A Comprehensive Guide to Pipelines

Last Modified on 09/16/2026 12:08 pm EDT

What You'll Learn in This Article:

- What a Pipeline is
- How Connections, Pipelines, and Tasks relate to each other
- How to build a Pipeline from scratch and attach it to a Campaign
- The full data payload available to a Pipeline, including quiz data
- How to add Conditions and chain multiple Tasks or Pipelines together
- How to test a Pipeline, read its activity, and troubleshoot common failures
- When to use one Pipeline vs. several
- How Pipelines differ from legacy Integrations, and how to think about migrating

This article is the starting point for working with Pipelines in Digioh. For platform-specific setup and error messages (Klaviyo, Iterable, Yotpo, etc.), see the **Platform-Specific Reference** section at the end, which links out to the dedicated article for each platform.

1. What Is a Pipeline?

Every Digioh Campaign (pop-up, quiz, inline form, banner, etc.) can send its submission data to one or more third-party platforms: your ESP, CRM, CDP, BI tool, or a custom endpoint. Digioh has two ways of doing this:

- **Legacy Integrations**: an older, single-step system where a form's fields are mapped directly to a destination, either through a simple field-mapping UI or a raw JSON template using bracket-style merge tags like [EMAIL] or [CUSTOM_1]. There is no visible sequence of steps, and no per-step logging.
- **Pipelines**: the current system. A Pipeline is a named tree of **Tasks** that runs automatically whenever a Campaign submission triggers it. Each Task does one job: map fields, wait, transform data, branch logic, or hand off to another Pipeline. Every Task's success or failure is individually visible in its activity.

The data flow:

Campaign form submission

↓

Digioh server receives the payload (form / analytics / attributes / prq)

↓

Pipeline executes its Tasks, following the tree from top to bottom

↓

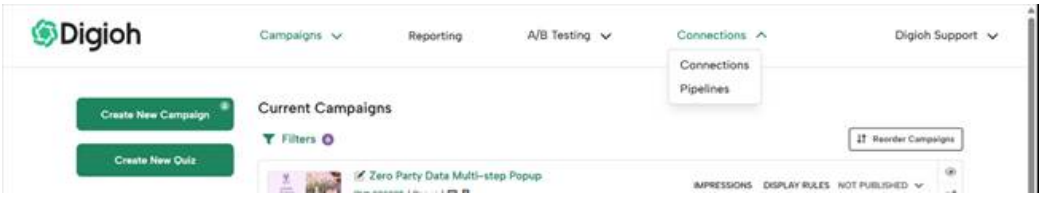
Data lands in one or more destination platforms (ESP, CRM, etc.)

Pipelines are separate from the legacy Integrations system. An account can have both in use at the same time, but a given Campaign→destination connection is built using one system or the other, not both.

Why Pipelines exist: legacy Integrations work well for a single, simple field-to-field mapping, but they break down once you need more than one step, for example creating a profile in your ESP, waiting a few seconds, and then adding that profile to a list; or sending different data to different platforms depending on what the user answered in a quiz. Pipelines make each of those steps an explicit, independently testable, independently loggable Task.

Finding your way around: in the top navigation, **Connections** is a dropdown with two destinations: **Connections** (connect and manage individual platforms) and **Pipelines** (build and manage them). This guide covers both.

> We recently named the "Integration" dropdown to "Connections." Older accounts will still have "Integrations" in the navbar.



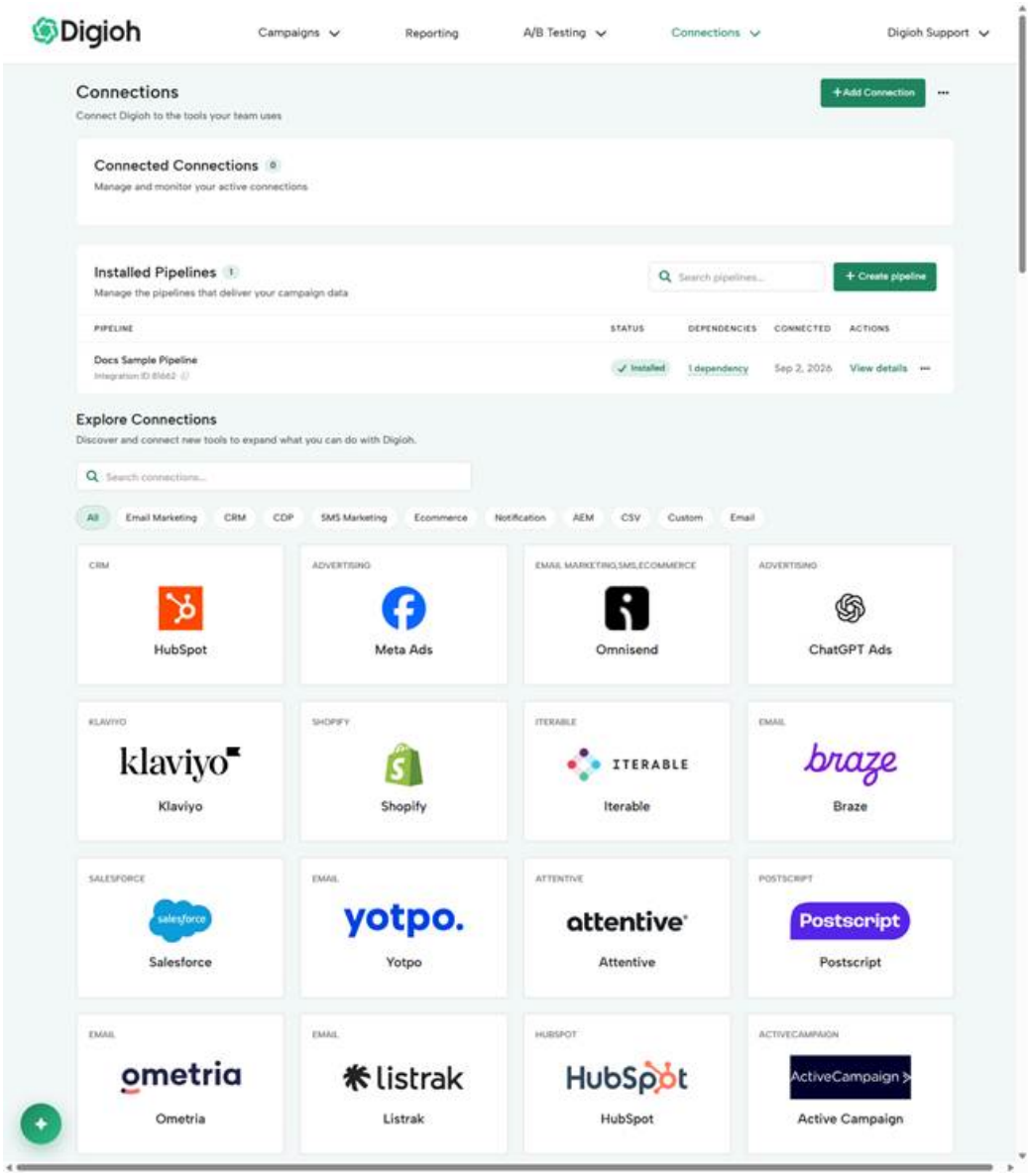
2. Core Building Blocks

Working with Pipelines involves three concepts: a Connection authenticates you to a destination, a Pipeline is the thing you build, and a Task is one step inside it. Here's how each one works:

Connections

A **Connection** is an authenticated link between your Digioh account and a specific third-party account: an API key, OAuth grant, or similar credential. You create a Connection once per destination account, then reuse it across as many Pipelines and Tasks as you need. Credentials are encrypted at rest on the Digioh server and are never exposed in the browser.

