LYFTDATA PRODUCT DOCUMENTATION

Integrations

File, object-store, HTTP, SIEM, observability, Microsoft, and analytics integration guides.

Version 2.1.0. Generated from the docs.lyftdata.com source corpus on 2026-05-25.

Contents

Overview

All product integrations

Source: integrations/overview.md

LyftData connects to common object stores, analytics platforms, and SIEM destinations. Use this page to find available integration guides and jump to detailed setup instructions.

Integration Guides

Integration	Source inputs	Destination outputs
Amazon S3 & compatible object storage	`s3`	`s3`
Azure Blob Storage	`azure-blob`	`azure-blob`
FileStore (local object store)	`file-store`	`file-store`
Google Cloud Storage (GCS)	`gcs`	`gcs`
HTTP endpoints (webhooks, API polling, delivery)	`http-server`, `http-poll`	`http-post`, `http-get`
Grafana JSON datasource	—	`http-post`
Splunk Cloud / Enterprise	—	`splunk-hec`

Where to go next

- Use the table above to jump straight to setup guides and verify supported directions.
- Start with storage patterns such as S3 or GCS if you are moving files today.
- When wiring SaaS or SIEM destinations, combine these pages with the CI/CD automation guide so promotions stay predictable.
- Need help with a connector not listed here? Reach out through the LyftData community.

> Need a printable summary? Use the table above and filter on the direction that matches your use case.

CrowdStrike Detection Monitoring (CDM)

Read from CDM

Source: integrations/cdm.md

LyftData supports reading from CDM

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to read data from CDM

Add LyftData input **http-poll** to a job and configure:

CrowdStrike Falcon

Write to Crowdstrike Falcon LogScale

`Source: integrations/crowdstrike.md`

LogScale HEC API

- add LyftData output **splunk-hec**
- configure Crowdstrike ingest endpoint URL
- configure HEC token
- configure Splunk fields: index, host, source, sourcetype, timestamp
- see also: LogScale HEC

LogScale Ingest APIs

- add LyftData output **http-post**
- configure Crowdstrike ingest endpoint URL
- configure required headers (Authorization: Bearer foo)
- see also: LogScale Ingest APIs

S3

https://www.crowdstrike.com/tech-hub/ng-siem/crowdstrike-falcon-logscale-s3-ingest/

Elastic

Write to Elastic

Source: integrations/elastic.md

LyftData supports writing to Elastic

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to write data to Elastic

Add LyftData output **http-post** to a job and configure:

FileStore (Local Object Store)

Read from and write to the local filesystem object store

Source: integrations/filestore.md

LyftData includes a local object-store implementation (`file-store`) that mirrors cloud object store behaviour while keeping data on disk.

Configure LyftData to read from FileStore

Add the **file-store** input to a job. Key fields (job-spec names in kebab-case):

- `path` – root directory for the local bucket (required).
- `object-names` – file names or prefixes relative to the root. Leave empty to target every file exposed by the selected `mode` .
- `mode` – choose `list-objects` , `download-objects` , or `list-and-download-objects` .
- `ignore-linebreaks` – surface each file as a single event instead of newline-delimited events.
- `timestamp-mode` – derive timestamps from file metadata or a pattern in the file name.
- `include-regex` / `exclude-regex` / `maximum-age` – filter candidate files by pattern or by an age window (e.g., `10m` , `1d`).
- `fingerprinting` / `maximum-fingerprint-age` – enable dedupe and control fingerprint retention.
- `preprocessors` – gzip/parquet/base64/extension handlers executed after read.

Example: list and download gzipped NDJSON files from disk

```
input:
file-store: path: /var/log/lyftdata object-names:


  • exports/


mode: list-and-download-objects include-regex:


  • '\\.json(\\.gz)?$'


fingerprinting: true maximum-age: 2h preprocessors:


  • extension
```

Configure LyftData to write to FileStore

Add the **file-store** output to a job. Key fields:

- `path` – root directory where objects are created (required).
- `object-name` – literal file name (`name: ...`) or field reference (`field: ...`).

- `mode` – `put` writes files; `delete` removes them.
- `disable-object-name-guid`, `guid-prefix`, `guid-suffix` – control collision-avoidance GUID naming for writes. By default a GUID is appended as `/<uuid>`; keep the GUID enabled and set prefix/suffix if you want file-like keys.
- `input-field` – event field to persist; omit to serialize the entire event after preprocessors.
- `batch` & `retry` – configure batching and failure handling.
- `track-schema` – maintain `__SCHEMA_NUMBER` when writing JSON payloads.
- `preprocessors` – gzip/base64/extension handlers executed before writing.

See File Handling for details on GUID naming, batching, and `${}` expansions.

Example: archive processed events locally

```
output:
file-store: path: /var/lib/lyftdata/archive object-name: name:
processed/${partition||unknown}/summary guid-prefix: "-" guid-suffix: ".json.gz" input-field:
payload preprocessors:


  - gzip


track-schema: true
```

Example: delete successfully processed files

```
output:
file-store: path: /var/log/lyftdata object-name: field: object_name mode: delete
```

Delete operations expect the incoming event to include the file path relative to the root (as produced by the `file-store` input). Delete mode does not append GUID prefixes.

