la secuencia de operaciones en un escenario Make, más allá del simple flujo lineal de módulo 1 -> módulo 2 -> módulo 3. En esencia, determinan cómo se ejecutan las operaciones, cómo se reparte o combina la información y qué caminos sigue el escenario en función de condiciones (Aslan, s. f.). Los más utilizados son:

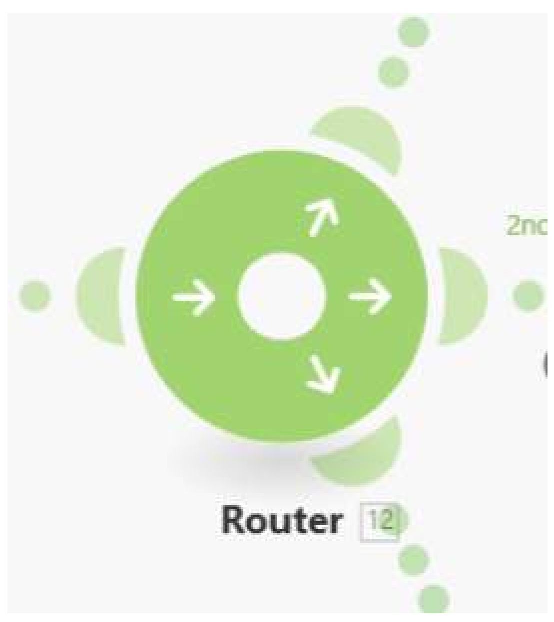
- ***Router* (enrutador):** un *router* sirve para dividir el flujo en rutas múltiples. Piensa en una bifurcación: a partir de un conjunto de datos de entrada, el *router* los puede enviar a diferentes ramas paralelas del escenario. Cada ruta puede tener sus propios módulos y filtros condicionales. Hay distintos modos de uso:

    - Puede usarse un *router* sin condición para simplemente duplicar la ejecución en dos o más caminos (todas las ramas se ejecutan con los mismos datos de entrada).

    - Más comúnmente, se añade un filtro a cada ruta para que solo ciertos datos vayan por esa vía. Por ejemplo, en un escenario de procesamiento de pedidos, podríamos tener un *router* con dos rutas: una con filtro “monto > 1000” (pedidos grandes) y otra con filtro contrario (pedidos pequeños), de modo que los

pedidos costosos desencadenen acciones adicionales en su rama (como notificar al gerente) mientras los demás siguen su curso normal.

Figura 5. Router



Fuente: elaboración propia.

- ***Iterator* (iterador):** un iterador hace lo opuesto a un agregador (que veremos enseguida): toma una colección (*array*) de elementos y la separa en unidades individuales (Aslan, s. f.). En Make, a veces un módulo devuelve varios ítems juntos en un solo *bundle,* por ejemplo,

una búsqueda que trae 50 registros, o un email con múltiples adjuntos (que vienen como un array de archivos). Si queremos procesar cada elemento uno por uno (por ejemplo, cada registro por separado), insertamos un *Iterator*. Este módulo recibe el array y emite múltiples *bundles*, cada uno conteniendo un ítem del array original. Así, convierte un *bundle* con un array de 50 elementos en 50 *bundles* secuenciales.

¿Cuándo usar un iterador? Cuando “cada elemento necesita su propio tratamiento”. Casos típicos:

Procesar cada línea de una orden de compra individualmente.

- Tomar cada fila de una hoja de cálculo obtenida y ejecutar una acción con sus datos (como vimos en el ejemplo 1, donde, tras obtener varias filas de Google Sheets, un *iterator* permitiría mandarlas de a una a GPT).
- Enviar correos personalizados a una lista de destinatarios: si tienes un array con varias direcciones, un iterador lanzará al

módulo de email las direcciones una por una, para que se envíen de esta forma.

El iterador se configura indicando qué campo del *bundle* entrante es el *array* que se desea iterar. Tras ejecutarse, los módulos subsecuentes “ven” los campos del elemento como campos normales.

Figura 6. Iterator

Fuente: elaboración propia.