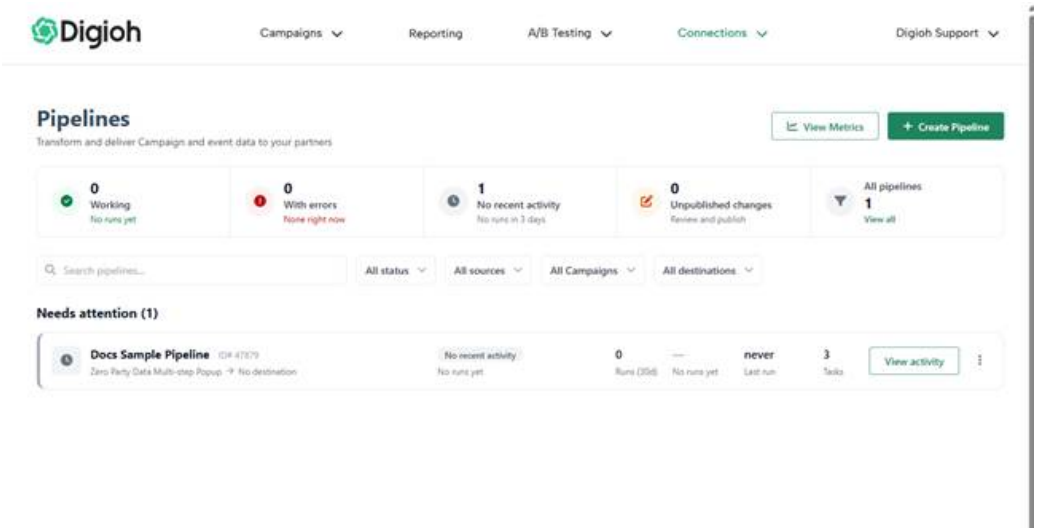
To create one, go to **Connections** in the top nav (the page itself, not the Pipelines page) and select **+ Add Connection**. This opens a searchable, category-filtered catalog (Email Marketing, CRM, CDP, SMS Marketing, Ecommerce, Notification, Admin, AEM, CSV, Custom). Pick a platform, then either enter your API credentials or complete its OAuth flow (e.g. Klaviyo redirects you to log in and click **Allow**). The same page lists every **Connected Connection** you already have, and every **Installed Pipeline** in the account, each with a status badge, its ID, and any dependencies.



Inside a Task, you'll pick one of these saved Connections from a **Connection** field so you don't have to re-enter credentials every time.

Pipelines

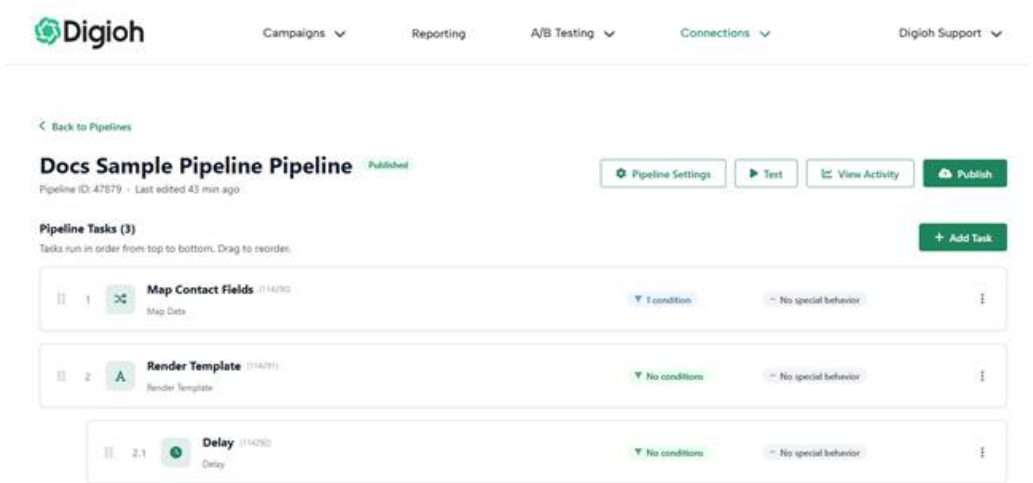
A **Pipeline** is the named thing itself: a tree of Tasks. Go to **Pipelines** in the top nav to reach the Pipelines dashboard: status tiles (Working, With errors, No recent activity, Unpublished changes, All pipelines), a search box, and filters by status, source, destination, and Campaign.



A Pipeline can optionally be tied to a Campaign or Quiz, or left unassigned and attached later (see Section 4). A single Pipeline can be attached to multiple Campaigns, and a single Campaign can have multiple Pipelines attached.

Tasks

A **Task** is one step in a Pipeline. Tasks are organized as a **nested tree**, not a flat list. You'll see numbering like 1, 1.1, 1.1.1, 1.2, 1.2.1, where a sub-task nests under the task whose output it depends on. To nest one task under another, open the parent task's **⋮ (more actions)** menu and choose **Add Subtask**. Drag-and-drop reordering works the same way it always has (click **Reorder**, drag, then **Save** or **Cancel**; nothing saves automatically while dragging), but now within that tree structure rather than a single flat sequence.



Every task row also shows two things inline, at a glance:

- A **Conditions** count: how many condition rules gate whether this task runs at all (see Section 6).
- A **Data Action** badge, one of three states. **Merge data**: this task's output merges into the data already flowing through the pipeline. **No special behavior** (the default): the task leaves the data untouched, and its tooltip spells out the consequence plainly: *"This task leaves the*

*data untouched, and a failure stops the pipeline."***Continue on failure**: the pipeline keeps running even if this task fails, the pattern to reach for when a task might legitimately fail and you'd rather branch around it with Conditions than halt everything (see the retry example in Section 6).

Opening any task shows three tabs: **Configuration** (the task's own settings), **Conditions** (its condition rules), and **Raw configuration** (the task's settings as raw JSON, handy for copying a complex setup into another task or Pipeline).

Task type reference:

The **Add Task** picker groups every Task type into three categories.

