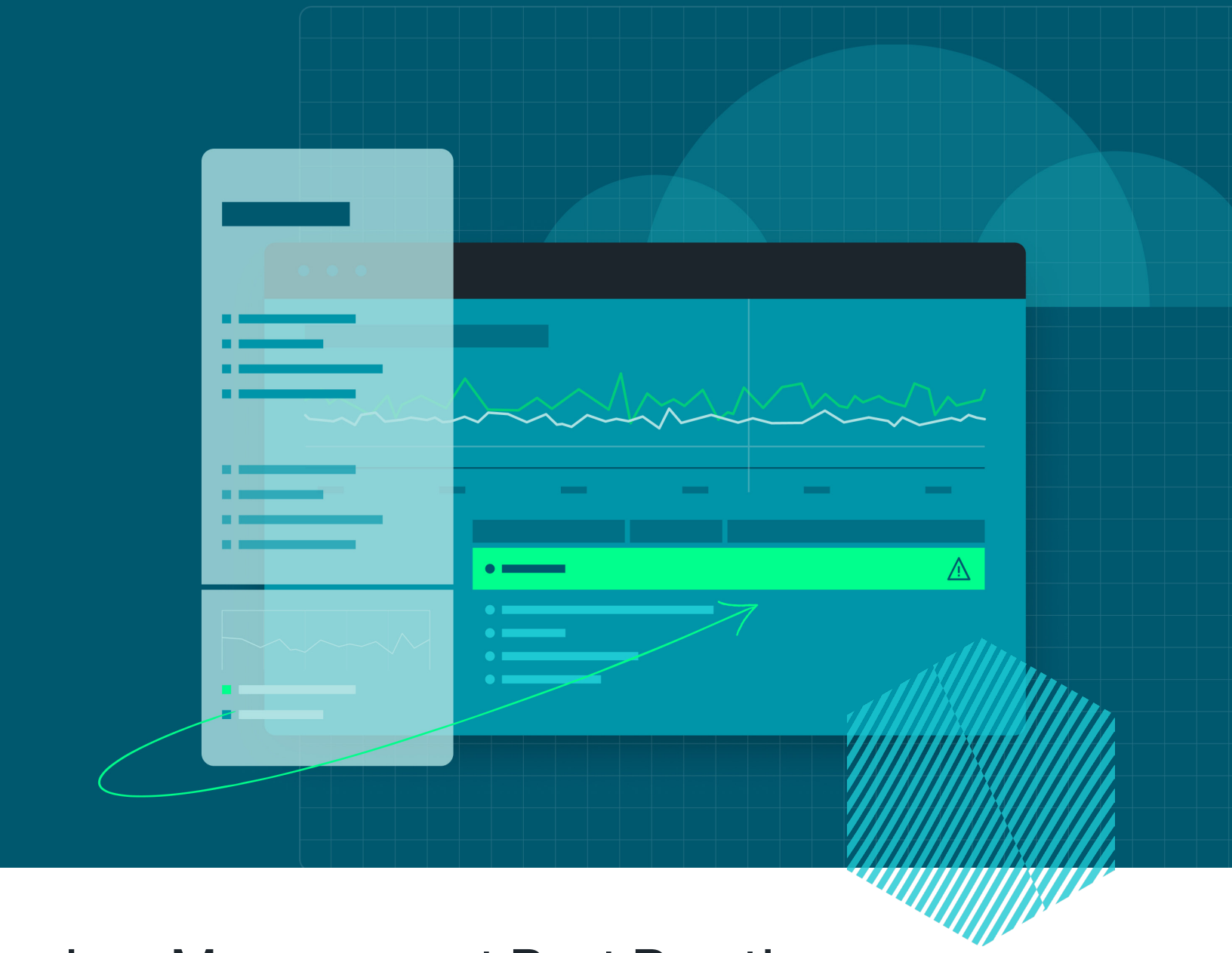
new relic



Log Management Best Practices

Shift to effective logging for full-stack observability

White paper / August 2022

Contents

Introduction

Log management has evolved. Organizations have moved beyond sifting through raw dumps of application and infrastructure logs whenever something breaks. Logging now plays a crucial role in an organization's operations, business intelligence, and marketing. Logs power observability. Well-structured logs are a superpower that enable organizations to understand quickly and easily how their whole system runs—and even preempt issues.

