Understanding Digioh Pipelines

Last Modified on 06/17/2026 5:33 pm EDT

This article explains how Digioh **pipelines** work end to end: what a pipeline is, the most common task types, the structure of the data a form submission passes through, and how to read logs and add conditions. Read this first if you are new to pipelines or troubleshooting one. For ESP-specific setup and errors, see the companion Klaviyo and Iterable articles linked at the end.

What is a pipeline?

A pipeline is the layer between a Digioh form submission and your ESP or CRM (such as Klaviyo or Iterable). It receives the raw submission data, transforms it, and routes it to the destination through a series of tasks that run in order.

- Pipelines run automatically when a form is submitted.
- They transform and route submission data into the ESP using a sequence of tasks.
- They are separate from the legacy "integrations" system.
- Every execution is logged under **Pipeline › Activity**.

> **Flow:** Form submission → Pipeline (Digioh) → ESP / CRM (Klaviyo, Iterable, and others).

The three most common task types

A pipeline can contain many task types, but three account for the large majority of configurations:

Task	What it does
Map Data	Maps fields from the submission payload to the ESP's field names, row by row. This is where you set references such as form.email to email.
Klaviyo / Iterable API	Makes the actual API call to the ESP (subscribe to list, update profile, track event). Configuration is minimal by design - the payload is built in Map Data first.
Render Template	Builds a fully custom payload using a Liquid template. Commonly used when the ESP expects a specific JSON format that Map Data cannot produce on its own.

Map Data task

The Map Data task is a row-by-row field mapper. Each row maps a value (either from the submission payload or hardcoded) and writes it to an output field, generating a JSON that is passed to the next step in the pipeline.

Element	What it means
Field / Value toggle	Field pulls from the live submission payload. Value hardcodes a static value - for example, always sending a specific list ID regardless of what the form captured.

Element	What it means
Item Data dropdown	Tells the pipeline where to look for the input - Item Data (the submission payload) is most commonly used, but it can be switched to pull from a previous step's output.
Input field path	The dot-notation path to the field - e.g. form.email, analytics.country, prq.results_url.
Output Field	The destination field name, exactly as the ESP expects it - e.g. email, dataFields.phoneNumber.
Default Value	A fallback value sent if the input field is empty ("Fallback if empty").
Transform	An optional transformation applied before sending - e.g. Phone to E.164 turns 5555551234 into +15555551234, or date formatting.
Array checkbox	Check this when the output field expects a list of values rather than a single value (multi-select fields).
Remove Unmapped Fields	When checked, any payload fields not explicitly mapped are stripped before passing to the next step. Keep this checked to avoid sending unexpected or empty values.

> **Copy a task's full configuration:** The button in the top-right corner of a task opens its entire configuration as JSON. Copy it to reuse a complex mapping in another pipeline or account - paste it into the same button on the destination task instead of rebuilding the mappings row by row. (See the caveat under Render Template if the task references another task by ID.)

Chaining steps

One Map Data step can feed into another. Switch the **Input Field** dropdown from Item Data to the name of a prior step, then reference a field path inside that step's output. The output of Step A becomes the input for Step B.

For example, in a quiz pipeline the *Map Quiz Results* step can feed the *Map Final Event Payload* step:

quiz_results.[0].name → data.attributes.properties.prodRecName_1

quiz_results.[1].name → data.attributes.properties.prodRecName_2

Working with arrays

Some payload fields are arrays - sets of values rather than a single value (quiz answers and results are the common example). Reference a specific item by its position using bracket notation and starting with a zero-index:

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

prq.results.[0].url → product URL for the top recommendation

prq.results.[1].name → second recommended product name

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

Klaviyo / Iterable API task

This task makes the actual API call to the ESP. Configuration is intentionally minimal because the payload was already built in the Map Data step before it.

Element	What it means
Operation dropdown	The action to perform - e.g. Subscribe to List, Get Profile By Email, Track Event.
Connection	The saved API credentials to use (e.g. the KlaviyoDigioh connection), set up under the account's integrations.
Path to Email / Payload	Where the email lives in the current payload - usually form.email via Item Data or email via the output of a prior Map Data step.
Info panel (blue box)	Updates dynamically based on the selected operation. Shows what the task requires and links to the relevant API documentation - always worth checking for an unfamiliar operation.

> **Iterable is the same structure:** The Iterable APItask mirrors the Klaviyo one, just pointed at a different connection.Some actions add fields - for example, Iterable's subscribe action alsoasks you to select a subscription group.

Render Template task

The Render Template task builds a fully custom payload from scratch using a templating language. Use it when the ESP expects a very specific JSON structure that Map Data cannot produce - for example nested objects, conditional fields, or arrays of subscriptions.