Transform Data: reshape or build a payload, without calling anywhere external.

Task Type	What it does
Map Data	Row-by-row field mapping: rename and reshape incoming fields into the shape a destination expects, with optional transforms and fallback values. The most common Task in any Pipeline.
Merge Data	Combines two or more prior data sources (Task outputs) into a single record before sending it on.
Render Template	Builds a fully custom JSON payload using a Liquid (recommended), Handlebars, or Scriban template. Use this when a destination expects a payload shape (nested objects, conditional fields, arrays) that Map Data alone can't produce. See Render Template in Depth below.
Transform Data	Encodes or converts a single field value. Available directly inside Map Data's Transform dropdown as well as a standalone Task; see the full transform reference below.

Flow Control: govern *when* and *whether* the pipeline continues.

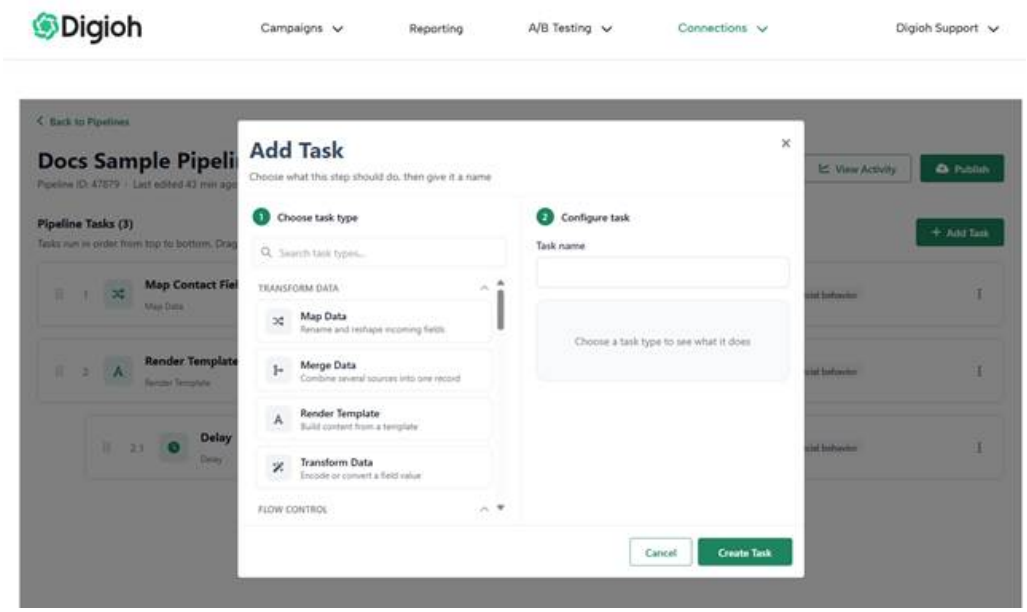
Task Type	What it does
Delay	Pauses the pipeline for up to a minute before the next Task runs; useful when a destination platform needs a moment to process a prior call (e.g. create a profile, then wait, then add it to a list).
Digioh Coupon	Reserves a coupon code for this contact.

Task Type	What it does
Send To Pipeline	Hands the record to another, already-built Pipeline (referenced by its Pipeline ID) for further processing, then returns control. Useful for reusing a mapping you've already built elsewhere instead of duplicating it.
Task Notification	Sends an email alert about this run to a comma-separated list of recipients; handy for flagging a human when something notable happens.
Terminate Pipeline	Stops the pipeline cleanly, without raising an error; use it as the "do nothing further" branch of a condition.

Send to Partner : call an external platform's API.

Task Type	What it does
Attentive	Subscribe contacts to SMS.
BigQuery	Run a BigQuery query.
Blueshift	Send customer data and events.
Gorgias	Create a support ticket.
Http Request	Call any endpoint with your own payload. The general-purpose escape hatch for a platform without a dedicated Task type. Supports custom headers, a configurable request body type (Form Data, JSON, text), a configurable response body type, and lets later Tasks reference its output.
Iterable API	Update users and send events.
Klaviyo	Create profiles, subscribe contacts, or track events.
Postscript	Subscribe contacts to SMS.
Salesforce Create Object	Create a Salesforce record.
Salesforce Get Object	Fetch a Salesforce record by id.

Task Type	What it does
Salesforce Query	Run a SOQL query.
Salesforce Update Object	Update a Salesforce record by id.
Shopify	Work with customers and products.
Snowflake Query	Run a Snowflake query.
Yotpo	Send data to reviews and loyalty.



New Task types are added routinely, particularly new platform-specific ones under Send to Partner; if a platform doesn't have a dedicated Task type yet, **Http Request** covers it.

Transform Reference

Every transform available in Map Data's Transform column (and as a standalone Transform Data Task):

Transform	What it does
Append Text	Adds text to the end of the value
Bool (true/false)	Converts to a true boolean. Use this instead of sending the literal string "true"
CSV to List	Splits a comma-separated string into a list

Transform	What it does
Date (e.g. 08/31/2026)	Formats as MM/DD/YYYY
Date (e.g. 2026-08-31)	Formats as yyyy-MM-dd
Date (with time, e.g. 2026-08-31 14:10:30)	Formats as yyyy-MM-dd HH:mm:ss
Date Custom	Formats using a custom pattern you supply
Date Sortable (e.g. 2026-08-31T14:10:30)	ISO-style sortable timestamp
Date Sortable with Offset	ISO-style timestamp including a UTC offset
Date Unix	Unix epoch timestamp
Hash MD5	MD5-hashes the value
HTML Decode	Reverses HTML Encode
HTML Encode	Escapes HTML-significant characters (e.g. &, <, >)
JSON Parse	Parses a JSON string into structured data
JSON Stringify	Serializes structured data into a JSON string
Lower Case	Lowercases the value
Number, Decimal	Converts to a decimal number, e.g. 123.45
Number, Integer	Converts to a whole number, e.g. 123
Phone to E.164 Format	Converts a phone number to E.164 (e.g. 5555551234 → +15555551234)
Prepend Text	Adds text to the beginning of the value
Remove HTML Tags	Strips HTML markup, leaving plain text
Remove Non Alpha Characters	Strips everything except letters
Remove Non Numeric Characters	Strips everything except digits
Replace Text	Replaces one substring with another