Google Analytics (GA4)

Draft pattern guide for ingesting GA4 exports

Source: integrations/ga4.md

GA4

LyftData supports importing Google Analytics 4 exports that land in Google Cloud Storage or Amazon S3.

Configure LyftData to read GA4 exports from Google Cloud Storage

Add the **gcs** input to a job. Common fields:

- `bucket-name` – GA4 export bucket (required).
- `object-names` – prefix pointing at the export folder (e.g., `analytics_123456/events/`).
- `mode` – `list-and-download` to enumerate daily files.
- `include-regex` – narrow results to GA4 export format, typically `"\\.parquet$"`.
- `timestamp-mode` – `last-modified` to process newest exports first.
- `fingerprinting` – deduplicate files across reruns.
- `credentials` – GA4 exports live in GCP; use a service account with `roles/storage.objectViewer`.

Example: ingest daily parquet exports from GCS

```
input:
gcs: bucket-name: analytics-prod object-names:


  • analytics_123456/events/


mode: list-and-download include-regex:


  • "\\.parquet$"


maximum-age: 3d fingerprinting: true timestamp-mode: last-modified credentials: service-
account: key: "${secret|ga4/gcs_reader}" preprocessors:


  • parquet
```

Configure LyftData to read GA4 exports from Amazon S3

If GA4 exports are mirrored to S3, use the **s3** input:

- `bucket-name` – destination bucket.
- `object-names` – export prefix (for example `ga4/events/`).

- `mode` – `list-and-download`.
- `include-regex` – match `.parquet` or `.json.gz` depending on the export.
- `access-key` / `secret-key` – credentials with list/get access.
- `preprocessors` – include `parquet` or `extension` so events are decoded into JSON.

Example: ingest GA4 parquet exports from S3

```
input:
s3: bucket-name: analytics-s3-mirror object-names:


  - ga4/events/


mode: list-and-download include-regex:


  - "\\.parquet$"


maximum-age: 3d fingerprinting: true timestamp-mode: last-modified access-key:
"${secret|ga4/s3_access_key}" secret-key: "${secret|ga4/s3_secret_key}" preprocessors:


  - parquet
```

Google Ad Manager (GAM)

Draft pattern guide for ingesting GAM report exports

Source: integrations/gam.md

GAM

LyftData supports collecting Google Ad Manager (GAM) report exports stored in Google Cloud Storage or S3-compatible buckets.

Configure LyftData to read GAM exports from Google Cloud Storage

Use the **gcs** input when GAM exports land in GCS:

- `bucket-name` – GAM export bucket (required).
- `object-names` – prefix such as `gam/reports/`.
- `mode` – `list-and-download` to fetch each report.
- `include-regex` – match `.csv` / `.csv.gz` depending on export configuration.
- `preprocessors` – use `extension` to decompress `.gz` and convert CSV lines to events if needed.
- `credentials` – service account with `storage.objects.get` and `storage.objects.list`.

Example: download GAM CSV exports from GCS

```
input:
gcs: bucket-name: gam-reports object-names:


  • reports/daily/


mode: list-and-download include-regex:


  • "\\.csv(\\.gz)?$"


fingerprinting: true timestamp-mode: last-modified credentials: service-account: key:
"${secret|gam/gcs_reader}" preprocessors:


  • extension
```

Configure LyftData to read GAM exports from S3

If reports are delivered to S3 or compatible storage, configure the **s3** input:

- `bucket-name` – destination bucket.
- `object-names` – GAM export prefix.
- `mode` – `list-and-download`.

- `include-regex` – match `.csv` or `.json.gz`.
- `access-key` / `secret-key` – credentials with list/get permissions.
- `preprocessors` – `extension` or `gzip` to decode compressed reports.

Example: download GAM exports from S3

```
input:
s3: bucket-name: gam-export-s3 object-names:


  - reports/daily/


mode: list-and-download include-regex:


  - "\\.csv(\\.gz)?$"


fingerprinting: true timestamp-mode: last-modified access-key: "${secret|gam/s3_access_key}"
secret-key: "${secret|gam/s3_secret_key}" preprocessors:


  - extension
```

Google Cloud Storage (GCS)

Read from and write to GCS

Source: integrations/gcs.md

GCS

LyftData ships first-class support for reading from and writing to Google Cloud Storage (GCS).

