GUIDE

chronosphere

Log Data Processing Guide: Reduce Costs Without Losing Visibility

A practical, how-to guide for observability teams

The challenge you're facing

Your log data is growing exponentially – 250% year-over-year, on average – but only a fraction provides value to monitoring and troubleshooting. You know you need to reduce costs, but you're worried about creating blind spots that could impact incident response or compliance requirements.

250%

year-over-year log growth (on average)

The good news: You can significantly reduce log data costs while maintaining the visibility you need. The key is applying the right processing rules to the right data types.

USE THIS GUIDE to understand how to pay for the logs you need – in the right format — while improving your ability to troubleshoot.

Before you start: understand your log usage

To effectively reduce your log data, you need to understand how your team is using your logs. Does this dataset populate dashboards? Is it referenced in alerts? Or does your team search it while troubleshooting?

Different processing approaches work better for different use cases. Random sampling might break your monitoring practices, while dropping error logs entirely could eliminate crucial troubleshooting context.

Quick decision framework

DATA TYPE	EXAMPLE	PROCESSING RULE
High-volume data with low individual value	• *Successful API calls* • *Routine operations*	• *Sample log* • *Convert to metric*
Non-critical environments or log levels	• *DEBUG logs* • *Staging environments*	• *Sample log* • *Drop log*
Essential data with size issues	• *Verbose ERROR logs*	• *Drop field* • *Remove whitespace and unneeded punctuation*
Any data you reduce or optimize	• *The raw data a metric is derived from*	• *Route to object storage*

Processing rules reference

Convert to metric

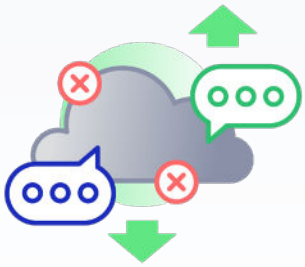
What it does:
Transforms raw log data into time-series metrics for monitoring and alerting.

When to use:
You want to monitor behaviors over time (request counts, latency percentiles, rates, etc.). Here, you can sacrifice individual log details for aggregate insights.

Expected outcome:
Significant cost savings vs. raw log storage, but only when implemented upstream (before ingestion).

EXAMPLE

Calculate the rate of ERROR messages created by a service.



Drop log

What it does:
Completely discards specified log types.

When to use:
Non-critical datasets that you don't need real-time visibility into.

Expected outcome:
Maximum cost savings for targeted data, but zero visibility into dropped data. (Use cautiously.)

EXAMPLE

Drop all DEBUG logs from staging environments, or drop verbose health check logs.

Sample log

What it does:
Keeps a specified percentage of logs while discarding the rest.

When to use:
High-volume, repetitive logs where individual entries have low value. In this situation, you would need to maintain "success patterns" for troubleshooting.

Expected outcome:
Substantial volume reduction while preserving representative samples for pattern analysis.

EXAMPLE

Keep 1% of successful HTTP 200 status code logs

(500 out of 50,000 per minute).

Drop field

What it does:
Removes unnecessary or redundant fields from log events while keeping the core log.

When to use:
When removing duplicate fields, null fields, or unneeded information.

Expected outcome:
Modest per-log savings that accumulate significantly at scale. Maintains most of the log context.

EXAMPLE

Remove redundant environment fields or empty "user_id" fields.

Remove whitespace or punctuation

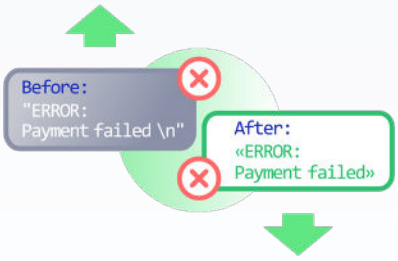
What it does:
Strips extra spacing, line breaks, and unnecessary punctuation from log content.

When to use:
Logs have excessive formatting, punctuation, or whitespace.

Expected outcome:
Small per-log size reduction with meaningful cumulative savings across millions of logs.

EXAMPLE

Transform "ERROR: Payment failed \n" to "ERROR: Payment failed"



Route to object storage

What it does:
Sends raw log data to cheaper long-term storage (Amazon S3, Google Cloud Storage, Azure Blob) instead of your primary logging platform.

When to use:
Compliance requires long-term log retention and you want to apply processing rules but preserve original data.

Expected outcome:
Dramatic reduction in long-term retention costs while maintaining compliance posture and enabling data rehydration when needed.

EXAMPLE

Route 100% of raw data to S3 while keeping processed logs in your observability platform.

Pro tips for success

1. **Start with non-critical data**
   Begin with staging environments or DEBUG logs to gain quick wins.

2. **Layer your approach**
   Combine multiple rules – convert high-value logs to metrics, sample the remainder, and route all raw data to object storage.

3. **Monitor the impact**
   Track both cost reduction and any operational blind spots you've created.

4. **Keep compliance in mind**
   Route original data to object storage before applying destructive processing rules.

5. **Document your decisions**
   Maintain a record of what processing rules you've applied and why, for future context.

Remember:
The goal isn't to eliminate all log data. It is to pay only for the signal, not the noise.