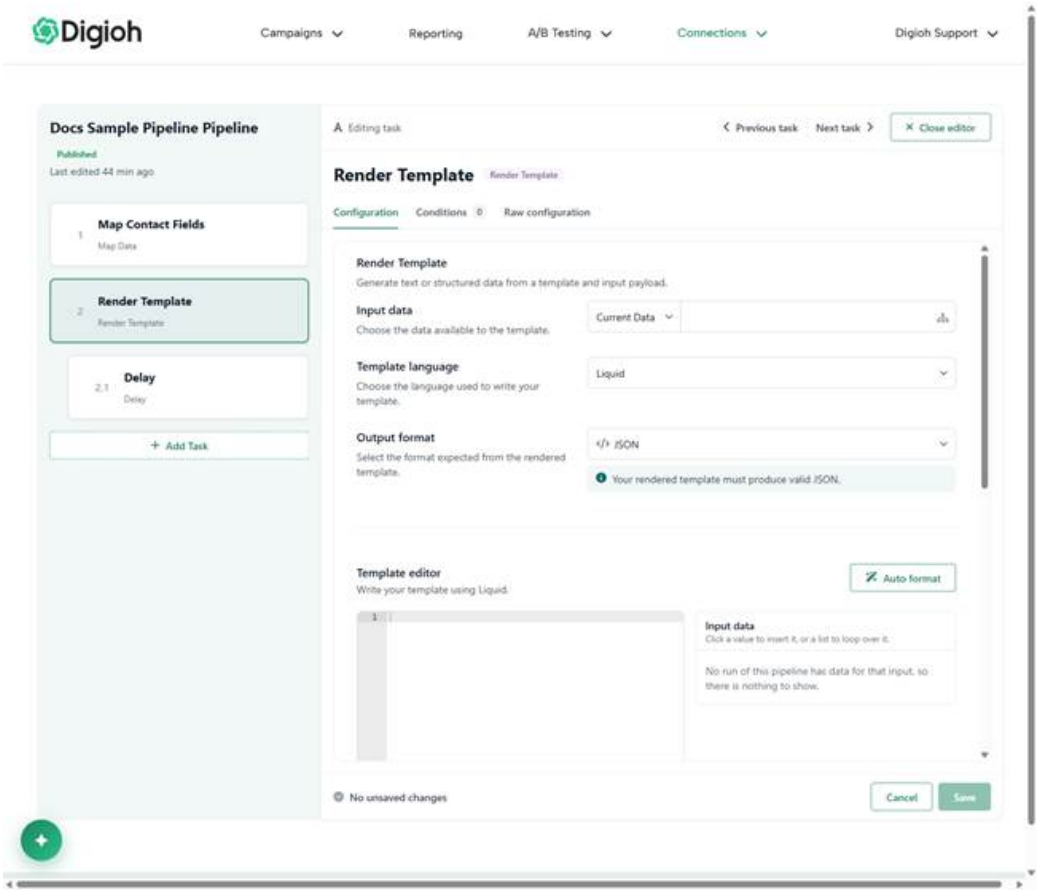
Transform	What it does
Replace Text Regex	Replaces text matching a regular expression
Sha256 Email	SHA-256-hashes an email address (common format for privacy-conscious ad/ESP integrations)
String Escape	Escapes special characters for safe embedding in a string
String Unescape	Reverses String Escape
Title Case	Capitalizes Each Word
Upper Case	Uppercases the value
URL Decode	Reverses URL Encode
URL Encode	Percent-encodes the value for safe use in a URL

Render Templates in Depth

Render Template renders a template against a data model. Its fields are:

- **Input data**: the data source available to the template, chosen from a **Current Data** dropdown plus a **Browse a recent run** icon for picking a specific field visually instead of typing a raw path. Typically this is the full submission payload, so form.*, analytics.*, attributes.*, and prq.* are all available under model. inside the template.
- **Template language**: Liquid (most common), Handlebars, or Scriban.
- **Output format**: almost always JSON; the task validates that your rendered output actually parses as valid JSON.

The template editor is a line-numbered code editor with an **Auto format** button, and a live **Input data** side panel that lists every field you can click to insert into the template, or loop over. Below the editor, a **Test data (JSON)** box lets you paste a sample payload and preview how the template renders against it; useful before the Pipeline has any real runs to pull sample data from.



1. Conditionally include a field only if it was actually submitted , which prevents sending a blank/empty key to a destination that rejects empty values:

```
{% if model.form.email != null and model.form.email != "" %}

"email": "{{model.form.email}}",

{% endif %}
```

2. Build a nested object. Many destinations (Iterable, Klaviyo, Salesforce Marketing Cloud) expect custom data under a nested key rather than flat fields:

```
{

 "email": "{{model.form.email}}",

 "dataFields": {

 "firstName": "{{model.form.first_name}}",

 "lastName": "{{model.form.last_name}}",

 "signupSource": "Digioh Campaign {{model.attributes.box_name}}"

 }

}
```

3. Loop over an array to build a list. Quiz results (prq.results) are a common case where you need every recommended product, not just the top one:

```
{
 "email": "{{model.form.email}}",
 "recommendedProducts": [
 {% for product in model.prq.results %}
 {
 "name": "{{product.name}}",
 "sku": "{{product.sku}}",
 "url": "{{product.url}}"
 }{% unless forloop.last %},{% endunless %}
 {% endfor %}
 ]
}
```

4. Combine a fallback default with dynamic data. Liquid's default filter fills in a value when a field is blank, similar to Map Data's "Fallback value," but usable inline within a larger template:

```
{
 "email": "{{model.form.email}}",
 "source": "{{model.analytics.web_source | default: "Direct Traffic"}}",
 "optIn": {{model.form.opt_in | default: "false"}}
}
```

5. Generate a timestamp for the moment the pipeline runs. The submission payload carries attributes.date_created (when the form was submitted), but not a "right now" value; build one with Liquid's date object:

```
{
 "todayDate": "{{ date.now | date.to_string '%Y-%m-%d %H:%M:%S' }}"
}
```

date.now returns the current timestamp, and date.to_string formats it using strftime-style tokens (%Y, %m, %d, %H, %M, %S). Useful for stamping a record with the time the task actually executed, which can differ from the submission time if a Delay task runs earlier in the tree.

6. Merge a prior task's result with a static value, then deduplicate it. Useful when you need to add an ID to a list a previous task already returned, without double-adding it if it's already there:

```
{% assign message_type_ids =
model.TaskResults[113315].dataFields.subscribedMessageTypeIds %}

{% assign message_type_ids = message_type_ids | concat: [141197] %}

{% assign message_type_ids = message_type_ids | uniq %}

{

 "subscribedMessageTypeIds": [

 {% for id in message_type_ids %}

 {{ id }}{% if forloop.last == false %},{% endif %}

 {% endfor %}

 ]

}
```

{% assign %} stores an intermediate value inside the template. The first line pulls a list off an earlier task's result using the same TaskResults[<task_id>].<Property> syntax Map Data uses (see Section 5); concat appends a static value onto that list; uniq removes any duplicate before the loop builds the final array. This is also a second, template-based way to write the "loop with a trailing comma" pattern from example 3: {% if forloop.last == false %},{% endif %} is equivalent to {% unless forloop.last %},{% endunless %}, just phrased the other way around; use whichever reads clearer to you.