Configure LyftData to read from GCS

Add the **gcs** input to a job. Key fields (names shown exactly as they appear in the job spec):

- `bucket-name` – target bucket (required).
- `object-names` – list of object names or prefixes. Leave empty to operate on every object surfaced by the selected mode; be cautious with very large buckets.
- `mode` – choose `list-objects`, `download-objects`, or `list-and-download-objects` depending on whether you need metadata only, already-identified objects, or listing plus download.
- `ignore-linebreaks` – set when an object should be surfaced as a single event instead of newline-delimited events.
- `timestamp-mode` – derive timestamps from creation time, last modified time, or processing time to drive downstream filtering.
- `include-regex` / `exclude-regex` and `maximum-age` – reduce the candidate list before downloading by pattern or by an age window like `2h` or `36h45m`.
- `fingerprinting` and `maximum-fingerprint-age` – prevent re-processing by persisting object hashes, and control how long fingerprints are kept.
- `credentials` – provide a service-account JSON, application default credentials, or a `gs://` URL through the available `GcsCredentials` variants. Values can be inlined, supplied via `{{ }}` context placeholders, or injected via `${ }` runtime expansions (prefer `${secret|scope/name}` / `${dyn|NAME}` for secrets).
- `preprocessors` – configure gzip/parquet/base64/extension handlers for content transformation during ingest.

Example: list and download JSON objects

```
input:
gcs: bucket-name: analytics-prod object-names:


  • exports/daily/


mode: list-and-download-objects ignore-linebreaks: true include-regex:


  • '\\.json(\\.gz)?$'
```

```
maximum-age: 6h fingerprinting: true timestamp-mode: last-modified credentials: service-
account: key: "${secret|gcp/analytics_gcs_reader}" preprocessors:


  • gzip
```

This configuration lists objects under `exports/daily/`, filters to recent JSON or JSON.gz files, downloads each object once, and surfaces the entire payload as a single event.

Configure LyftData to write to GCS

Add the **gcs** output to a job. Key fields:

- `bucket-name` – destination bucket (required).
- `object-name` – literal or field-derived destination via `ObjectDestination`. Use `name: ...` for a fixed path or `field: ...` to reuse an event field.
- `mode` – set to `put` (default) to upload events or `delete` to remove objects by name.
- `disable-object-name-guid`, `guid-prefix`, `guid-suffix` – control the automatic GUID appended to uploads to avoid collisions. By default a GUID is appended as `/<uuid>`; keep the GUID enabled and set prefix/suffix if you want file-like keys.
- `input-field` – choose the field that contains the body to upload. When omitted, the full event is serialized after preprocessors run.
- `batch` & `retry` – tune batching for throughput and configure retries/timeouts for robustness.
- `track-schema` – enable when writing JSON so `__SCHEMA_NUMBER` is updated alongside the payload.
- `credentials` – reuse the same `GcsCredentials` forms as the input (service account key, application credentials, or `gs://` URLs).
- `preprocessors` – run gzip/base64/extension handlers before the payload is written to GCS.

See File Handling for details on GUID naming, batching, and `${}` expansions.

Example: upload processed events with file-like keys

```
output:
gcs: bucket-name: analytics-prod-archive object-name: name:
processed/${partition||unknown}/summary guid-prefix: "-" guid-suffix: ".json.gz" input-field:
payload preprocessors:


  • gzip


track-schema: true credentials: service-account: path: /etc/lyftdata/service-account.json
```

Example: delete source objects after successful processing

```
output:
```

```
gcs: bucket-name: analytics-prod object-name: field: object_name mode: delete credentials:
application-credentials: key: "${secret|gcp/analytics_gcs_writer}"
```

The delete output expects each incoming event to carry the object name (for example from the GCS input). Deletion runs without generating GUID prefixes.

Recommendations for files and folders

- keep individual files below 100–150 MB (compressed gzip or Parquet) for predictable processing latency
- organise exported data with directory-style prefixes such as `Y/M/`, `Y/M/D/`, or `Y/M/D/H/`

Grafana JSON API Datasource

Visualize LyftData explorer metrics from Grafana using the simpod JSON API plugin

Source: integrations/grafana-json.md

Overview

The simpod Grafana JSON API datasource can call LyftData's explorer APIs to build dashboards without a bespoke connector. Once configured, Grafana can surface job throughput, licensing volumes, and any future explorer datasets exposed by the server.

Prerequisites

- LyftData server 1.9 or later with the metrics explorer service enabled (default).
- Administrative access to the LyftData server to start it with additional CLI flags or environment variables.
- A Grafana instance where you can install plugins and add datasources.
- Network connectivity from Grafana to the LyftData server over HTTPS (recommended) or HTTP.

Step 1 — configure explorer authentication

The explorer routes (`/api/db/explorer/*`) require either an admin JWT or a fixed token. Grafana works best with the fixed token so dashboards can authenticate without user credentials.

Option A: supply the token via CLI flag

Start the server with a value that includes the literal header string you want Grafana to send. Including the `Bearer` prefix keeps it compatible with standard auth conventions.

```
cargo run --bin lyftdata -- run server \
--lyftdata-db-explorer-fixed-token "Bearer grafana-explorer-RANDOMHEX"
```

Replace `RANDOMHEX` with a random string (for example the output of `openssl rand -hex 16`).