Using logs for effective observability requires more thought and care than simply dumping massive quantities of poorly formatted logs into a database or file. How can organizations intelligently change their logging practices so that detailed logs can improve their ability to correlate incidents across applications and infrastructure, in real time, without having to toggle between different applications and tools? How can they better achieve end-to-end observability? How can they come closer to attaining full-stack observability so it can serve their businesses?

It's easy to shift logging practices to ensure that logs enhance full-stack observability. In this white paper, we discuss some logging best practices for modern organizations.

Traditional logging

Traditionally, logging happens in a data silo stored separately from other systems. In the past, observability relied on application performance monitoring (APM) and infrastructure monitoring. While monitoring is important, it doesn't tell the entire story inside the logs of applications and infrastructure devices. Many separate and siloed monitoring and logging tools are application-centric, only focus on one piece of the stack, and cannot provide complete information about what is happening and why.

It's critical that teams have the information they need to accelerate time to market, gain greater insight into customer behavior, and reduce incident response time.

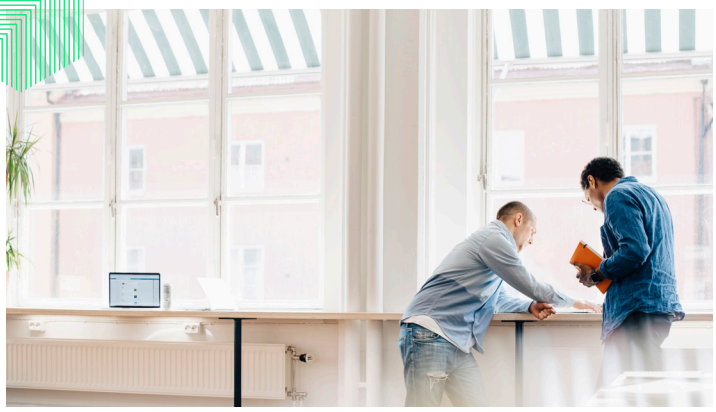
Many organizations either choose not to have granular detail from their logs and struggle to determine the underlying cause of issues, or they use separate tools and try to map log details to errors and traces. Maintaining detailed logs in separate silos prevents teams from seeing the complete picture, increases costs and time to market for products, reduces visibility into the customer experience, and increases the mean time to resolution (MTTR).

Full-stack observability

The ability to see everything in the tech stack that could affect the customer experience is called full-stack observability or end-to-end observability. It is based on a complete view of all telemetry data (metrics, events, logs, and traces).

Full-stack observability provides complete visibility into how complex applications and systems perform—ideally from a single, integrated solution—to troubleshoot incidents, reduce MTTR, and understand the customer experience.

With full-stack observability, engineers and developers don't have to sample data, compromise their visibility into the tech stack, or waste time stitching together siloed data. Instead, they can focus on the higher-priority, business-impacting, and creative coding they love.



Logging for full-stack observability

Generating logs for the entire stack can be overwhelming. Developers and engineers might have questions about what to log, how much detail to include, and whether too much data will lead to high costs. Many organizations pay the high price of centralizing their log management in a different platform and ultimately are forced to limit the log data sent based on performance and price. This data sampling provides limited visibility and value to the business. With this in mind, we look at some log management best practices for full-stack observability.

Log the right things

Logs are generated by writing text to a standard output or a file. The most important decision is selecting what goes into logs. Logs should include all necessary metadata to help pinpoint events and root causes when investigating. Log metadata elements may include error messages or stack traces and related values, metrics, or events.