- ***Aggregator* (agregador):** contrario al iterador, un agregador sirve para combinar múltiples *bundles* en uno solo. Es decir, recoge piezas de datos dispersas en varios “paquetes” y los une en una estructura de array o conjunto unificado (Aslan, s.f.). En Make existen distintos tipos de agregadores (numérico, de texto, de tabla), pero aquí nos centramos en el agregado de colecciones (*array aggregator*). Este módulo suele usarse después de rutas paralelas o iteradores, para reconsolidar la información.

¿Cuándo es útil un agregador? Cuando queremos reunir resultados separados antes de la siguiente acción. Ejemplos:

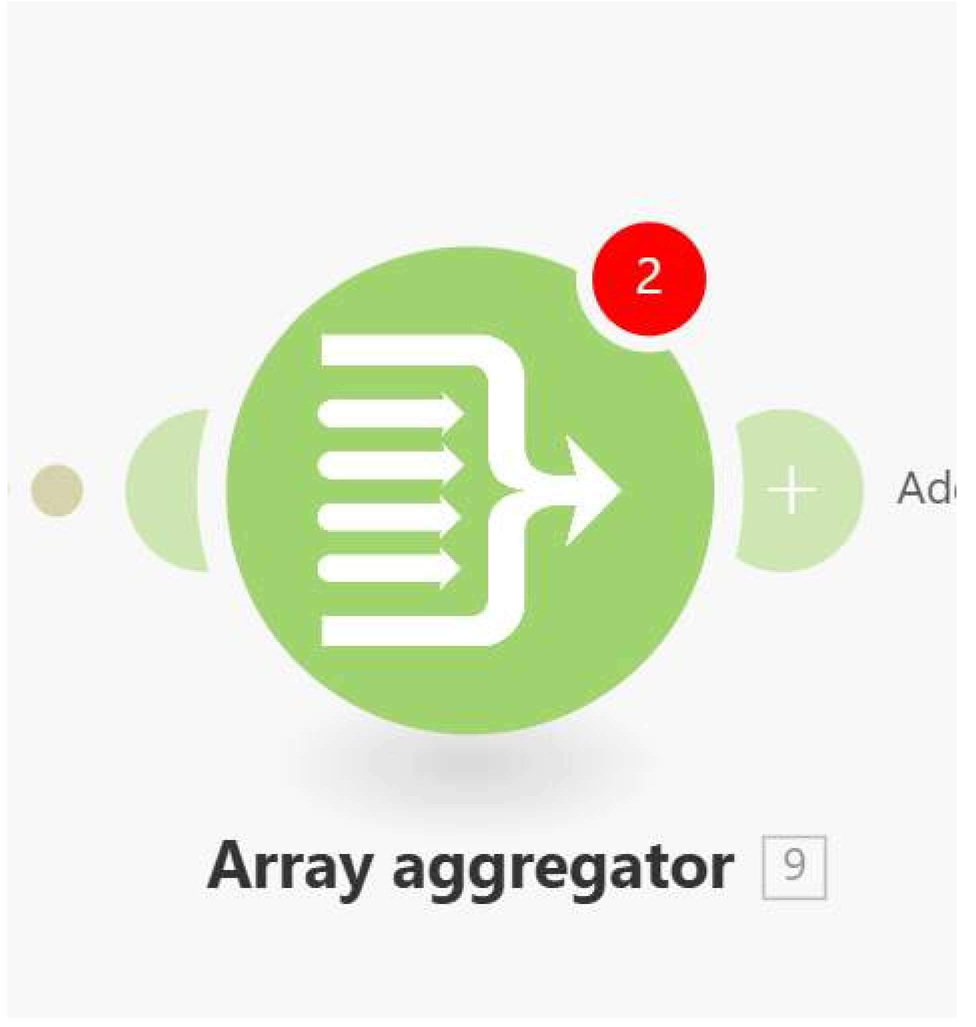
- Una consulta API que, por limitaciones, devuelve resultados paginados en varios *bundles*: con un agregador podríamos juntar todos los resultados en un solo array.
- Armar un resumen o reporte: imaginemos que en distintas ramas calculamos métricas (ventas, visitas, gastos) y queremos luego enviarlas juntas en un único mensaje. Cada rama produce un

bundle con su métrica; un *Aggregator* puede reunirlas en un solo *bundle* (por ejemplo, con campos “ventas”, “visitas”, “gastos”) para luego pasarlo a un módulo de Email y enviar un informe consolidado.