Option B: use an environment variable

Set `LYFTDATA_DB_EXPLORER_FIXED_TOKEN` before launching the server (or in your service manager):

```
export LYFTDATA_DB_EXPLORER_FIXED_TOKEN="Bearer grafana-explorer-token"
cargo run --bin lyftdata -- run server
```

The value is stored in the settings database once the server boots. You can rotate it by restarting with a new value.

Step 2 — install the Grafana plugin

Install the datasource on your Grafana instance. For Grafana OSS or Grafana Enterprise:

```
grafana-cli plugins install simpod-json-datasource
systemctl restart grafana-server
```

Grafana Cloud users can enable the plugin from the cloud portal instead of using `grafana-cli`.

Step 3 — add the datasource in Grafana

1. Navigate to **Connections** → **Data sources** → **Add data source**. 2. Choose **JSON API**. 3. Fill in the basic configuration:

- **Name**: for example `LyftData explorer`.
- **URL**: `https://your-server-host/api/db/explorer` (include `/api/db/explorer` as the base path).
- **Access**: `Server` (so Grafana adds the custom header).

4. Under **HTTP settings** → **Custom HTTP Headers**, add:

- **Header**: `Authorization`
- **Value**: the exact token string configured above (for example `Bearer grafana-explorer-token`).

5. Save & test. Grafana performs a `GET /api/db/explorer/` request; the server replies with `200 OK`, confirming connectivity.

> If you prefer to use admin JWTs instead, generate a short-lived token and place `Authorization: Bearer YOUR_JWT` in the header list. Dashboards will stop working when the JWT expires, so fixed tokens are recommended for shared panels.

Step 4 — build panels with the explorer APIs

Switch to a panel and add a query using the JSON datasource. The plugin's **Builder** mode aligns with LyftData's explorer schema.

Metrics catalog

When the panel loads the first time, Grafana calls `POST /metrics`. LyftData returns:

```
[
{ "label": "Job Metrics", "value": "Jobs", "payloads": [ { "name": "selected_jobs", "type":
"multi-select" }, { "name": "selected_job_versions", "type": "multi-select" }, { "name":
"selected_workers", "type": "multi-select" }, { "name": "selected_series", "type": "multi-
select", "options": [ { "label": "Events In", "value": "events_in" }, { "label": "Rated Input
Bytes", "value": "rated_input_bytes" } ] } ] }, { "label": "License Volumes (30 Days)",
"value": "Volumes", "payloads": [] } ]
```

Pick **Job Metrics** to unlock the payload selectors.

Dynamic payload options

For multi-select fields without inline `options`, Grafana issues `POST /metric-payload-options` with the current payload. The server resolves job names, workers, and available versions from the metrics repository. The request body looks like:

```
{
"metric": "Jobs", "name": "selected_jobs", "payload": { "selected_workers": [] } }
```

Query execution

When you click **Run query**, Grafana sends `POST /query`. Example request:

```
{
"targets": [ { "refId": "A", "metric": "Jobs", "payload": { "selected_jobs": ["sample-job"],
"selected_series": ["events_in", "rated_input_bytes"], "selected_workers": [],
"selected_job_versions": [] } } ], "range": { "from": "2025-01-01T00:00:00Z", "to": "2025-01-
01T12:00:00Z" } }
```

The server responds with Grafana data frames (time series or tables) that the panel renders immediately.

Endpoint summary

Endpoint	Method	Purpose
`/api/db/explorer/`	`GET`	Used by Grafana's **Test** button; returns `200 OK` and an empty body.
`/api/db/explorer/metrics`	`POST`	Returns the metrics catalog shown in Builder mode.
`/api/db/explorer/metric-payload-options`	`POST`	Resolves dependent select lists (jobs, workers, versions).
`/api/db/explorer/query`	`POST`	Executes the explorer query and returns Grafana-formatted frames.

All routes require the `Authorization` header and are available when the server starts with metrics services enabled.

Troubleshooting

- **401 Unauthorized / 403 Forbidden**: confirm the `Authorization` header in Grafana exactly matches the configured fixed token. Restart the server with a new token if needed.
- **Plugin errors about empty metrics**: ensure the metrics subscriber service is running and that historical metrics exist; otherwise the catalog may be empty.
- **Timeouts**: reduce the time window or limit selected jobs; Grafana waits for LyftData to finish assembling the response before drawing the panel.
- **Dashboard uses JWTs**: rotate tokens before they expire or switch to the fixed token method to avoid interruptions.

With the datasource in place, you can mix explorer charts alongside other observability data in Grafana dashboards.

HTTP Endpoints (Webhooks, API Polling, Delivery)

Receive webhooks, poll APIs, and send events over HTTP with LyftData

Source: integrations/http-endpoints.md

LyftData can interact with HTTP endpoints in three ways:

- **Receive webhooks** with the `http-server` input.
- **Poll remote APIs** on a schedule with the `http-poll` input.
- **Deliver events** to downstream services with the `http-post` (or `http-get`) output.