Everything an organization logs should have a purpose. Whether it's usage data, user events, or application errors and exceptions, it should be valuable to the team. Log data information should:

- Be immediately useful in some way
- Provide the necessary details to understand underlying causes and make decisions

Anticipate common scenarios

Logs aren't just for incident response. Logs can help with other parts of the business, such as performance profiling or gathering statistics.

Logging with some common scenarios in mind ensures the logs provide direct value to the organization. For example, user interaction logs can provide crucial insight into the customer experience. System logs can monitor issues or hardware failures. Detailed application logs may provide insight into performance and potential problems such as memory leaks. All of which can be important in making business decisions.

Log meaningful messages

Log messages are only as valuable as the information and context they provide. By adding enough detail and making them understandable, teams can use the logs effectively. Third-party infrastructure tends to capture the necessary granular details. Still, for applications written in-house, teams should always capture the log details that will enable them to diagnose and determine why an error/event happened so they can take the necessary actions that impact the business.

For application errors, the message should convey what is happening with that line of code. For example, a `Transaction Failed` error message is not as helpful as an error message like `Transaction Failed: Could not create user ${path/to/file:line-number}`. Logs that include data about transactions help developers and engineers see why transactions failed.

Usually, error codes or status codes can indicate the application's type of problem. Rather than just outputting the error code text or number, adding a short description in the log can save other developers or engineers the time and effort of looking it up when troubleshooting.

Logs should provide critical information to the organization. Developers and engineers should avoid logging cryptic or non-descriptive messages that only limited members of the team would understand.

Keep logs simple and concise

While it's essential to include enough information within the log message, the opposite applies as well. Having excessive, unnecessary data in the message can bloat the log storage size and costs, slow the search logs, and distract from the core issue—making it tougher to debug.

Teams should keep logs concise to capture only the most critical information. Logs should include why an error happened while avoiding unnecessary noise.

They should provide information about the root cause of an error without including every little detail about the environment. For example, if an application failed to connect and retrieve data from an internal API, it may be helpful to log any error messages from the API or the network state information of the environment. It's likely unnecessary to include how much memory the application uses or how many applications are running.

Don't forget the time

Teams should include a timestamp for logs. While this may sound obvious, developers and engineers who are used to writing logs to a database that automatically includes the date and time might not think to add a timestamp in their log messages. They should select the most granular level that makes sense and output it inside logs. High-frequency tasks may need to track time down to the millisecond, while low-frequency tasks may only need to track to the minute—or even the day. What's important is not simply the granularity but applying a consistent standard across the organization.

Another potentially obvious and vital note is to synchronize all systems at the same time so an observability platform can use the timestamp to correlate log events with other telemetry data.

[1] (New Relic, Inc., n.d.)

Use a parsable log format

An observability platform that can't extract data from logs isn't very helpful. Teams should use a log format that developers and engineers can parse and keep a consistent log structure so that it's easy to collect and aggregate. For example, New Relic log management makes it easy to define custom log parsing rules,[1] but parsing rules can't work their magic if log data is unintelligible.

A good example of an unparsed log format is a default NGINX access log containing unstructured text. It is useful for searching but not much else. In an unparsed format, teams would need to do a full-text search to answer most questions. Here's an example of a typical line:

```
127.180.71.3 - - [10/May/2022:08:05:32 +0000]
"GET /downloads/product_1 HTTP/1.1" 304 0 "-"
"Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
```

After parsing, the log is organized into attributes, like `response code` and `request URL`. Here's an example of the same log information in a parsable log format:

```
{
  "remote_addr":"93.180.71.3",
  "time":"1586514731",
  "method":"GET",
  "path":"/downloads/product_1",
  "version":"HTTP/1.1",
  "response":"304",
  "bytesSent": 0,
  "user_agent": "Debian APT-HTTP/1.3
  (0.8.16~exp12ubuntu10.21)"
}
```

If the format is entirely custom, setting the log type triggers customer-defined parsing rules.