- Crear una lista única: por ejemplo, después de iterar sobre órdenes y recolectar los ID de productos comprados en cada una, podríamos agregarlos en una lista total de productos vendidos en el día.

En general, el *array aggregator* produce un array que contiene ciertos campos de todos los *bundles* de entrada. Se configura indicando cuál es el módulo fuente cuyos outputs queremos combinar y qué campos incluir. También permite, en Make, definir si esperamos un número fijo de *bundles* o agrupar por alguna clave (*group by*) antes de emitir (Make, 2025a). Tras el agregado, el siguiente módulo verá una colección en un solo bundle en lugar de muchos separados.

Figura 7. Array aggregator

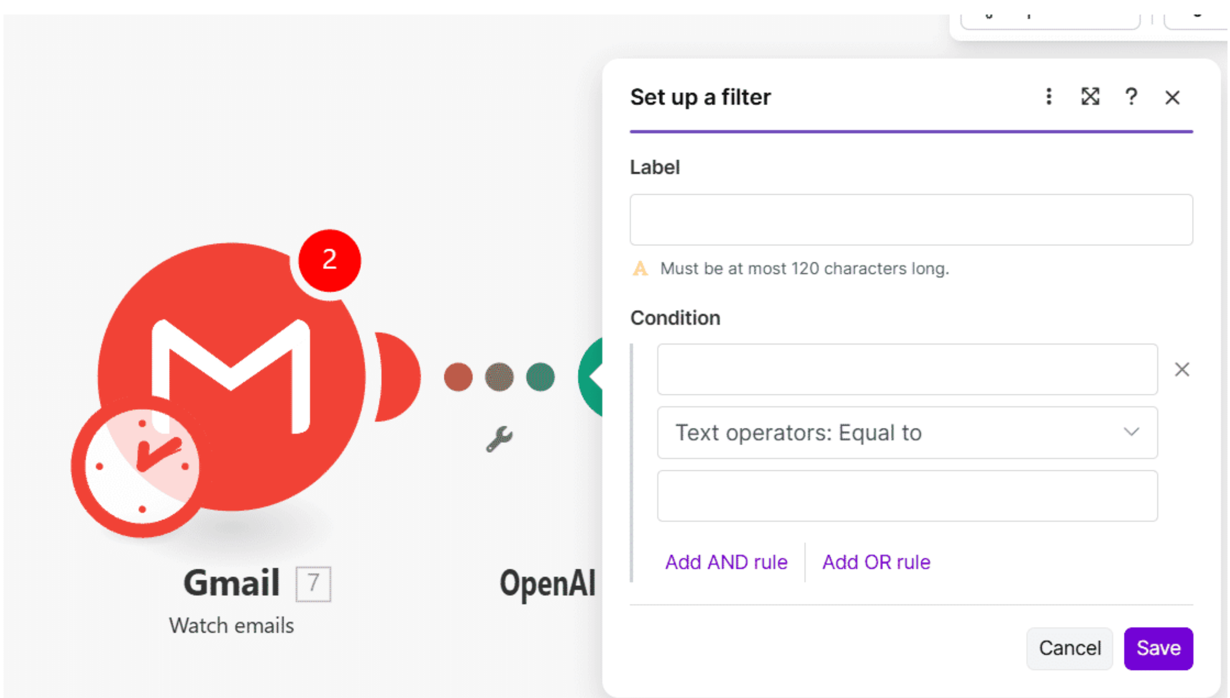


Fuente: elaboración propia.