This guide is operator-focused (recipes and common gotchas). For full schema details, use:

- `http-server` input
- `http-poll` input
- `http-post` output

Receive webhooks with `http-server`

Use `http-server` when an upstream system pushes events to LyftData (webhooks, alerts, callbacks).

Key fields:

- `address` (required) and optional `path` for routing.
- `tls` to enable HTTPS.
- `source-ip-field` to capture the caller IP.
- `custom-response` and `content-type` to control the response body.
- `only-body`, `json`, and `ignore-linebreaks` to control how request bodies become events.

Example: accept JSON webhooks over HTTPS

```
input:
http-server: address: 0.0.0.0:8443 path: /webhooks/ingest tls: cert:
/etc/lyftdata/certs/server.crt key: /etc/lyftdata/certs/server.key source-ip-field: source_ip
custom-response: '{"status":"accepted"}' content-type: application/json json: true ignore-
linebreaks: true
```

Notes:

- `http-server` is an ingestion surface. For authentication and rate limiting, place it behind your API gateway / reverse proxy.
- If the webhook sends non-JSON payloads, set `json: false` and parse later with actions.

Poll APIs with `http-poll`

Use `http-poll` to run scheduled API requests and emit the responses as events.

Key fields:

- `url`, `method`, `headers`, `query`, and optional `body` / `form-urlencoded-body`.
- `trigger.interval` or `trigger.cron` for scheduling.
- `payload-mode` (`auto`, `json`, `raw`, `binary`) to control how the body is interpreted.
- `ignore-line-breaks` and `document-mode` to control event framing.
- `response.*` to include status codes and headers in the emitted event.
- `retry` and `timeout` for reliability.
- `pagination` when the upstream API spans multiple pages.

Example: poll a JSON endpoint and capture status/headers

```
input:
http-poll: url: https://status.example.com/api/incidents trigger: interval: duration: 60s
method: get headers: Accept: application/json response: status-field: http_status headers-
field: response_headers response-field: body payload-mode: json ignore-line-breaks: true
```

Authentication recipes

API keys and bearer tokens (`auth.header-token`)

```
input:
http-poll: url: https://api.example.com/v1/events method: get auth: header-token: token:
"${secret|api/token}" headers: Accept: application/json payload-mode: json ignore-line-
breaks: true
```

This sets the `Authorization` header to `Bearer <token>` by default. Override `auth-header` or set `prefix: ""` when needed.

OAuth2 via Credential Manager (`auth.oauth2-via-credential-manager`)

```
input:
http-poll: url: https://api.example.com/v1/events auth: oauth2-via-credential-manager:
credential-id: "2b64d2a0-0b76-4cd0-9ef0-3f0ef3d2a9a1" credential-scope-mode: allowed payload-
mode: json ignore-line-breaks: true
```

Use `credential-scope-mode: custom` with `credential-scopes: [...]` when the API requires explicit scopes.

AWS SigV4 signing (`auth.aws-sig-v4`)

```
input:
http-poll: url: https://abc123.execute-api.us-east-1.amazonaws.com/prod/events auth: aws-sig-
v4: access-key-id: "${secret|aws/access_key_id}" secret-access-key:
"${secret|aws/secret_access_key}" region: us-east-1 service: execute-api payload-mode: json
ignore-line-breaks: true
```

Pagination patterns (including RSS/Atom)

`http-poll` supports multiple pagination strategies:

- **Cursor** (`pagination.strategy: cursor`): extract cursors and/or next-page URLs from a JSON response and persist the cursor in Worker KV (so the next run can resume).
- **Link header** (`pagination.strategy: link-header`): follow RFC8288 `Link: <...>; rel="next"` headers within a run (stateless; no checkpoint).
- **Query param** (`pagination.strategy: query-param`): increment a numeric query parameter (for example `?page=2` , `?paged=3`) and persist the “next value to request” (defaults to runtime artifacts; Worker KV is an explicit opt-in).

Example: RSS/Atom polling with WordPress-style `paged` pagination

This example polls an RSS feed, paginates via `?paged=` , parses XML into JSON, and expands RSS items into events.

```
name: rss-wordpress-news-poll
description: "Poll an RSS feed via http-poll and paginate via a page query-param."


input: http-poll: trigger: interval: duration: 300s url: https://wordpress.org/news/feed/
method: get headers: Accept: application/rss+xml ignore-line-breaks: true document-mode: true
payload-mode: raw


pagination: strategy: query-param param: paged start: 2 step: 1 max-pages-per-run: 10 #
checkpoint defaults to runtime-artifacts. # terminal-status-codes defaults to [404].
continue-if-body-regex: "<item>"


actions:


  - xml:


input-field: _raw remove: true


  - expand:


mode: events: skip-list:


  - "^/rss/channel/item/\\d+/"


exclude-non-empty-arrays: false
```

Deliver events over HTTP (`http-post`)

Use `http-post` (or `http-get`) when you need to push events to a downstream HTTP service (SaaS APIs, collectors, webhooks, SIEM/HEC endpoints).

