Contents

Structured Logging For Cloud Collectors **1**
1. Overview . . . 1
Key features . . . 2
Compatibility . . . 2
2. Installation . . . 3
Via Composer . . . 3
Verify . . . 3
3. Configuration . . . 3
3.1 General . . . 4
3.2 Custom Format . . . 4
3.3 Service Fields . . . 5
3.4 Locking values per environment . . . 6
4. Formats . . . 7
Exceptions . . . 8
Request correlation . . . 8
5. Which logs are covered . . . 9
6. Checking coverage . . . 9
7. Use cases . . . 10
7.1 A store on AWS ECS, shipping to CloudWatch . . . 10
7.2 A store on Google Cloud Run . . . 10
7.3 An Elasticsearch or OpenSearch cluster . . . 10
7.4 A plain server that must keep its files . . . 11
8. Troubleshooting . . . 11
9. Support and changelog . . . 11

Structured Logging For Cloud Collectors

User Guide
Version: 1.1.0
Compatibility: Adobe Commerce and Magento Open Source 2.4.8 – 2.4.9, PHP 8.2 – 8.5
Vendor: Webmaster Ramos

---

1. Overview

Adobe Commerce and Magento Open Source write their logs as free text into a handful of files under `var/log`. On a server that is fine. In a container it is wrong twice over: the files disappear when the container is replaced, and the lines carry no fields a log platform can index, so searching them means reading them.

This extension changes where those records go and what shape they arrive in. It does not add logging of its own and it does not ask any other extension to change a single call: it replaces the destination of the logs the platform already writes. Disabled, which is how it ships, nothing changes at all. Enabled, every record leaves as one structured line on standard output, on standard output and standard error split by severity, or into the same files as before - in whichever of eleven formats the collector in front of the store already understands.

Whether the collector is Elastic, Grafana Loki, Graylog, Datadog, Google Cloud Logging, Splunk, CloudWatch or a plain Docker log driver, the work is picking the format it parses natively and turning the extension on. Nothing is shipped anywhere by the extension itself; the collector already running beside the store does the shipping.

The whole product in one picture - the same logging call, before and after:

Figure 1: The same record: a line of text in a file, then a structured line on stdout

Key features

- Eleven output formats, including several that need no ingest pipeline
- Three destinations: stdout + stderr, stdout only, or the usual `var/log` files
- Disabled by default, and completely invisible while disabled
- Covers the main log and six platform channels, one line per record
- A request id on every line, so a single request can be followed end to end
- Exceptions written as typed error objects rather than flattened strings
- Service, environment and version fields, lockable per environment
- Scanner-induced errors recorded as warnings so alerting stays quiet
- A console command that reports which logs are covered and which are not
- No database tables, no external services, no data leaving the store

Compatibility

Platform	Versions
Adobe Commerce	2.4.8, 2.4.9
Magento Open Source	2.4.8, 2.4.9
PHP	8.2, 8.3, 8.4, 8.5
Themes	Admin only - no storefront output, theme-independent
Database	No tables added
External dependencies	None

The extension is written against Monolog 3, which the platform ships from 2.4.8 onwards. On 2.4.7 and earlier, which ship Monolog 2, the package declines to install rather than failing later.

---

2. Installation

Via Composer

```
composer require webmaster-ramos/module-structured-logging
bin/magento module:enable WebRamos_StructuredLogging
bin/magento setup:upgrade
bin/magento setup:di:compile
bin/magento cache:flush
```

`setup:di:compile` is required in production mode and harmless elsewhere. The extension adds no database table, so `setup:upgrade` only registers it.

Verify

```
bin/magento module:status WebRamos_StructuredLogging
```

Nothing about the store's behaviour changes at this point. The extension is disabled by default and stays invisible until it is turned on.

---

3. Configuration

Navigate to **Stores → Configuration → Webmaster Ramos → Structured Logging**.