If an organization has multiple applications that serve a common purpose, teams should focus on standardizing a log format for all the apps. This makes incorporating data into their observability platform easier, even if the team associated with each app wants visibility into different attributes.

An in-depth look at log formats

There are three consistent format categories to how the text is structured with implications for usability once a log aggregation tool collects data. The three format categories are:

- **Structured**—One of the most common structured formats for logging is JSON. Many tools can quickly parse it. It is very flexible and lightweight. Ideally, all logs generated are in a structured format. While JSON helps organize hierarchical data, other examples of structured log data include common formats like comma-separated values (CSV) and tab-separated values (TSV).
- **Common**—A common format isn't structured but is well known, defined, and consistent. The Apache common log format for access logs is an example. The advantage of a common format is that many tools can parse the data out of the box.
- **Custom**—If an application isn't logging in a structured or common format, it is writing logs with a custom format. To recognize the start and end of an individual log line during log forwarding, teams may need to parse. Creating customer-defined parsing rules helps make the data more valuable.

Categorize and group logs

Specifying a data model for logs enables teams to search more effectively. They should define and include attributes whenever possible to categorize and group logs accordingly.

The OpenTelemetry standards for logs by a coalition of industry leaders, including New Relic, cover many elements such as naming conventions and field value definitions.[2] While not every framework natively supports a log formatted to exactly these standards, they can serve as a guideline.

Common attributes that can be useful to have in a log data model include resources, logs in context, and log levels.

Resources
Resources define when and where the logs came from, such as:

- Date and time
- Machine hostname or identifier
- Application or service name

The hostname can be meaningful in logs from classic host-based applications with named environments. A pod or container ID would better organize logs from containerized or orchestrated environments.

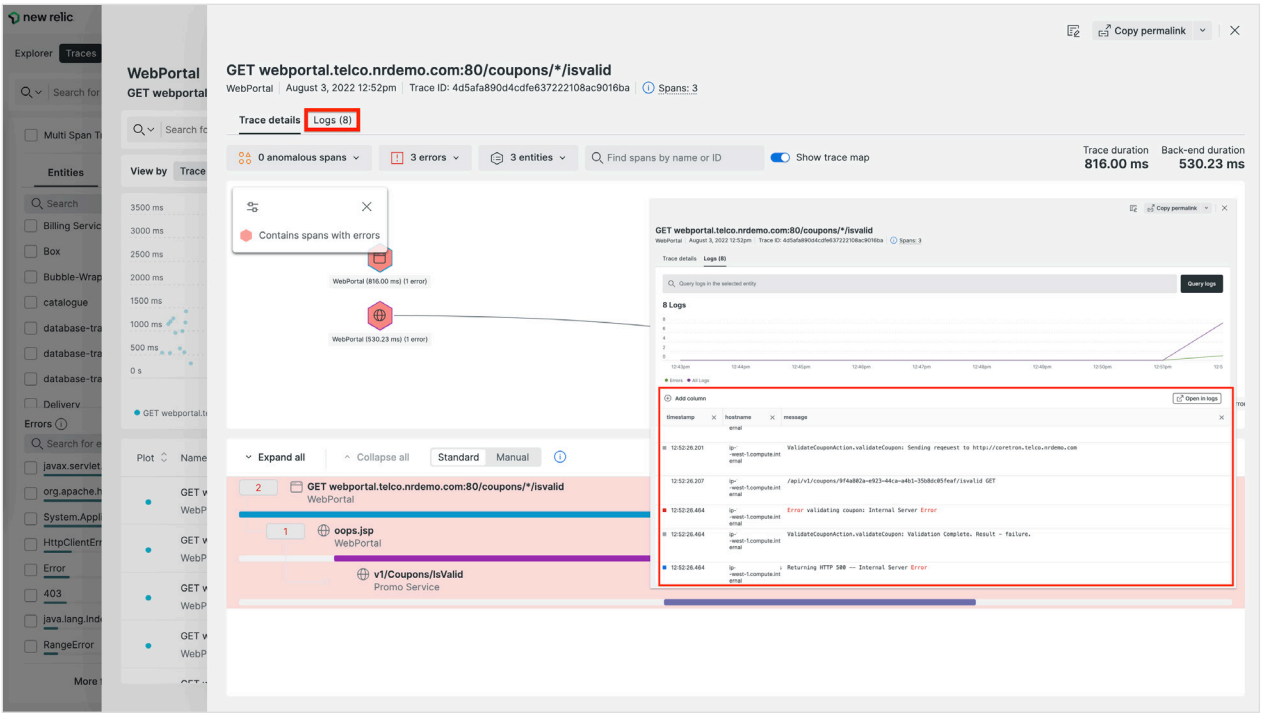
Often orchestrated or Platform-as-a-Service (PaaS) environments automatically populate logs with lots of metadata. This is great for organization, but it is also important to annotate the logs with useful qualifiers that a system can't know about, such as product version numbers, staging vs. production environments, test branches, or A/B testing versions. Log aggregation means all logs from multiple sources are collected into the same system. Without the right metadata, teams can't identify a real error log in production from a transaction that failed as part of a test run.