Key fields:

- `url` supports `${}` runtime expansions for dynamic routing.
- `headers` to set auth and content type.
- `body` selects the event, a field, or a literal/template string.
- `batch` and `retry` control throughput and resiliency.

Example: batch and send events as a JSON array

```
output:
http-post: url: https://collector.example.com/events/acme-retail method: post headers:
Content-Type: application/json Authorization: Bearer ${secret|collector/token} body: event:
{} retry: timeout: 30s retries: 5 batch: mode: fixed fixed-size: 100 timeout: 250ms wrap-as-
json: true
```

Troubleshooting and common gotchas

- **401/403**: confirm token source (`{{ }}` context vs `${ }` runtime expansions), required scopes, and header names/prefixes.
- **429/rate limiting**: reduce schedule frequency, narrow the requested time window, and tune retries; avoid infinite pagination by keeping `max-pages-per-run` low until you confirm stop conditions.
- **Unexpected event splitting**: set `ignore-linebreaks: true` when the body should stay intact; use `document-mode: true` when downstream batching relies on document boundaries.
- **Non-JSON payloads**: set `payload-mode: raw` and parse later (for example `xml` for RSS/Atom). For JSON arrays, prefer `events-field` instead of post-processing with regexes.

For templating patterns, see Variable Expansion and Context Management.

Logsign

Write to Logsign

Source: integrations/logsign.md

LyftData supports writing to Logsign

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to write data to Logsign

Add LyftData output **http-post** or **file** to a job and configure:

Microsoft Azure

Read from and write to Microsoft Azure

Source: integrations/ms-azure.md

LyftData supports reading from and writing to Azure Blob Storage.

Configure LyftData to read from Azure Blob Storage

Add the **azure-blob** input to a job. Key fields (job-spec names in kebab-case):

- `container-name` – blob container to read from (required).
- `blob-names` – list of blob names or prefixes. Leave empty to target the entire container when the selected `mode` allows listing.
- `mode` – choose `list-objects`, `download-objects`, or `list-and-download-objects`.
- `storage-account` / `storage-master-key` – required credentials. These accept literals, `{{ }}` context placeholders, or `${ }` runtime expansions (prefer `${secret|scope/name}` / `${dyn|NAME}` for secrets).
- `ignore-linebreaks` – surface each blob as a single event instead of newline-delimited events.
- `timestamp-mode` – derive timestamps from last-modified metadata or a pattern in the blob name.
- `include-regex` / `exclude-regex` / `maximum-age` – filter candidates by pattern or by age (durations such as `8h`, `2d`).
- `fingerprinting` / `maximum-fingerprint-age` – enable dedupe and tune fingerprint retention.
- `preprocessors` – apply gzip/parquet/base64/extension handlers before events enter the pipeline.

Example: list and download CSV exports

```
input:
azure-blob: container-name: reporting blob-names:


  - exports/daily/


mode: list-and-download-objects include-regex:


  - '\\.csv(\\.gz)?$'


maximum-age: 4h fingerprinting: true storage-account: "${secret|azure/storage_account}"
storage-master-key: "${secret|azure/storage_master_key}" preprocessors:


  - extension
```

Configure LyftData to write to Azure Blob Storage

Add the **azure-blob** output to a job. Key fields:

- `container-name` – destination container (required).
- `blob-destination` – literal name (`name: ...`) or field reference (`field: ...`).
- `mode` – `put` uploads blobs; `delete` removes existing blobs.
- `disable-blob-name-guid`, `guid-prefix`, `guid-suffix` – control collision-avoidance GUID naming for writes. By default a GUID is appended as `/<uuid>`; keep the GUID enabled and set prefix/suffix if you want file-like blob names.
- `input-field` – select the event field to upload; omit to serialize the entire event after preprocessors.
- `content-type` – override the default `text/plain` content type.
- `batch` & `retry` – configure batching and failure handling.
- `track-schema` – keep `__SCHEMA_NUMBER` in sync when writing JSON payloads.
- `preprocessors` – gzip/base64/extension handlers executed before upload.
- `storage-account` / `storage-master-key` – credentials for writes.

See File Handling for details on GUID naming, batching, and `${}` expansions.

Example: upload transformed data to Azure

```
output:
azure-blob: container-name: processed blob-destination: name:
exports/${partition||unknown}/summary guid-prefix: "-" guid-suffix: ".json.gz" input-field:
payload preprocessors:



  • gzip



track-schema: true storage-account: "${secret|azure/storage_account}" storage-master-key:
"${secret|azure/storage_master_key}"
```

Example: delete source blobs after successful processing

```
output:
azure-blob: container-name: reporting blob-destination: field: blob_name mode: delete
storage-account: "${secret|azure/storage_account}" storage-master-key:
"${secret|azure/storage_master_key}"
```

Delete operations expect the incoming event to include the blob name (for example from the Azure input). GUID prefixes are not applied when deleting.