Un elemento importante, aunque no es exactamente un módulo, son los ya mencionados filtros entre módulos: un filtro es una condición que decide si un *bundle* continúa al siguiente módulo o se detiene allí. Los filtros se configuran en la "línea" que une módulos y actúan como compuertas lógicas (*if-then*). En combinación con *routers* e *iterators*, los filtros hacen que un *Flow Control*

sea completo, permitiendo rutas condicionales sofisticadas. Por ejemplo, podríamos poner un filtro justo después de un iterador, para que, de los 100 elementos iterados, solo algunos avancen a cierto procesamiento especial (los que cumplan determinada propiedad). Los filtros no consumen operaciones extra (vienen incluidos en la lógica de Make) y son preferibles a implementar las condiciones "a mano" dentro de los módulos, porque brindan claridad visual.

Figura 8. ***Set up* Gmail**



Fuente: elaboración propia.

Módulos de Tools (Herramientas)

El grupo *Tools* en Make abarca módulos utilitarios destinados a transformar datos, almacenar variables temporales y otras funciones auxiliares dentro del escenario. No interactúan con apps externas, sino que operan sobre la información dentro de Make. Veamos los más destacados para nuestro contexto:

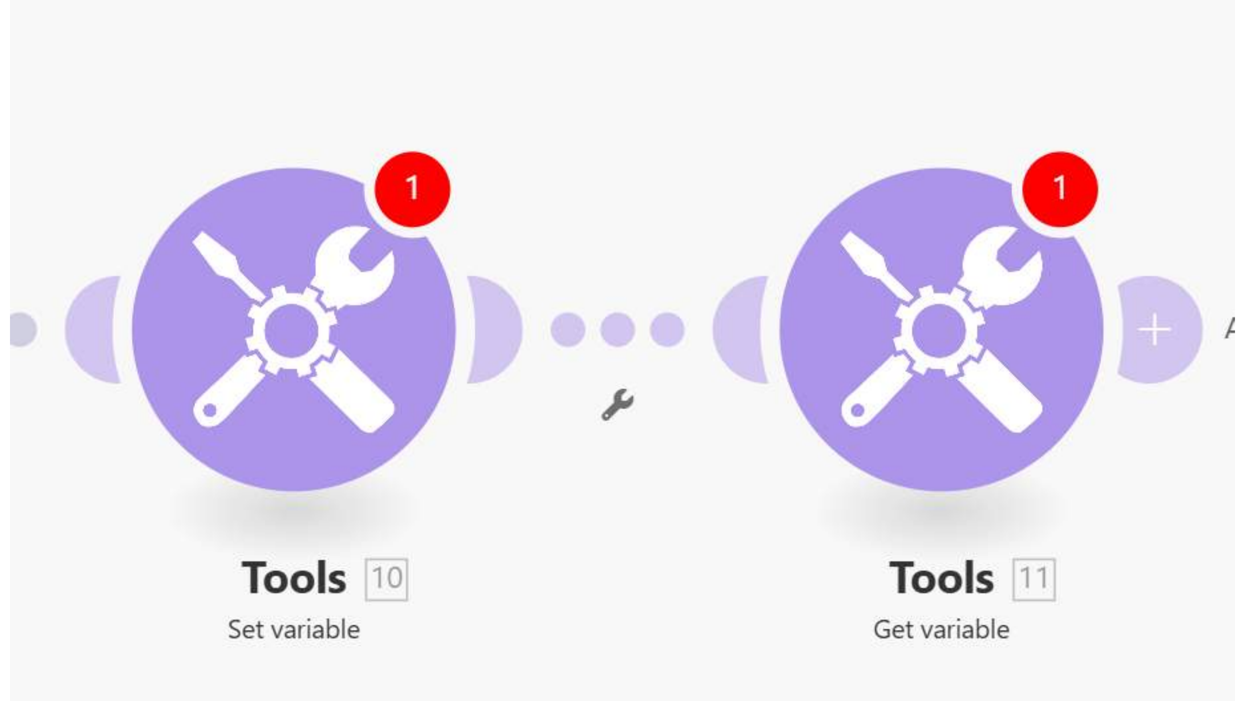
- ***Set Variable* / *Get Variable* (Variables de escenario):** estos módulos permiten crear y usar variables personalizadas durante la ejecución de un escenario. Son extremadamente útiles cuando necesitas reutilizar un dato en varios puntos del flujo sin tener que pasarlo explícitamente por cada módulo intermedio. Por ejemplo, quizá obtienes un valor al inicio (una ID de usuario, o la fecha actual) y más adelante, en otra ruta necesitas ese mismo valor. En vez de volver a calcularlo o arrastrarlo por los módulos, puedes guardarlo con Set Variable y luego recuperarlo con *Get Variable* donde lo requieras (Make, 2025b).