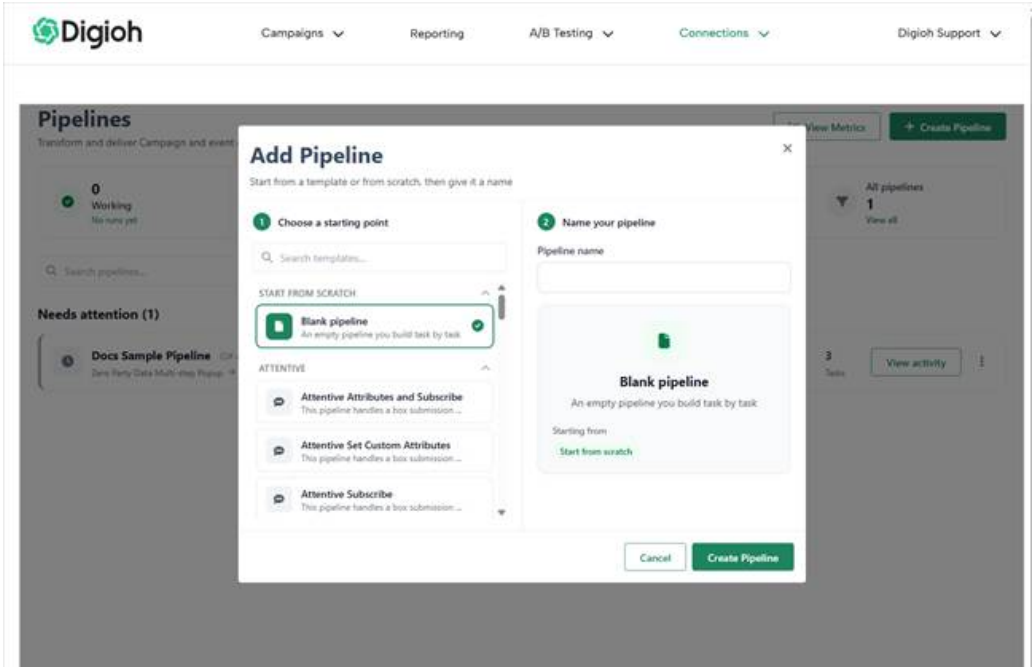
A syntax error or missing comma in any of these fails the entire Task, so build and test templates incrementally: add one field, confirm it renders correctly (use the Test data box above, or check the pipeline's activity after a real run), then add the next. For anything unusually complex, it's reasonable to draft the Liquid with AI assistance or ask Digioh Support for a second pair of eyes before publishing.

3. Creating Your First Pipeline

1. **Set up a Connection first**, if one doesn't already exist for your destination platform (see Connections above).

2. Go to **Pipelines** in the top nav and select **+ Create Pipeline**. This opens an **Add Pipeline**

dialog with a searchable, categorized template gallery (e.g. "Attentive Attributes and Subscribe"). Start from a matching template if one exists, or choose **Blank pipeline** to build from scratch. Name the Pipeline and select **Create Pipeline**.

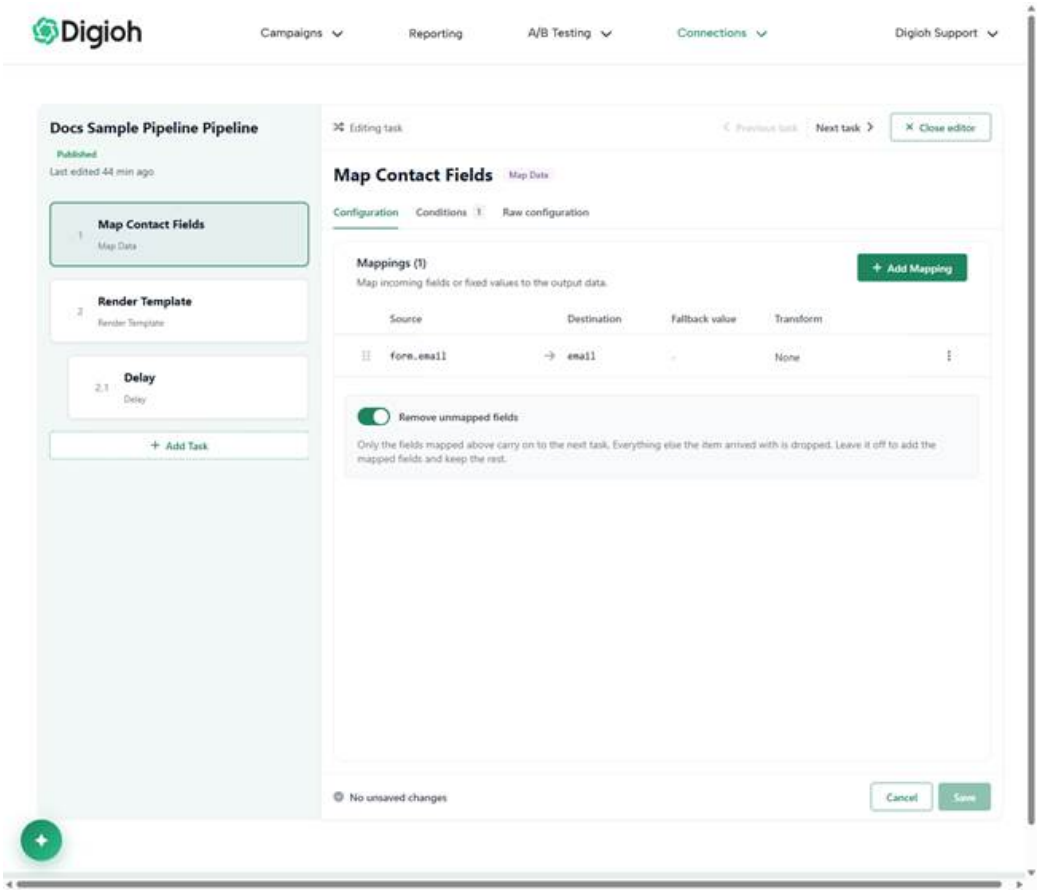


There's no Campaign or Quiz picker at this step; attach the Pipeline to a Campaign afterward (see Section 4).

3. **Add Tasks** with **Add Task**, building out the tree in the order and nesting you need. A typical pattern is:

- A **Map Data** task that builds the outgoing payload from form.* / analytics.* / attributes.* fields (and prq.* for quizzes)
- A nested **Send to Partner** task (or **Http Request**) that sends that payload
- Optionally, a **Delay** task followed by a second API task, if the destination needs the first call to finish processing before the second one runs (e.g. create-then-list-add patterns)

4. **Configure each Task.** For a Map Data task: for each row, choose **Data field** or **Fixed value**, set the **Source** (pick **Current Data** and either type a path or use the **Browse a recent run** icon), set the **Destination** exactly as the platform expects it (e.g. email or dataFields.phoneNumber), optionally set a **Fallback value** used when the source is empty, apply a **Transform** if the destination needs a specific format or type, and check **Treat as array** if the destination expects a list rather than a single value. Toggle **Remove unmapped fields** on to send only what you've explicitly mapped (recommended); leave it off to pass the mapped fields through alongside everything else the item arrived with. For a Send to Partner task: choose the Operation and the Connection to use.