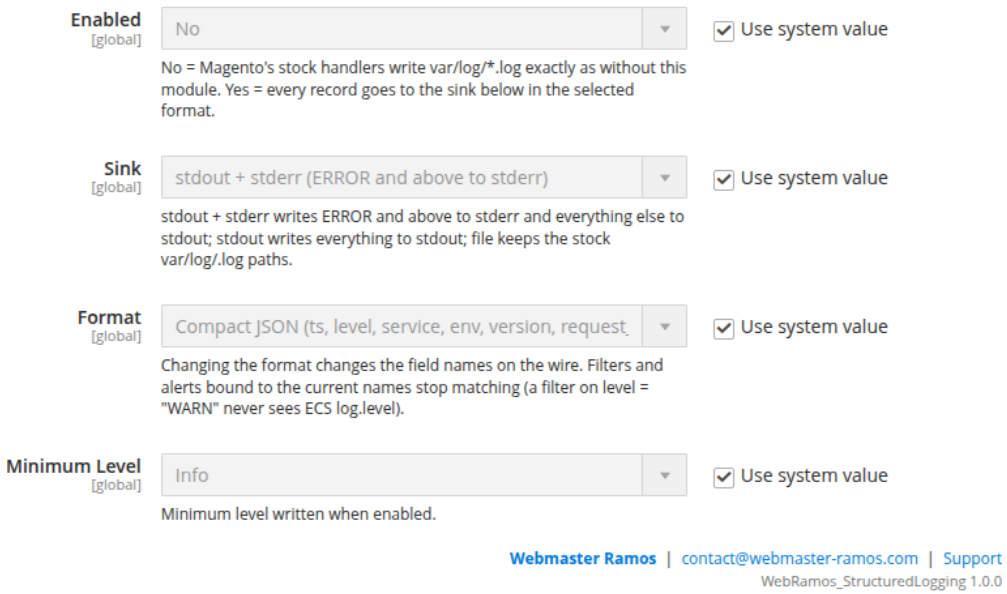

Figure 2: The General group: enable, destination, format and severity threshold

Every setting is global. Log routing is a property of the installation, not of a store view, so the section is available at default scope only.

3.1 General

Field	Description	Default
Enabled	No leaves every record with the platform's own handlers and files. Yes sends each record to the sink below, in the format below.	No
Sink	Where records go when enabled.	stdout + stderr
Format	The shape of each line. See section 4.	Compact JSON
Minimum Level	Records below this severity are not written.	Info

The three sinks:

Sink	Behaviour
stdout + stderr	Errors and above go to standard error, everything else to standard output. This is what most container platforms expect, and it lets an operator separate the two streams.
stdout only	Everything to standard output. Choose this when the collector reads one stream.
var/log files	The same structured lines, written to the same files as before (`var/log/system.log`, `payment.log`, and so on). Choose this on a plain server with a file-tailing agent.

3.2 Custom Format

These two fields apply only when **Format** is set to *Custom line template*; they stay hidden otherwise.

Field	Description	Default
Line Template	The line to write, with placeholders.	`[%datetime%] %channel%.%level_name%: %message% %context% %extra%`
Date Format	PHP date format used for `%datetime%`.	`Y-m-d\TH:i:s.vP`