Element	What it means
Path to Model	Where the template pulls its data from - typically Item Data (the full payload), so all form.*, analytics.*, attributes.*, and prq.* fields are available.
Template Type	The templating language: Liquid (most common), Handlebars, or Scriban.
Output Type	The format of the rendered output - almost always JSON.
Template editor	Where the template is written. Reference fields as {{model.form.email}}. Liquid supports conditionals and loops.

Reading an example template:

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

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

{% endif %}
```

This includes the email field in the outgoing payload only if the user actually submitted an email -

preventing a blank email key from reaching the ESP.

Powerful but fragile: A syntax error or missingcomma in the JSON will fail the entire step. Reserve Render Template forcomplex ESP requirements or conditional field inclusion. You can use tryto use AI for help with Liquid, or reach out to Digioh Support

The item data payload

When a form is submitted, Digioh passes a structured JSON payload with four top-level objects. You reference fields using dot notation (e.g. form.email, analytics.country_name, prq.results_url). A full list of the available fields is available here.

Object	What it contains
form	The user's submitted data - what you map most often (form.email, form.first_name, form.custom_2, form.main_email).
analytics	Visitor and session data, captured automatically on every submission - device, browser, UTMs, geo, page URL, visit history. Useful for enriching ESP profiles.
attributes	Submission metadata - submission_id, box_id, box_name, pipeline_id, date_created, user_guid. Useful for logging and deduplication.
prq	Quiz-specific data, present on quiz submissions only - the answers array, results array, and results_url.

Named custom fields: For quiz submissions, form.named_custom_fields lets you reference answers by the actual question name instead of a field number - handy when field order might change.

Preview mode: Analytics fields do not populate when a form is opened in preview mode - visitor and session data is only captured on live submissions. If analytics.* values look empty while testing, confirm you are not in preview.

Quiz data: the prq object

The prq object is only included on quiz submissions. It contains three sub-objects:

Sub-object	What it contains
prq.answers	Array of all quiz answers, organized by page. Each entry includes the page name, the button value (display label), and the custom value (what gets stored).
prq.results	Array of recommended products, ranked by weight. Each product object contains name, description, price, image, url, sku, and available.
prq.results_url	A shareable URL that loads the user's quiz results - works cross-browser, not just from local storage.

Because prq.answers and prq.results are arrays, reference individual items by position:

prq.answers.[0].page_name → "Bike type" (first quiz page)

prq.answers.[1].entries.[0].value → "Performance Oriented" (display label)

prq.results.[0].name → "KICKR CORE 2" (top product)

prq.results_url → shareable results page URL

Mapping the results page URL: On every quizsubmission, prq.results_url is populated with a shareable link to theuser's results. Map it to an ESP profile property (e.g.digioh_results_url) to store the personalized link on the contact recordand reuse it in downstream email flows.

Accessing and reading logs

Pipeline logs live under **Pipeline › Activity**. Each log shows the full request sent to the ESP (your mapped payload) and the full response (where errors surface).

- You can re-run a log from the log detail view using **Test Pipeline** - useful for testing a fix without waiting for a new live submission. On quiz pipelines, this reprocesses the full payload without retaking the quiz. You can even edit the previewed payload (e.g. change the email) before re-running.

Empty logs: If the pipeline logs are completely empty, the pipeline was never triggered - see "Troubleshooting: nothing is sending" below.

72-hour retention: Logs are retained for only 72 hours, after which they are no longer available.

Adding conditions to tasks

Any task in a pipeline can be given a condition so it only runs when specific criteria are met. This is key for multi-step pipelines that route data differently based on what the user submitted. Any payload field (form.*, analytics.*, attributes.*, prq.*) can drive a condition. Examples:

- Only run a Klaviyo step if form.opt_in equals true.
- Skip the SMS step if form.phone is empty.
- Route to different lists based on a quiz answer.
- Run a step only for US visitors (analytics.country = US).

Conditions are configured similarly to a Map Data step.

One pipeline or several?

Whether to split work across multiple pipelines or keep it in one depends on reuse and complexity.

- Use **separate** pipelines when a flow should be reused across several campaigns - for example a shared sign-up pipeline that you connect to many campaigns, while only some of those campaigns should also trigger a quiz-submission pipeline.
- Keep related steps in **one** pipeline when the logic is simple - a typical Iterable flow of update profile, then subscribe to a list or message type, is only a few tasks.
- Avoid very large pipelines (e.g. 30 tasks). Smaller, focused pipelines are easier to troubleshoot because you are only looking at one place.

Troubleshooting: nothing is sending

If submissions are coming in but no data reaches the ESP **and the pipeline logs are completely empty**, the pipeline was never triggered. Check the connection before investigating the pipeline itself:

- Make sure you've **Published** the pipeline.
- The pipeline must be connected to the campaign. A pipeline exists independently and only runs when it is explicitly linked.
- On the form tab, check the *override campaign integrations* setting. If it is set to skip integrations, the pipeline will not run even when it is connected at the campaign level.