Another resource that can help teams identify the log source is its log forwarding. For example, most of the log forwarding solutions provided by New Relic automatically annotate the data with the type and version of the tool used to ship the data.

[2] (OpenTelemetry, n.d.)

Logs in context

It's useful for teams to see logs in context of issues in their apps and hosts. For example, the New Relic logs in context feature can add application information to logs automatically. The New Relic APM agent provides application performance management data to the logging framework and includes them in application logs. The result is that logs in context automatically correlates log data with associated application events and traces. APM errors and distributed traces link directly to the logs created during the same transaction as the error or trace. Logs in context creates this correlation by inserting a span ID, trace ID, and application name into the log messages. So, teams can bring application and log data together and troubleshoot much more quickly.



Logs filtered to show errors in context of trace in the New Relic observability platform

Log levels
Developers, DevOps practitioners, and managers sometimes refer to log levels as severity levels. Log levels describe the relative importance of the event (with terms such as debug, info, warning, error, and fatal) and the density level of information from the logging framework. A severity attribute helps filter out or discard less critical information so that teams can look solely for critical errors.

Effective use of log levels can limit the amount of data, reduce the cost of using a centralized log management tool, and keep search results speedy. In some instances, it might not be possible to control how applications generate logs, but ideally, the log management system can discard unwanted data. For example, in New Relic, teams can surface outliers using machine learning-driven patterns based on the log level. Color-coded log levels also provide a visual indicator to focus attention on the most critical areas.

Teams should use log levels with care, particularly the debug log level. Debugging can help capture very verbose messages associated with a specific behavior, but unnecessary debugging can create a significantly higher volume of logs and slow ingestion and search functions without providing additional value. Larger teams and projects may benefit from log-level standards for consistent grouping, categorizing, and logging methods.

Use logging tools and frameworks

Instead of spending time and resources implementing a logging solution from scratch, using an established, well-tested logging tool and framework can save teams time and trouble. For example, New Relic APM language agents decorate logs with the necessary metadata to provide access to the automatic logs-in-context feature and forward logs without the need to install or maintain third-party software—all in a single deployment.