5. **Save** each Task, then **Publish** the Pipeline as a whole. Changes to a Pipeline are staged and are **not live until you Publish**; this lets you build and test without risk of half-finished changes affecting live traffic.

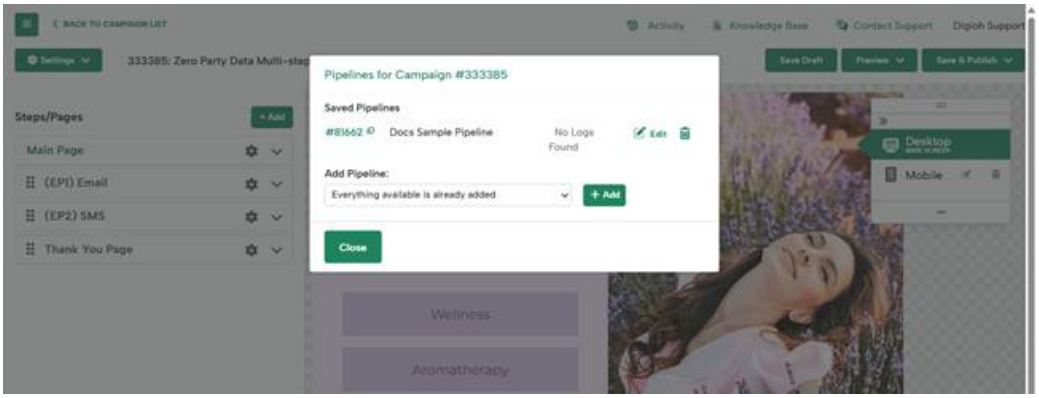
6. **Test it.** Submit the Campaign live (not in Preview; see the payload caveat below) and check the Pipeline's **View Activity** page to confirm it ran and each Task succeeded (see Section 7).

4. Attaching Pipelines to Campaigns

A Pipeline only runs once it's linked to a Campaign; creating a Pipeline on its own does nothing until it's attached.

From the Campaign List: select **Connections** from a Campaign's additional options menu, then choose which Pipeline(s)/Connection(s) to add.

From the Design Editor: open the **Settings** dropdown and select **Manage Connections**. This opens a **Pipelines for Campaign #\<id\>** dialog listing every Pipeline already attached (with **Edit** and delete icons, and a link to its logs) plus an **Add Pipeline** dropdown and **+ Add** button for attaching another.



Multiple Pipelines per Campaign: a Campaign can have more than one Pipeline attached, for example one that subscribes the user to an email list and a separate one that notifies your CRM. When more than one is attached, **they run in the order shown** , which matters whenever a later call depends on data created by an earlier one.

5. The Data Payload

Every Campaign submission passes a structured JSON payload into the Pipeline. Reference any field using **dot notation**, e.g. form.email, analytics.country_name, attributes.box_name, prq.results.[0].name. The full field list is documented in **Available Fields in Digioh Pipelines**; the four top-level objects are:

Object	Contains
form	The user's submitted data. The fields you map most often: form.email, form.first_name, form.last_name, form.phone, form.opt_in, form.custom_1 through form.custom_50, and form.main_* variants. For quiz submissions, form.named_custom_fields lets you reference an answer by its question name instead of a field number, useful when field order might change.
analytics	Visitor and session data captured automatically on every live submission: device, browser, operating system, geo-location, UTM parameters, referring/landing/submit URLs, page-visit history, and more.
attributes	Submission metadata: submission_id, box_id, box_name, user_guid, client_id, integration_id, pipeline_id, date_created, and similar system fields. Useful for logging and deduplication.

Object	Contains
prq	Present **only on quiz submissions**. Contains prq.answers (an array of every quiz answer, by page), prq.results (an array of recommended products, ranked by weight, each with name/description/price/image/url/sku/availability), and prq.results_url (a shareable, cross-browser link to the user's personalized results page).

Static values: in a Map Data task, choose **Fixed value** instead of **Data field** to hard-code a constant (e.g. always sending a specific list ID regardless of what the form captured) rather than pulling from the payload.

Working with arrays: prq.answers and prq.results are arrays; reference a specific item by its zero-based position:

prq.results.[0].name → the top recommended product's name

prq.results.[0].url → that product's URL

prq.results.[1].name → the second recommended product's name

prq.answers.[0].page_name → the first quiz page's name

Chaining output between tasks: one task's output can feed a later task. In the later task's Source picker, switch from **Current Data** to the name of the earlier task, then reference a field on its result using TaskResults[<task_id>].<Property>, e.g. TaskResults[109710].Profile.id. The task tree itself makes this visual: a task nested under another (added via **Add Subtask**) is typically consuming that parent task's result this way. The same syntax works inside a Render Template's Liquid, as model.TaskResults[<task_id>].<Property> (see example 6 in Section 2).

> Submitting a form via Digioh Preview intentionally strips some analytics and attribute values, and analytics.* fields will look empty. Always verify a Pipeline's mapping with a real, live submission, not Preview, before concluding a field isn't populating.

6. Conditions & Multi-Step Logic

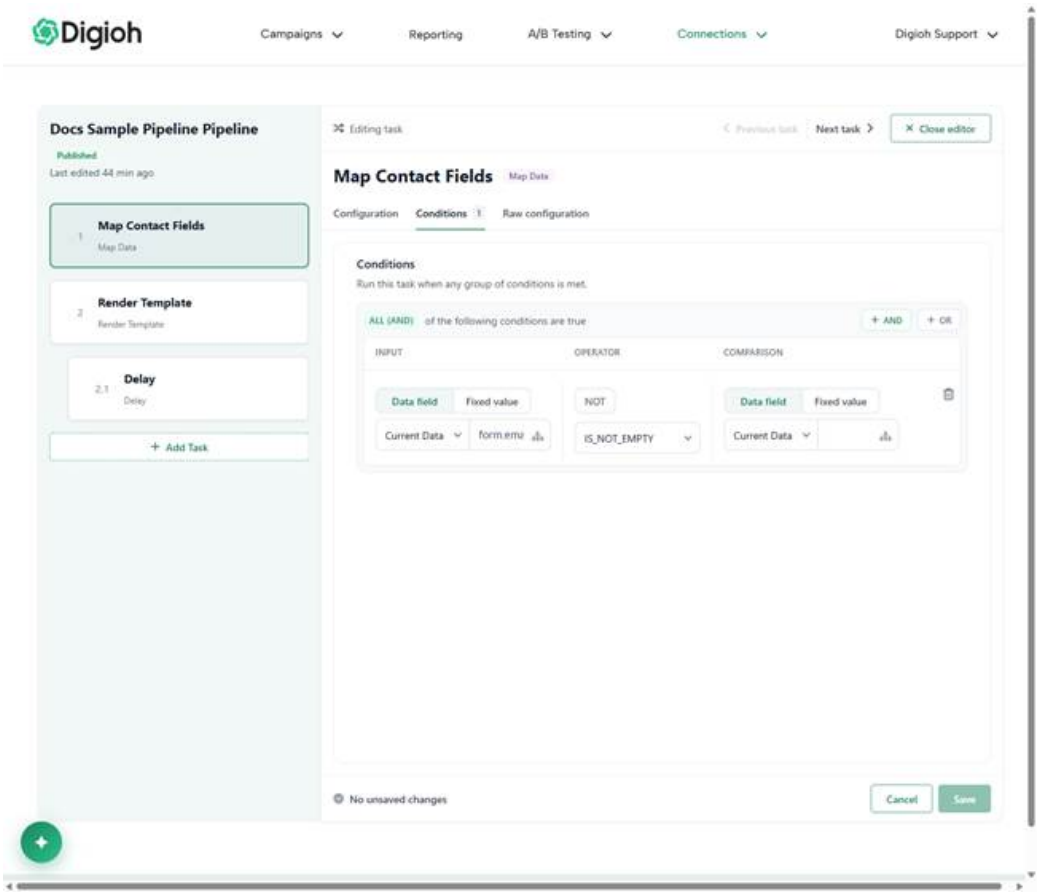
Any Task in a Pipeline can be given **Conditions** so that it only runs when specific criteria are met. Open the task and switch to its **Conditions** tab (the tab shows a live count of how many rules are configured).