Microsoft Sentinel

Read from and write to Microsoft Sentinel

Source: integrations/ms-sentinel.md

LyftData supports reading from and writing to Microsoft Sentinel

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to read data from Microsoft Sentinel

Add LyftData input **azure-blob** or **s3** to a job and configure:

Configure LyftData to write data to Microsoft Sentinel

Add LyftData output **azure-blob**, **gcs**, **s3** or **http-post** to a job and configure:

Microsoft Windows

Read from Microsoft Windows

Source: integrations/ms-windows.md

LyftData supports reading from Microsoft Windows

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to read data from Microsoft Windows

Add LyftData input **windows-event-log** to a job and configure:

S3 Object Storage (AWS, Minio, Wasabi, Linode, Etc.)

Read from and write to S3

Source: integrations/s3.md

S3

Use this guide to configure LyftData jobs that read from or write to S3-compatible object storage (AWS S3, MinIO, Wasabi, DigitalOcean Spaces, etc.). Bring credentials with permission to list, read, and write the target buckets before you start.

Configure LyftData to read from S3

Add the **s3** input to a job. Key fields (names shown exactly as they appear in the job spec):

- `bucket-name` – target bucket (required).
- `object-names` – object keys or prefixes. Leave empty to scan the entire bucket when the chosen `mode` allows listing.
- `region` / `endpoint` – optional region or custom endpoint (for MinIO and other providers).
- `mode` – switch between `list-objects`, `download-objects`, and `list-and-download-objects` depending on whether you only need metadata, specific keys, or listing + download.
- `ignore-linebreaks` – treat the entire object as a single event instead of splitting on newlines. Combine with preprocessors if you need to parse structured data afterwards.
- `timestamp-mode` – derive timestamps from last-modified, object-name patterns, or disable timestamp filtering.
- `include-regex` / `exclude-regex` / `maximum-age` – reduce the candidate list by pattern or age (accepts durations like `15m`, `6h30m`).
- `fingerprinting` / `maximum-fingerprint-age` – avoid reprocessing objects and control dedupe retention.
- `access-key`, `secret-key`, `security-token`, `session-token`, `role-arn` – authentication. Provide the combination required by your platform. Values can be literal strings, `{{ }}` context placeholders, or `${ }` runtime expansions (prefer `${secret|scope/name}` / `${dyn|NAME}` for secrets).
- `preprocessors` – gzip/parquet/base64/extension handlers applied after download.

Example: list and download Parquet objects

```
input:
s3: bucket-name: analytics-prod object-names:


  • exports/daily/


mode: list-and-download-objects include-regex:
```

```
    - '\\.parquet(\\.gz)?$'


maximum-age: 12h fingerprinting: true timestamp-mode: last-modified access-key:
"${secret|s3/access_key}" secret-key: "${secret|s3/secret_key}" preprocessors:


    - parquet
```

This configuration lists objects beneath `exports/daily/`, filters to recent Parquet files, downloads each object once, and converts rows into JSON events.

Example: download a specific object as a single event

```
input:
s3: bucket-name: analytics-prod object-names:


    - raw/snapshot.json


mode: download-objects ignore-linebreaks: true access-key: "${secret|s3/access_key}" secret-
key: "${secret|s3/secret_key}" preprocessors:


    - extension
```

Configure LyftData to write to S3

Add the **s3** output to a job. Key fields:

- `bucket-name` – destination bucket (required).
- `object-name` – literal or field-derived destination. Use `name: ...` for a fixed key or `field: ...` to reuse an event field.
- `mode` – `put` (default) uploads objects; `delete` removes objects by key.
- `disable-object-name-guid`, `guid-prefix`, `guid-suffix` – control collision-avoidance GUID naming for writes. By default a GUID is appended as `/<uuid>`; keep the GUID enabled and set prefix/suffix if you want file-like keys.
- `input-field` – source field to upload; omit to serialize the entire event after preprocessors.
- `batch` & `retry` – optimize throughput and resiliency.
- `track-schema` – maintain `__SCHEMA_NUMBER` when writing JSON payloads.
- `access-key`, `secret-key`, `security-token`, `session-token`, `role-arn` – same authentication options as the input.
- `preprocessors` – gzip/base64/extension handlers executed before upload.

See File Handling for details on GUID naming, batching, and `${}` expansions.

Example: upload processed events with dynamic keys

```
output:
s3: bucket-name: analytics-prod-archive object-name: name:
processed/${partition||unknown}/summary guid-prefix: "-" guid-suffix: ".json.gz" input-field:
payload preprocessors:



  • gzip



track-schema: true access-key: "${secret|s3/writer_access_key}" secret-key:
"${secret|s3/writer_secret_key}"
```

Example: delete the original object after processing

```
output:
s3: bucket-name: analytics-prod object-name: field: object_name mode: delete access-key:
"${secret|s3/writer_access_key}" secret-key: "${secret|s3/writer_secret_key}"
```

The delete output expects each incoming event to include `object_name` (for example from the S3 input). Delete operations do not generate GUID prefixes.

