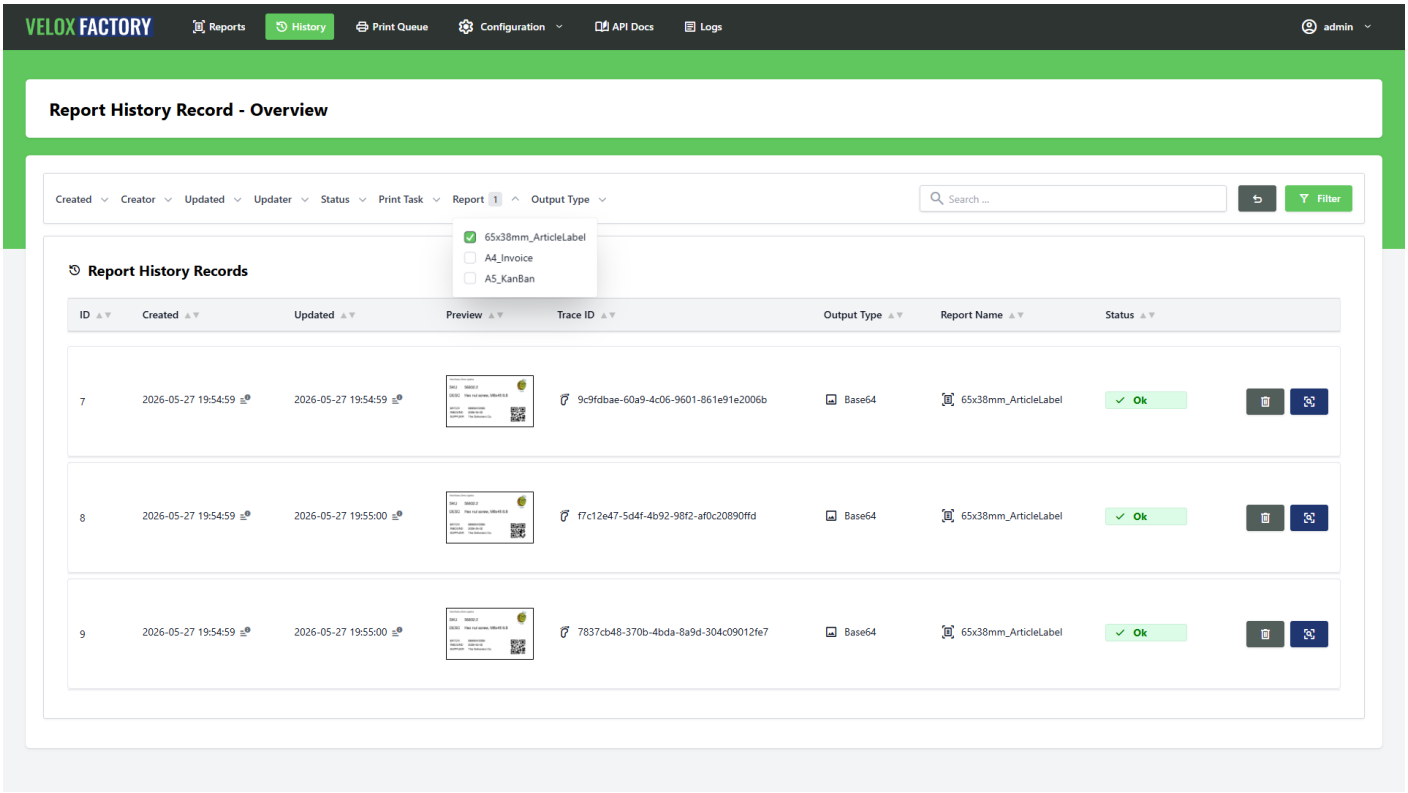
The concept of Report History Records

Every render request tells VeloxFactory what to produce. A `ReportHistoryRecord` remembers exactly what was asked for, what came back, and what the result looked like — permanently, until you decide otherwise. It is the foundation for traceability, debugging, and on-demand reprinting in VeloxFactory.