Conditions are a real boolean expression builder, not a single simple check:

- Each **condition row** compares an **Input** against a **Comparison**, joined by an **Operator** (e.g. IS_NOT_EMPTY, STARTS_WITH), with an optional **NOT** toggle to negate it. Both the Input

and the Comparison side can be a **Data field** (from the payload, or a prior task's output, via the same **Current Data** picker used in Map Data) or a **Fixed value**.

- Rows are grouped into an **ALL (AND)** or **ANY (OR)** group; add more rows to a group with **+ AND** / **+ OR**.
- Groups themselves combine with each other: **+ Add AND group** adds another group joined by AND, and **+ Wrap all in OR** switches the top-level join to OR. This lets you build genuinely nested logic (e.g. *(A AND B) AND (NOT C STARTS_WITH "X")*) without needing a separate Pipeline for every variation.



Typical examples:

- Only run a "subscribe" Task if form.opt_in equals true
- Skip an SMS-related Task if form.phone **IS_EMPTY**
- Route to a different list or profile field depending on a quiz answer
- Only run a Task where analytics.country **NOT STARTS_WITH** a country code you don't support

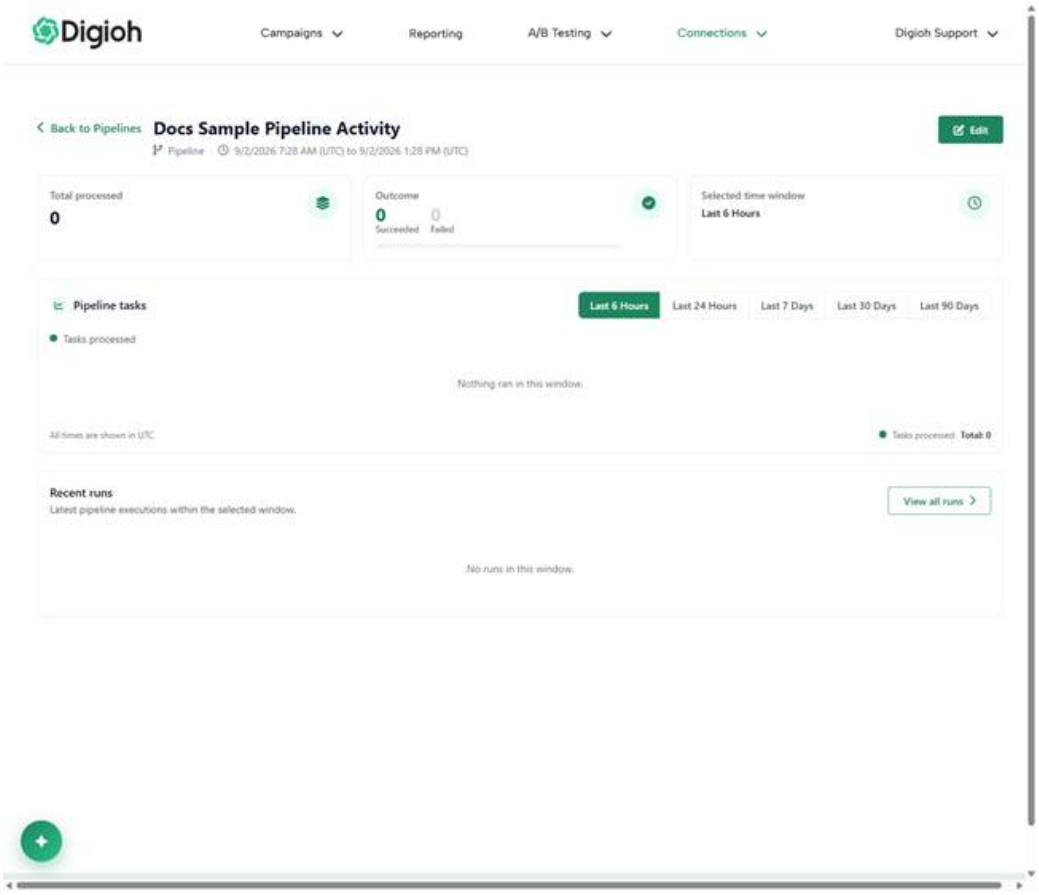
Combine Conditions with **Delay** (to sequence dependent calls), **Merge Data** (to combine multiple sources before a call), **Terminate Pipeline** (to cleanly stop a branch that shouldn't continue), and **Send to Pipeline** (to hand a sub-set of the payload off to a separate, reusable Pipeline mid-execution) to build branching, multi-destination workflows.