Recommendations for buckets and folders

- Keep individual objects below 100–150 MB (compressed) for predictable processing latency.
- Organise exports using partition-style prefixes such as `Y/M/`, `Y/M/D/`, or `Y/M/D/H/`.

Creating an AWS policy for programmatic access

Grant the job permissions to list and read/write objects. Replace `BUCKETNAME` with your bucket name:

```
{
"Version": "2012-10-17", "Statement": [ { "Sid": "List", "Effect": "Allow", "Action":
"s3:ListBucket", "Resource": "arn:aws:s3:::BUCKETNAME" }, { "Sid": "RW", "Effect": "Allow",
"Action": ["s3:GetObject", "s3:PutObject", "s3:DeleteObject"], "Resource":
"arn:aws:s3:::BUCKETNAME/*" } ] }
```

Scuba Analytics

Read from and write to Scuba Analytics

Source: integrations/scuba.md

LyftData supports reading from and writing to Scuba Analytics

> note

The LyftData UI is designed to be self explaining and already documents all possible configuration parameters. All published features are available while we are working on this web documentation. Please ask us for direct support

> Note

Configure LyftData to read data from Scuba Analytics

Add LyftData input **s3**, **azure-blob**, **gcs** or **http-poll** to a job and configure:

Configure LyftData to write data to Scuba Analytics

Add LyftData output **s3**, **azure-blob**, **gcs** or **http-post** to a job and configure:

Twilio Segment

Draft pattern guide for ingesting Twilio Segment exports and API feeds

Source: integrations/segment.md

LyftData supports ingesting data exported by Twilio Segment via object storage (S3, Azure Blob, GCS) or the Segment HTTP delivery API.

Configure LyftData to read Segment exports from Amazon S3

Segment warehouses frequently mirror data into S3. Configure the **s3** input:

- `bucket-name` – Segment export bucket.
- `object-names` – prefix where Segment writes files (for example `segment/events/`).
- `mode` – `list-and-download-objects` to stream files as they arrive.
- `include-regex` – match `.json.gz` or `.csv.gz` depending on export format.
- `preprocessors` – include `extension` (or `gzip`) so gzip-compressed files are decompressed.
- `fingerprinting` – avoid reprocessing already ingested files.

Example: ingest Segment JSON exports from S3

```
input:
s3: bucket-name: segment-prod object-names:


  • exports/


mode: list-and-download-objects include-regex:


  • '\\.json(\\.gz)?$'


fingerprinting: true timestamp-mode: last-modified access-key:
"${secret|segment/s3_access_key}" secret-key: "${secret|segment/s3_secret_key}"
preprocessors:


  • extension
```

Configure LyftData to read Segment exports from Azure Blob or GCS

The same pattern applies to **azure-blob** and **gcs** inputs. Substitute the appropriate credentials (`storage-account` / `storage-master-key` or `GcsCredentials`) and field names (`container-name`, `blob-destination`, etc.) following the examples in the Azure and GCS guides. Use `include-regex` to capture `.json.gz` exports and enable `fingerprinting` to dedupe.

Configure LyftData to receive Segment events via HTTP

For real-time delivery, Segment can send events to an HTTP endpoint. Configure the **http-poll** input when polling Segment APIs, or use **http-server** when Segment posts events to LyftData.

Example: poll the Segment delivery API

```
input:
http-poll: url: https://api.segmentapis.com/export/v1/jobs/{job_id}/download method: get
headers: Authorization: "Bearer ${secret|segment/token}" json: true trigger: interval:
duration: 15m retry: timeout: 30s retries: 5
```

Tune the URL, authentication headers, and triggers based on the Segment API being used (Batch/Delivery). Enable `json: true` to parse Segment payloads automatically.

Splunk Cloud, Splunk Enterprise

Write to Splunk HEC

`Source: integrations/splunk.md`

Splunk HEC output

LyftData supports Splunk Http Event Collector (HEC) to send data to Splunk.

Configure LyftData to send data to Splunk HEC

Add LyftData output **splunk-hec** to a job and configure:

- **URL**: Splunk HEC endpoint, example: https://127.0.0.1:8088/services/collector/event
- **HEC token**: authentication token
- **Splunk fields**: (index, host, source, sourcetype). Edit the value or the fieldname in the data, where the value can be found.
- **Batch**: Batching input events together
- **Retry**: Retry behaviour
- **Insecure**: Ignore TLS certificate validation errors (This is unsafe to use)
- **Metrics**: Send a metrics formatted payload to the HEC endpoint
- **Remove**: Consume (remove) fields from the event payload before submitting to the endpoint. Applicable to time-field, host-field, source-field, sourcetype-field, index-field and hec-token-field
- **Event Field**: If specified the field's contents will be submitted as the event payload to the endpoint
- **Time Field**: Use the specified field for the timestamp of the endpoint, should be in Unix epoch format