Set Variable crea una variable con un nombre que defines y le asigna un valor. Puedes elegir la "duración" de la variable:

- *One cycle* (un ciclo): significa que si en un solo "run" del escenario se procesan múltiples paquetes (por ejemplo, varios *webhooks* entrantes a la vez), la variable se reiniciará por cada uno. Generalmente, "*one execution*" es más común para nuestros fines.
- *One execution* (una ejecución): la variable vive a lo largo de toda la ejecución del escenario (que puede abarcar varias rutas). Al finalizar el escenario, la variable desaparece.

Luego, un módulo *Get Variable* puede usarse en otra ruta o más adelante para recuperar el valor guardado (suministrando simplemente el nombre de la variable). Un detalle importante es que el Set Variable debe ejecutarse antes en el tiempo que el *Get Variable*.

Figura 9. ***Tools***



Fuente: elaboración propia.

- ***Text Parser*** **(*Parseador* de texto):** este es un conjunto de módulos dentro de Tools diseñados para extraer o manipular texto mediante patrones. En integraciones avanzadas (por ejemplo, al procesar correos, *logs* o incluso salidas de IA que son texto estructurado), el *Text Parser* es invaluable para obtener justo las partes de texto que necesitamos. Make provee distintas opciones, siendo la más poderosa Match Pattern (coincidir patrón), que permite usar

expresiones regulares (*regex*) para buscar y capturar texto (Won_t_Tell, 2025).

Match Pattern: Con este módulo puedes definir un patrón de búsqueda utilizando una expresión regular, y el módulo encontrará en el texto de entrada aquellas partes que encajen con ese patrón, devolviéndolas como output. Las expresiones regulares son secuencias de caracteres especiales que representan reglas de coincidencia (por ejemplo, \d+ significa "una o más cifras consecutivas"). Aprender *regex* es muy útil, pero aun sin ser experto se pueden usar patrones sencillos o apoyarse en herramientas. Lo importante es que Match Pattern "atrapa" trozos de texto específicos dentro de un texto grande (Won_t_Tell, 2025). Por ejemplo, supongamos que recibimos vía *webhook* el cuerpo de un email y queremos extraer un número de pedido que siempre aparece como "*Order*: 12345" en algún lugar del texto. Podemos configurar Match Pattern con un *regex* como *Order*:\s*(\d+). Este buscaría la palabra "*Order*:" seguida de cualquier espacio y luego capturaría el grupo de dígitos como resultado. El módulo

entonces nos daría ese número como salida mapeable.

Otros submódulos de *Text Parser* incluyen:

- *Replace*: buscar y reemplazar texto (admite también regex o texto fijo) (Make, 2025c). Útil para limpiar *strings*, por ejemplo, eliminar espacios extra o ciertos símbolos.
- Parse HTML (*Get elements from* HTML): un módulo capaz de extraer contenidos específicos de código HTML, por ejemplo, todas las imágenes (<img>) o enlaces de un fragmento HTML (Make, 2025c). Esto es particularmente útil si recibes un email en HTML y quieres sacar solo los enlaces, o *scrapear* datos simples de un trozo de página web (aunque para *scraping* complejo se sugiere usar API especializadas).