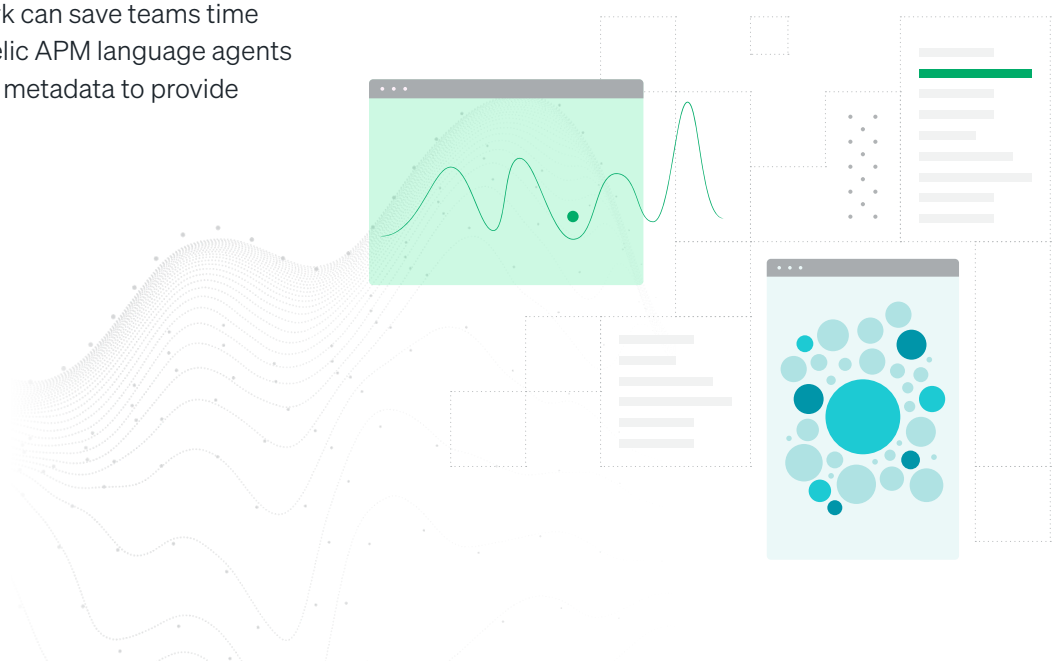
Using a consistent logging framework simplifies engineering team adoption, normalizes the log output, and ensures that teams can uniformly enable logs in context. Teams should be cautious when introducing logging frameworks and test their performance impact, just as they would with any new code.

Reference large values, don't include them

In some cases, teams need a larger chunk of data from the log to provide deeper context, like a memory dump or a set of files or images. It is usually better to save this data separately or even upload it to a designated server and reference its location in the log than to save it as a blob within the log. Teams should keep logs as lightweight as possible and access the data separately.

Share useful views, queries, and alerts

Teams should create and share standard visualizations, queries, and alerts for their logs to provide broader insight into the current state of their organization and increase cross-team visibility and communication. That is the power of full-stack observability.



What not to log

It's tempting to log anything and everything that can be useful, but teams should avoid some exceptions and pitfalls.

Sensitive information

Teams should handle sensitive information with care. It's essential to protect regulated data, such as personally identifiable information (PII) and credit card numbers, in accordance with regulations and laws, like the General Data Protection Regulation (GDPR) in the European Union[3] and the Health Insurance Portability and Accountability Act (HIPAA) in the United States.[4]

The Open Web Application Security Project (OWASP) logging guide specifies what should not be in logs, such as access tokens, passwords, sensitive information, and information individuals want to remain private.[5]

For logs stored on a private server or database, it's easy to log PII, such as names and email addresses, accidentally. To track a specific user's actions or events, teams should use anonymous identifiers. Although log data is safe in an observability platform like New Relic, they should be very cautious about transmitting PII outside the organization.

Source code and proprietary data

In addition to regulatory and compliance information, teams may want to avoid storing other sensitive information within logs, such as source code from applications or protected data within the organization.

In addition to storing logs securely, it's important to secure access to them as well. Information that can reveal trade secrets or unreleased or unannounced projects and features does not belong inside logs. So, teams should eliminate this information from logs, especially if they store logs externally on a third-party service.

Duplicate information

Adding duplicate information to logs won't break things, and having too much information is usually better than not having enough. However, including duplicate information can create unnecessary logs that don't serve a purpose, leading to higher costs.