Retry pattern: this is also where **Continue on failure** (see Section 2) earns its keep. Set that Data Action on a task that might legitimately fail (say, updating a profile with a phone number the destination rejects), then add a follow-up task, gated by Conditions, that retries a fallback

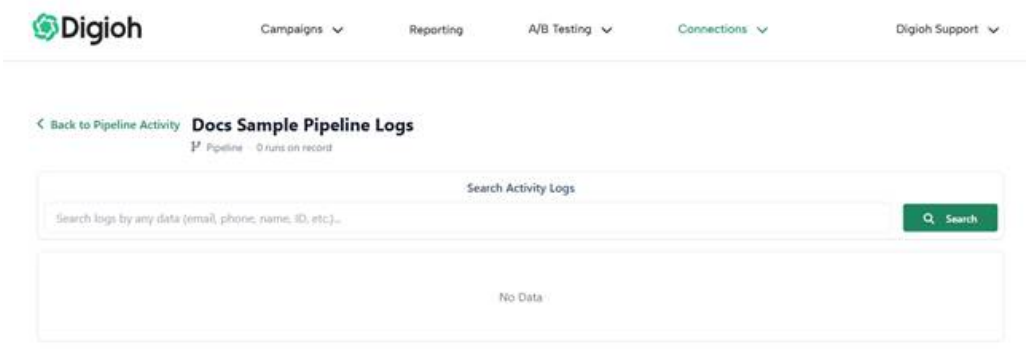
version of the same call (for example, the same update without the phone field). Because the first task was set to continue on failure, the pipeline keeps moving into that fallback branch instead of stopping cold.

7. Testing, Logs & Troubleshooting

Where to look: open a Pipeline and select **View Activity**. This opens an analytics-style dashboard first: **Total processed**, **Outcome** (Succeeded / Failed), and a time-window filter (Last 6 Hours / 24 Hours / 7 Days / 30 Days / 90 Days), plus a **Pipeline tasks** chart and a **Recent runs** list.



Select **View all runs** for the full run list, which now includes a **Search Activity Logs** box: search by any data the run carried (email, phone, name, ID, etc.), not just by date. Each run shows the outcome of every Task in the tree; drill in to see the full request sent and response received for a given Task, which is where destination-side error messages surface.



Log retention: how long runs stay searchable hasn't been reconfirmed as part of this update. The UI offers filtering as far back as 90 days, which is longer than previously documented here. Check with your Digioh contact for the current retention window before relying on a specific number.

Re-testing without a new submission: the pipeline editor has a **Test** action for re-running against sample or prior data without waiting for (or repeating) a live submission; useful right after you've changed a mapping. (Exact replay behavior, e.g. editing the payload before re-running, wasn't re-verified as part of this update; confirm the current flow directly in your account.)

"Nothing is sending" checklist, in order:

1. Has the Pipeline actually been **Published**? Unpublished changes never go live.

2. Is the Pipeline actually **attached to the Campaign**? A Pipeline exists independently of any Campaign and only runs once explicitly linked (see Section 4).

3. Does the Campaign's Form tab have an **override campaign integrations** setting that's skipping integrations entirely?

4. If the activity dashboard shows no runs at all, the Pipeline was never triggered. The issue is upstream of the Pipeline itself (points 1 through 3), not inside one of its Tasks.

Common in-Pipeline failure patterns:

- **A task earlier in the tree failed.** By default, a task's Data Action is "No special behavior," which, per its own tooltip, means *a failure stops the pipeline*. If a downstream task never ran, check whether something above it in the tree failed first, rather than assuming the downstream task itself is misconfigured. If a task is meant to fail sometimes by design, set its Data Action to **Continue on failure** instead (see Section 6 for the retry pattern this enables).
- **Invalid or expired API key** on the Connection: surfaces directly in the failure detail and is usually the first thing to check when every Task in a Pipeline fails at the same step.
- **Type mismatches at the destination.** Many destination platforms auto-create fields based on the literal type of the value they receive. Sending the string "true" instead of an actual boolean, for instance, can create a plain text field instead of a checkbox/boolean field,

which then makes that field hard to use in downstream flow/segment logic. Use the **Bool (true/false)** transform in Map Data to send the correct type the first time.
- **Blank required identifiers.** Some destinations require a specific top-level identifier (e.g. a top-level email address) separate from any custom fields; if that field isn't explicitly mapped, every submission can fail even though data appears elsewhere in the payload.
- **Rate limiting** on high-volume Campaigns (especially multi-step quizzes that call a Pipeline on every page) can produce 429-style errors from the destination. This is a volume issue, not a configuration one.

For errors specific to a given destination platform, check that platform's own Pipeline article (see Section 10); those go deeper on the exact error strings you'll see and how to resolve each one.

> Connections store credentials encrypted at rest on the Digioh server; they are never exposed to the browser. Data in transit uses HTTPS/TLS 1.2 or higher, and Digioh's endpoints sit behind CDN-level rate limiting and a firewall. If a use case requires reading data back from a destination (e.g. a preference center), Digioh supports additional safeguards: identity authentication to prevent substitution attacks, and profile-field limiting so only the fields you explicitly need are ever exposed to the browser.

8. Best Practices: One Pipeline or Several?

Whether to consolidate logic into one Pipeline or split it across several comes down to reuse and complexity:

- **Use separate Pipelines** when a flow needs to be reused across several Campaigns, for example a shared sign-up Pipeline attached to many Campaigns, while only some of those Campaigns also need a separate quiz-submission Pipeline.
- **Keep related steps in one Pipeline** when the logic is simple and self-contained. A typical "update profile, then subscribe to a list" flow is only a few Tasks and doesn't need to be split.
- **Avoid very large Pipelines** (e.g. 30+ Tasks). Smaller, focused Pipelines are easier to read, test, and troubleshoot, since a failure narrows down to one specific place rather than one step in a sprawling tree.

9. Migrating from Legacy Integrations

Legacy Integrations, the older field-mapped or JSON-template setups, continue to work, and there is no forced migration. The key differences to understand if you're maintaining or converting one:

	Legacy Integration	**Pipeline**
Structure	A single mapping or JSON template; no visible step sequence	A nested tree of independently configurable, independently loggable Tasks

	Legacy Integration	Pipeline
Field reference syntax	Bracket merge tags, e.g. [EMAIL], [CUSTOM_1]	Dot-notation JSON paths, e.g. form.email, form.custom_1
Multi-step logic (delays, conditions, chaining calls)	Not supported natively	Native, via Delay, Conditions, Merge Data, and Send to Pipeline Tasks
Per-step visibility in activity	No, pass/fail is for the whole Integration	Yes, each Task's success/failure and payload is individually visible
Reusable authentication	Credentials are entered directly into each legacy Integration	A Connection is created once and reused across any number of Pipelines
Publishing	Changes typically apply on save	Changes are staged and require an explicit Publish

When it's worth migrating: you need multi-step logic (a delay between calls, conditional branching, chaining several API calls), you want per-step failure visibility instead of a single pass/fail, or you want to reuse one Connection across multiple workflows instead of re-entering credentials.

When it's not urgent: a working legacy Integration doing one simple, single-step job has no functional reason to change. Reach out to Digioh Support if you'd like help planning a migration for a specific Integration.

10. Platform-Specific References

- Understanding Digioh Pipelines *(companion deep-dive to this guide: task types, the payload, logs, and conditions in full detail)*
- Klaviyo Pipelines in Digioh
- Iterable Pipelines in Digioh
- Iterable Pipeline Examples
- Yotpo Pipeline Integration
- Available Fields in Digioh Pipelines *(the full field list mentioned in Section 5)*