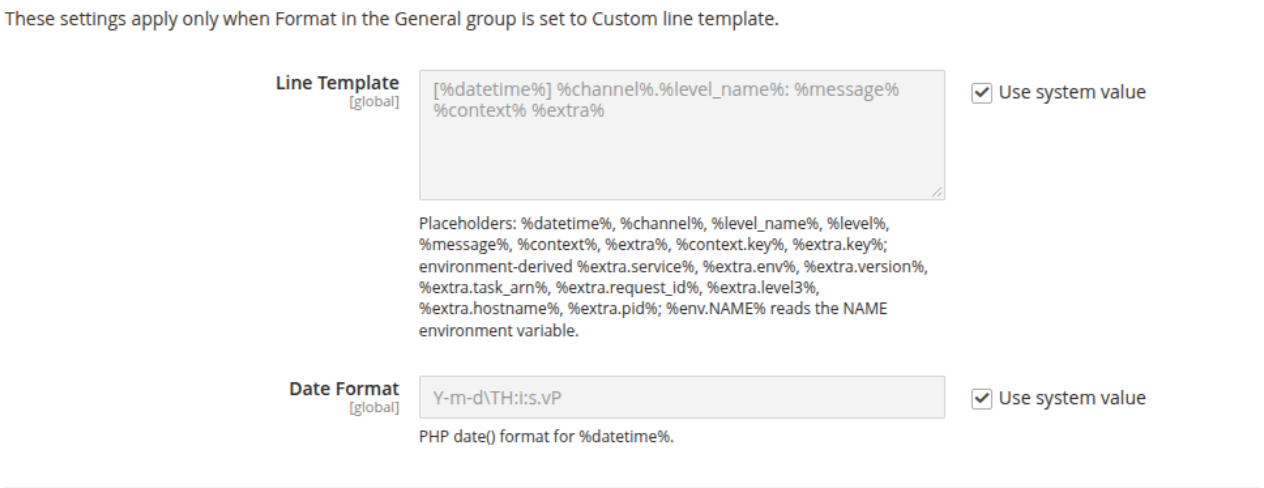

Figure 3: The custom template fields, shown only when the format is Custom line template

Available placeholders:

Placeholder	Meaning
`%datetime%` `%channel%` `%level_name%` `%level%` `%message%`	The record itself
`%context%` `%extra%`	The whole context / extra array
`%context.key%` `%extra.key%`	One entry of either
`%extra.service%` `%extra.env%` `%extra.version%`	The Service Fields below
`%extra.request_id%` `%extra.level3%`	Request id; severity as INFO / WARN / ERROR
`%extra.hostname%` `%extra.pid%` `%extra.task_arn%`	Host, process id, AWS ECS task
`%env.NAME%`	The value of the `NAME` environment variable

The custom template cannot escape values. When a message may contain a space or a quote, prefer the **logfmt** format over a hand-built `key=value` template.

3.3 Service Fields

Field	Description	Default
Service	The service name written into JSON formats.	empty
Environment	prod, staging, dev, and so on.	empty
Version	Release identifier: a git tag or a short commit id.	empty

These three appear in every JSON format, and they are what makes a log platform able to separate one environment from another.

Each field may name an environment variable instead of a value: `%env.NAME%`, the same placeholder as in the custom template. The extension replaces it with the `NAME` variable when it writes a record - from the process environment, then the FastCGI parameters, otherwise empty - so `%env.APP_VERSION%` reports whatever release is running without the stored value ever changing. Text and placeholders can be mixed, as in `build-%env.BUILD_ID%`.

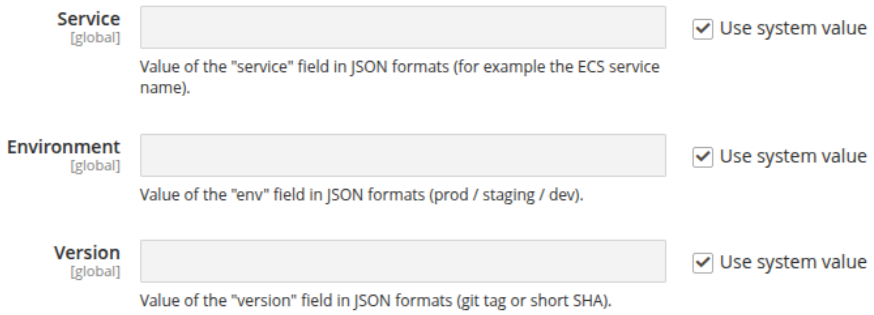

Figure 4: Service Fields: the values written into every JSON line

3.4 Locking values per environment

A deployment usually wants production to be structured and a developer's machine to stay on the familiar files. Put the values into the `system` section of `app/etc/env.php`:

```
'system' => ['default' => ['webramos_structured_logging' => [
    'general' => [
        'enabled' => '1',
        'sink'    => 'stdout',
        'format'  => 'gcp',
        'level'   => 'info',
    ],
    'fields' => [
        'service' => 'magento-web',
        'env'     => 'prod',
        'version' => '%env.APP_VERSION%',
    ],
]]],
```

Written this way - by hand, by `bin/magento config:set` with its lock option, or by whatever generates the deployment file - the values beat the database, the running application uses them, and the administration panel renders those fields read-only so nobody can change them from the browser by accident.