What Gets Stored

A `ReportHistoryRecord` is created at render time when `createHistoryRecord: true` is set in the request — or automatically when a print task is dispatched. It captures a complete snapshot of the rendering event:

Field	Description

`traceId`	Unique identifier shared across the render request, the history record, and any linked print task. Used to correlate events in logs and across systems.
`reportConfig`	Reference to the `ReportConfig` that was rendered.
`outputType`	The output type used: `base64`, `url`, or `none`.
`apiPayload`	The complete render request body — parameters, data, flags, everything sent to the render endpoint. Stored as JSON.
`apiResponse`	The complete API response returned by VeloxFactory — including any errors. Stored as JSON.
`reportPdf`	The rendered PDF, Base64-encoded. Present on successful renders; `null` on failure.
`reportPdfFileName`	The UUID-based filename assigned to the rendered PDF.
`reportThumbnail`	A thumbnail image of the first page of the rendered PDF. Generated asynchronously in the background after the record is created.
`status`	Automatically calculated from the stored response. See below.

Status

The status of a `ReportHistoryRecord` is calculated automatically every time the record is saved, based on the content of the stored API response:

Status	Meaning
Ok	No errors in the response and a PDF was produced. The render completed successfully.
Error	The response contains one or more errors. The render failed — the error messages are stored in the API response payload.
Render Fail	No errors in the response, but no PDF was produced either. An edge case indicating something unexpected occurred during rendering.
Unknown	Status could not be determined from the stored response.

History records are created for both successful and failed renders. A failed render still produces a complete record — including the error messages — which is often more useful than a successful one when something goes wrong.

The Thumbnail

When a history record is created, VeloxFactory dispatches a background job that converts the first page of the rendered PDF into a thumbnail image using `pdftoppm` (part of `poppler-utils`). The thumbnail is stored on the record and displayed in the history list and the report card grid — giving you an immediate visual of what was produced without opening the PDF.

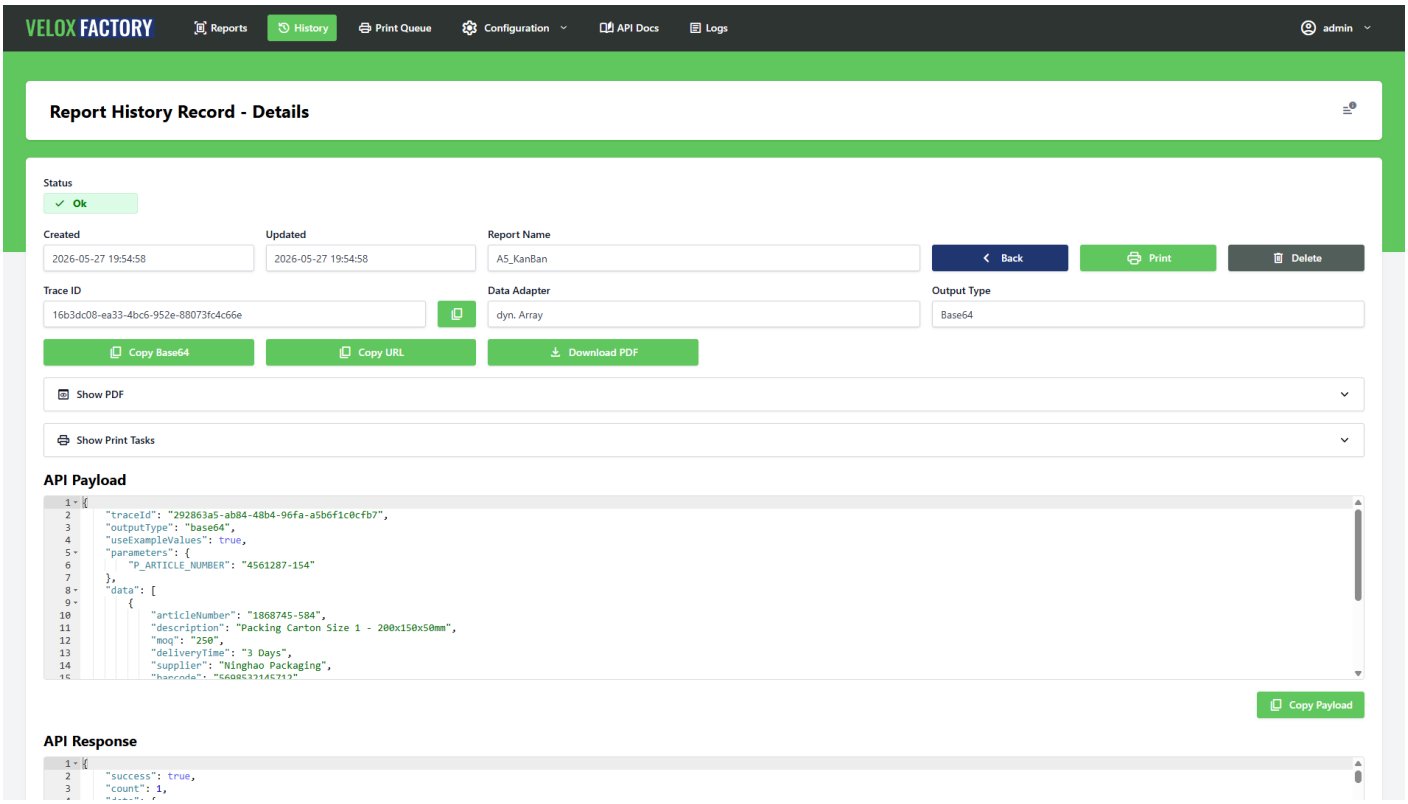
> ℹ **Thumbnail generation runs asynchronously.** The history record is available immediately after rendering; the thumbnail appears once the background job has completed. This requires the Laravel queue worker (Supervisor) to be running. If the queue is down, thumbnails will not be generated until it is back up.