[3] (European Commission, n.d.)
[4] (U.S. Department of Health and Human Services (HHS), n.d.)
[5] (Open Web Application Security Project (OWASP), n.d.)

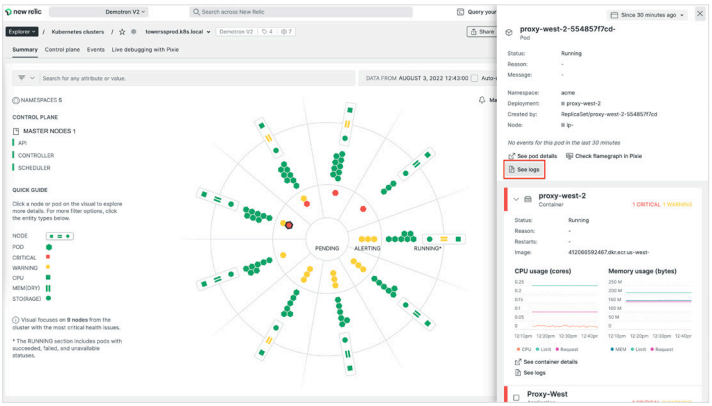
Conclusion

Making logs work to enhance full-stack observability supports real-time decisions that impact the business and ensures developers and engineers spend less time debugging and responding to incidents and more time focusing on innovation.

With these best practices in place, logs can provide the necessary detail to keep everything running smoothly for customers, enable deeper visibility into the entire stack to resolve issues quicker, and speed development.

New Relic observability platform

New Relic provides a single, unified platform for all telemetry data, including detailed logs. The New Relic observability platform incorporates log management, APM, infrastructure monitoring, serverless monitoring, mobile monitoring, browser monitoring, synthetic monitoring, distributed tracing, Kubernetes monitoring, and more. These capabilities enable organizations to visualize, analyze, and troubleshoot their entire software stack. As part of this platform, New Relic log management enables organizations to combine logging data with application and infrastructure monitoring data, resulting in a powerful, all-in-one observability platform.



APM, infrastructure, event, and access to logs combined in one view

New Relic ties together metrics, events, logs, and traces from the entire software stack integrated with AIOps (artificial intelligence for IT operations), which enables organizations to search logs faster and is more affordable than disparate legacy solutions. Instead of using separate tools in different parts of the stack, developers and engineers can view all the detailed logs related to a specific error easily in a unified view.

Speed and scalability problems in legacy logging solutions make it challenging to query detailed logs because it can take minutes or even hours to run with delayed data. In contrast, a New Relic log management search takes just seconds, so investigating and responding to incidents in the full software stack is as fast as possible.

The New Relic observability platform includes log management, a free tier for low-volume customers, and a low price per GB that enables teams to ingest all the detailed logs they need.

To begin using New Relic log management, sign up for a free account today. Free accounts include 100 GB/month of data ingest, one full-platform user, and unlimited basic users.

Sign Up Now

References

European Commission. n.d. “EU data protection rules.” European Commission. Accessed July 19, 2022. https://ec.europa.eu/info/law/law-topic/data-protection/eu-data-protection-rules_en.

New Relic, Inc. n.d. “Parsing log data.” New Relic Documentation. Accessed July 28, 2022. https://docs.newrelic.com/docs/logs/ui-data/parsing/#custom-parsing.

OpenTelemetry. n.d. “OpenTelemetry Logging Overview.” OpenTelemetry. Accessed July 18, 2022. https://opentelemetry.io/docs/reference/specification/logs/overview/.

Open Web Application Security Project (OWASP). n.d. “OWASP Logging Guide.” https://owasp.org/www-pdf-archive/OWASP_Logging_Guide.pdf.

U.S. Department of Health and Human Services (HHS). n.d. “Summary of the HIPAA Security Rule.” HHS.gov. Accessed July 19, 2022. https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html.

new relic.