This also matters before the database is reachable: the extension reads these values during start-up, when the configuration table cannot be queried yet, so the earliest records of a process are already in the right shape.

> **Keep this block the same on every release.** Magento hashes the whole `system` section of `app/etc/env.php` and `app/etc/config.php`, stores the hash when `bin/magento app:config:import` or `setup:upgrade` runs, and answers every storefront and admin

page with "The configuration file has changed" while a server's files hash differently. A value that changes per release or per server - a commit id, a build number, a task id - written literally into the block therefore breaks rolling deployments and rollbacks. Write `%env.NAME%` instead, as in the example, and give each process the variable. Under PHP-FPM the workers must see it: keep the pool's `clear_env = no`, or pass it with `env[NAME]` or `fastcgi_param`.

The placeholder reads the process environment and the FastCGI parameters, which under PHP-FPM include the request headers (`HTTP_*`). Never point it at a secret or at a request header.

Environment placeholders (`CONFIG__DEFAULT__…`, `MAGENTO_DC_SYSTEM__…`) are not a lock. They grey the field in the administration panel and appear in `bin/magento config:show`, but the running application never reads them. Use the `env.php` block above.

---

4. Formats

Format	Choose it for	A line looks like
Compact JSON	A general JSON platform: CloudWatch Logs Insights, Splunk, anything with a JSON parser	`{"ts":"2026-08-28T19:25:34.175Z","level":"INFO","service":"magento-web","env":"prod","version":"9f3c1ab","request_id":"5455c044…","msg":"Order placed"}`
Elastic Common Schema	Elasticsearch or OpenSearch - ECS 8.x field names, no ingest pipeline needed	`{"@timestamp":"…","log":{"level":"info","logger":"main"},"message":"Order placed","ecs":{"version":"8.11"},"service":{"name":"magento-web"}}`
Logstash	An existing Logstash pipeline	`{"@timestamp":"…","@version":1,"host":"…","message":"Order placed","type":"magento-web","channel":"main","level":"INFO"}`
Syslog RFC 5424	rsyslog, syslog-ng, or a syslog input	`<14>1 2026-08-28T19:15:01+00:00 host magento-web 1896 main - INFO: Order placed`
Google Cloud Logging	Cloud Run, GKE or GCE - parsed natively from stdout, and exceptions reach Error Reporting	`{"severity":"INFO","time":"…","message":"Order placed","serviceContext":{"service":"magento-web","version":"9f3c1ab"}}`

Format	Choose it for	A line looks like
Datadog	Datadog - reserved attributes and unified service tagging, so no remapping	`{"timestamp":"…","status":"info","message":"Order placed","service":"magento-web","env":"prod","host":"…"}`
GELF 1.1	Graylog, through a GELF input or a GELF output on the collector	`{"version":"1.1","host":"…","short_message":"Order placed","timestamp":1787944501.034,"level":6,"_service":"magento-web"}`
logfmt	Grafana Loki - its logfmt parser and level detection work out of the box	`ts=… level=info msg="Order placed" service=magento-web env=prod request_id=5455c044…`
Monolog JSON	A pipeline already written against Monolog's own JSON shape	`{"message":"Order placed","context":{},"level":200,"level_name":"INFO","channel":"main","datetime":"…"}`
Magento line	Keeping the familiar text while changing only the destination	`[2026-08-28T19:15:01+00:00] main.INFO: Order placed [] []`
Custom line template	Anything else - see section 3.2	whatever the template says

Changing the format changes the field names on the wire. Filters, dashboards and alerts bound to the old names stop matching - a filter on `level = "WARN"` never sees Elastic Common Schema's `log.level`. Change the format and the alerts together.

Exceptions

When a record carries an exception, it is written as a typed object rather than folded into the message: class, code, message, and a stack trace bounded at 8 KB so one failure cannot flood a log budget. Compact JSON writes it under `err`, Elastic Common Schema under `error`, and the Google Cloud, Datadog and GELF formats use each platform's own reserved error fields.