Traceability and Debugging

The most valuable aspect of a history record is not the PDF — it is the payload. Every record stores the exact request that triggered the render and the exact response that came back. This means you can answer the following questions at any point in the future, without touching the calling application:

- What parameters were passed to this render?
- What data was submitted?
- Was the render triggered via the frontend or the API?
- What did VeloxFactory return — and did it succeed?
- If it failed, what was the exact error message?

The `traceId` ties everything together. It is present on the history record, on any linked print task, and in the server logs. When something goes wrong in production and you have a `traceId`, you can pull the history record and reconstruct the entire event in seconds.



Reprinting from a History Record

A successful history record holds the rendered PDF. That PDF can be dispatched to a printer at any time — without re-rendering the report — using the dedicated print endpoint:

```
POST                                                    /api/v1/report-history-record/{id}/print
```

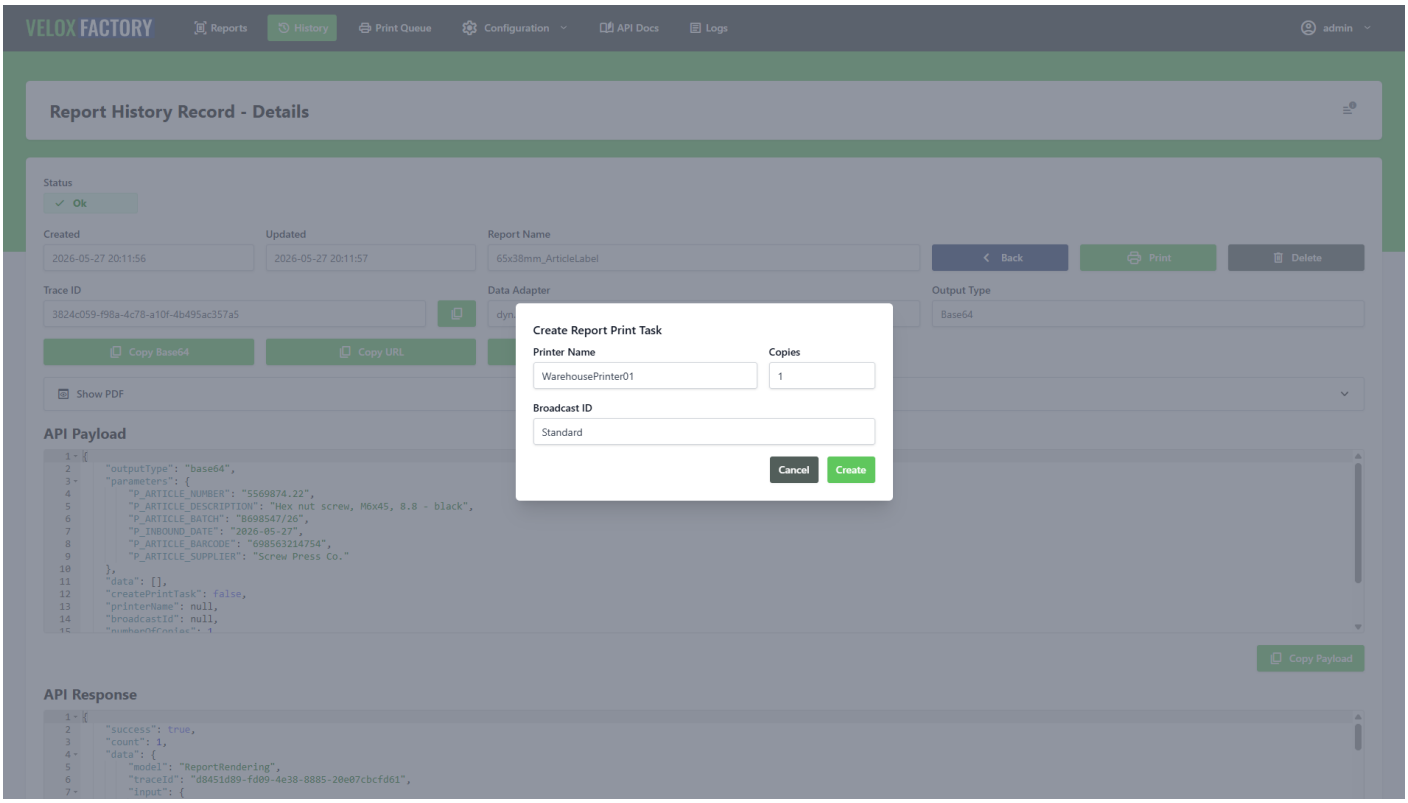
```
{



}
```

VeloxFactory creates a new `ReportPrintTask` from the stored PDF, assigns it a derived trace ID (the original trace ID with a short random suffix), and dispatches it to the print service. The original history record is linked to the new print task.

This is useful in several scenarios: a print job failed and needs to be retried, a physical document was lost and needs to be reprinted, or a record needs to be dispatched to a different printer than the one originally used.