- Otras herramientas útiles: Además de variables y *parseo* de texto, Tools incluye módulos como:

- *Set Multiple Variables* / *Get Multiple Variables*: similares a *Set/Get variable,* pero permiten establecer o recuperar varios valores en una sola operación (Make, 2025a), ahorrando operaciones cuando se manejan muchos datos.
- Sleep: para pausar la ejecución X segundos (Make, 2025a). Útil para esperar entre llamadas si una API lo requiere o para espaciar envíos masivos (simular comportamiento humano o no saturar servidores).
- *Iterator Control* (*Break, Continue*): permiten, en escenarios con iteración, cortar antes de terminar todas las iteraciones o saltar a la siguiente. Estos son avanzados y poco comunes, pero existen para casos específicos.
- Transformadores: por ejemplo, *Compose a string* (convierte cualquier dato a texto) (Make, 2025a), JSON > Create JSON/Parse JSON (ya mencionado, para construir JSON o desglosarlo), Aggregator (los agregadores de texto/tabla/número también se listan a veces bajo Tools) (Make, 2025a). Aunque los agregadores los cubrimos en Flow Control, vale decir que

hay un Text Aggregator en Tools que concatena fragmentos de texto de múltiples *bundles* en uno solo (Make, 2025a), se usa, por ejemplo, para juntar varias líneas de texto en un solo bloque (como unir todos los nombres de clientes en un solo campo, separados por comas, y luego enviarlos en un email).

En síntesis, los grupos Flow Control y Tools proveen los bloques lógicos para hacer que una automatización con Make pase de simple a avanzada y robusta. Los Flow Control (*routers, iterators, aggregators*, etc.) nos dan control sobre el camino que toman los datos, permitiendo ramificaciones paralelas, bucles y sincronización. Los Tools (*variables, parsers*, etc.) nos dan control sobre los datos mismos, permitiendo reformatearlos, almacenarlos temporalmente y extraer la esencia de ellos. Dominar estos módulos es esencial en el nivel avanzado, pues son la base para crear flujos inteligentes.

Nota aclaratoria sobre uso de IA

Este material fue asistido con herramientas de IA generativa para tareas de borrador, síntesis, reescritura y apoyo en la organización de contenidos. Cada sección fue revisada, editada y validada por el equipo humano, que verificó la precisión conceptual, la coherencia pedagógica y las fuentes citadas. Se invita a contrastar con las referencias bibliográficas incluidas y la documentación oficial. Dado que los modelos de IA evolucionan con rapidez, ciertas especificaciones técnicas podrían actualizarse; este texto refleja el estado del conocimiento al momento de su elaboración.

CONTINUAR

Referencias

Aslan, E. (s. f.). Flow control in Make.com. ConsultEvo. Retrieved January 8, 2026, from https://consultevo.com/make-com-flow-control-guide/

crm_automator. (2024, October 27). How to get OpenAI output into a JSON format [Forum post]. Make Community. https://community.make.com/t/how-to-get-openai-output-into-a-json-format/58927

Make Community (2023). Getting started with OpenAI GPT-3 and Make [Knowledge Hub post]. Make Community. https://community.make.com/t/getting-started-with-openai-gpt-3-and-make/7796

Make. (2025, December 12). Flow control. Make Help Center. https://help.make.com/flow-control

Make. (2025a, November 4). Tools. Make Help Center. https://help.make.com/util

Make. (2025b). Scenario variables. Make Help Center. https://help.make.com/scenario-variables

Make. (2025c). Text parser. Make Help Center. https://help.make.com/regexp

Marcelo_Barcia. (2025, March 24). Get specific text from a variable [Forum post]. Make Community. https://community.make.com/t/get-specific-text-from-a-variable/76232

PDF.co. (2024, September 11). Summarize scanned PDFs with ChatGPT using PDF.co and Make. PDF.co Tutorials. https://pdf.co/tutorials/summarize-scanned-pdfs-chatgpt-and-make

Tan, S. A. (2023). ChatGPT integrations: A guide for beginners. Make. https://www.make.com/en/blog/chatgpt-integrations-guide

Won_t_Tell. (2025, April 21). Tools - set variable doesn’t return required output [Forum post]. Make Community.

https://community.make.com/t/tools-set-variable-doesnt-return-required-output/79506

CONTINUAR