Request correlation

Every JSON line carries a `request_id`. It is taken from the `X-Request-Id` header when the load balancer or reverse proxy in front of the store sets one, and generated once per process otherwise - which covers command-line and cron work as well as web requests. One customer's failing checkout is therefore one query, not a timestamp search.

`task_arn` is added only when the process runs inside an AWS ECS task. On any other host the field is left out rather than filled with a placeholder.

---

5. Which logs are covered

The extension covers the main application log and six platform channels: payment, shipping, cron, Admin Adobe IMS, Service Proxy and Payment Services. A record that several handlers would have written - the main log plus a channel file, say - is emitted once, not twice.

Extensions that log through the platform's standard logger are covered automatically, with no change on their side. An extension that owns its own log channel and writes its own file keeps doing so; the command in section 6 is how those are found.

---

6. Checking coverage

```
bin/magento webramos:structured-logging:inspect
```

The report lists every logger the installation has configured and, for each one, where its records go. Two flags matter:

- **ESCAPE** - this log still writes its own file even when the extension is enabled. Almost always a third-party extension with its own channel.
- **DUPLICATE** - one record would produce two lines on the destination.

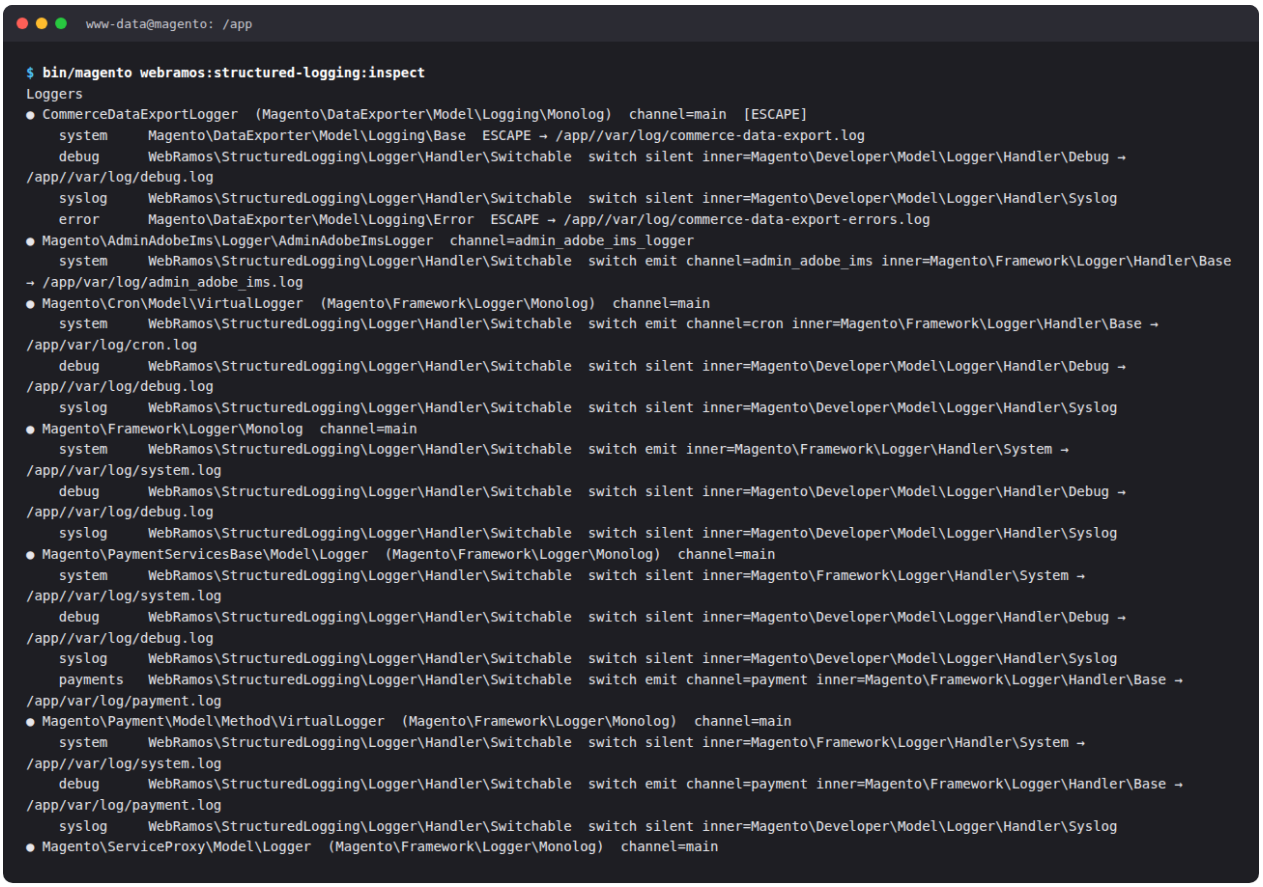

Figure 5: The coverage report

Options:

Option	Effect
`--strict`	Exit with a failure code when anything is flagged. Useful in a deployment check.
`--probe`	Write one marker record through every logger and measure `var/log` around it - proof by measurement rather than by configuration reading. Refused while the extension is disabled, and in production mode without `--force`.
`--probe --observe --seconds=N`	Measure real traffic for N seconds and write nothing at all. Safe anywhere, including production.
`--force`	Allow the active probe in production mode.

The report also lists the three service fields: where each value comes from (**lock** - the `system` section of `env.php` or `config.php` - the **database**, or the **default**), the stored value and, for `%env.NAME%`, what it resolves to in the command's own process and any variable that is not set. A plain value in the lock is marked, because a value that changes between releases does not belong there. PHP-FPM workers resolve the placeholders from their own environment, which can differ from the shell's.

The command never changes configuration, never clears a cache and never touches generated code. Turning the extension on to measure it would change the behaviour of every process on the instance, so it declines to do that on its own.

---