> ℹ **Reprinting uses the stored PDF — it does not re-render the report.** The document produced is identical to the original. If the report template or its data has changed since the original render, those changes are not reflected in the reprint.

This endpoint is also available directly from the frontend — the history record detail view has a **Print** button that opens a modal to enter the printer name and number of copies.



Retention and Deletion

Automatic Purging

History records are automatically purged after a configurable number of days. The retention period is set via the `PURGE_HISTORY_DAYS` environment variable (default: 30 days). Purging runs as a scheduled background job — no manual intervention required.

Deletion Constraints

A `ReportHistoryRecord` cannot be deleted while any `ReportPrintTask` still references it. The linked print tasks must be removed first. VeloxFactory provides a dedicated endpoint for this:

```
DELETE                    /api/v1/report-history-record/{id}/delete-all-report-print-tasks
```

This removes all print tasks associated with the record in one call, after which the history record itself can be deleted.

Impact on ReportConfig Deletion

A `ReportConfig` cannot be deleted while any history records reference it. This is a deliberate constraint: the history exists as a permanent trace of what was rendered using that configuration. To remove a report configuration entirely, its history records — and their linked print tasks — must be cleared first, either manually or by waiting for the automatic purge to run.

Revision #6
Created 2026-05-10 10:44:49 UTC by Benjamin Fischer
Updated 2026-05-27 18:28:01 UTC by Benjamin Fischer