7. Use cases

7.1 A store on AWS ECS, shipping to CloudWatch

Set **Enabled** to Yes, **Sink** to stdout + stderr, **Format** to Compact JSON, and fill the three Service Fields. The container's log driver already forwards both streams, so the next deployment produces queryable JSON with a service, an environment, a version, a request id and, inside ECS, a task identifier - with no agent to install and no pipeline to write.

7.2 A store on Google Cloud Run

Same, with **Format** set to Google Cloud Logging and the fields locked in `env.php` as `%env.K_SERVICE%` and `%env.K_REVISION%`, the platform's own environment variables. Cloud Logging parses the lines natively, and a record carrying an exception also reaches Error Reporting.

7.3 An Elasticsearch or OpenSearch cluster

Set **Format** to Elastic Common Schema. The field names are the ones the index template already expects, so no ingest pipeline is needed to rename anything.

7.4 A plain server that must keep its files

Set **Sink** to var/log files and pick a JSON format. The files stay exactly where the operations team expects them, with logrotate and any file-tailing agent unchanged - only the contents of each line become structured.

---

8. Troubleshooting

Nothing appears on standard output. Check that **Enabled** is Yes and that the value is not locked to something else in `app/etc/env.php` - a locked value wins over the administration panel, which is the point of locking it. Then check **Minimum Level**: at Info, debug records are not written.

Records still appear in `var/log`. Run the command in section 6. A log flagged ESCAPE belongs to another extension that writes its own file; it is not covered and the report names it.

One record produces two lines. The same command flags this as DUPLICATE.

The format select is missing an option. The list is built at runtime, so an extension that adds a format contributes to it directly. If an expected option is absent, that extension is not enabled.

A configuration field is greyed out. Its value is locked in `app/etc/env.php`. Change it there, or remove the lock.

Every page answers "The configuration file has changed" after a deployment. A value under `system` in `app/etc/env.php` or `app/etc/config.php` differs from what Magento last imported - typically a release id written literally into the service fields. Replace it with `%env.NAME%` (section 3.4) and run `bin/magento app:config:import` once.

Log volume is higher than expected. Raise **Minimum Level** to Warning. Debug and info detail belongs on a developer's machine, not in a production log budget.

---

9. Support and changelog

- **Author:** Webmaster Ramos
- **Email:** contact@webmaster-ramos.com
- **Website:** https://webmaster-ramos.com
- **Support:** https://webmaster-ramos.com/support
- **Changelog:** see `CHANGELOG.md` inside the package