New Relic

White Paper

Log Management Best Practices

Shift to Effective Logging for Full-Stack Observability

Introduction

Logging has evolved. You've 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 for observability, enabling you to quickly and easily understand how your whole system runs—and even preempt issues. That's why New Relic One provides a single unified platform for all telemetry data, including detailed logs.

It's easy to shift your logging practices to ensure that your logs enhance full-stack observability. In this article, we discuss some logging best practices for the modern organization.

What is full-stack observability?

Let's begin by talking about full-stack observability. Full-stack observability provides complete visibility into how your complex applications and systems perform from a single integrated solution for troubleshooting incidents, reducing mean time to resolution, and understanding customer experience. With all your information in one spot, you spend less time searching and troubleshooting to fix issues. In addition, you no longer have to switch tools between different parts of the software stack or have blind spots where you can't see what's happening.

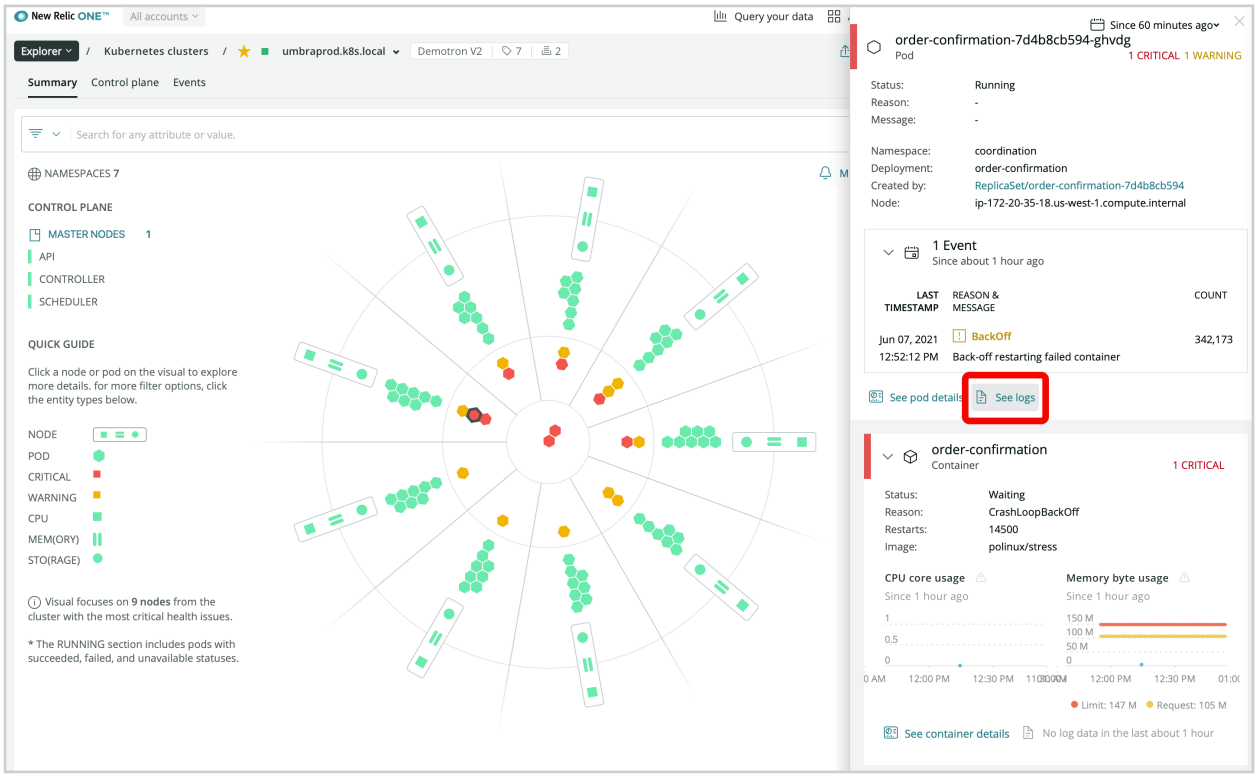
Traditional logging

Contrast this to traditional logging 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 great, it doesn't tell the entire story inside the logs of applications and infrastructure devices. And many separate and siloed monitoring and logging tools are application-centric and only focus on one piece of the stack, unable to provide complete information about what is happening and why. It's critical that you have the information you need to accelerate time to market, gain greater insight into customer behavior, and reduce incident response time.

Organizations wanting full-stack observability either chose not to have granular detail from their logs and struggle to determine the underlying cause of issues. Or, they have to leverage separate tools and try to map log details to errors and traces. Maintaining detailed logs in separate silos prevents seeing the complete picture, increases cost and time to market for products, reduces visibility into customer experience, and increases the average time to resolve issues.

New Relic One Full-Stack Observability

New Relic One Full-Stack Observability incorporates log management, APM, infrastructure monitoring, serverless monitoring, mobile monitoring, browser synthetic monitoring, distributed tracing, and Kubernetes monitoring. These capabilities enable you to visualize, analyze, and troubleshoot your entire software stack. As part of this, New Relic log management lets you 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

Full-stack observability puts together metrics, events, logs, and traces from your entire software stack integrated with artificial intelligence for IT operations (AIOps). This capability makes logs faster to search and more affordable than disparate legacy solutions. Rather than using separate tools in different parts of the stack, developers can easily view all the detailed logs related to a specific error. 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. New Relic's log management search takes just seconds, so investigating and responding to incidents in your full software stack is as fast as it can be.

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

Logging for Full-Stack Observability

Generating logs for your entire stack can be overwhelming. You might have questions about what to log, how much detail to include, and whether too much data will lead to high costs. Thousands of companies pay the high price of centralizing their log management in a different platform and are ultimately forced to limit the log data sent based on performance and price, giving them limited visibility and value to the business.

New Relic provides log management as part of the Telemetry Data Platform, which includes a free tier for low-volume customers and a low price per GB that makes it possible to ingest all the detailed logs an organization needs.

With this in mind, let's look at some best practices you can apply to your log management for observability.

Log the right things

All you need to generate a log is to write text to stdout or a file. The most important decision is choosing what goes into your logs. Your logs should include as much metadata as necessary to help you pinpoint events and root cause when investigating. These can be elements such as error messages or stack traces and related values, metrics, or events.

Everything you log should have a purpose. Whether it's usage data, user events, or application errors and exceptions, it should be valuable to your team. A basic rule of thumb for whether to log something can be to ask yourself, "Is this information immediately useful in some way, and will it provide the details I need to understand the underlying cause and make decisions?"

Anticipate common scenarios

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

Logging with some common scenarios in mind ensures the logs provide direct value to your organization. For example, user interaction logs can provide crucial insight into your customer experience; system logs can monitor issues or hardware failures. Detailed application logs may give you 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 it human-readable, your team can use the logs effectively. Third-party infrastructure tends to already capture the granular details needed. Still, for applications written in-house, you should always ask, "Am I capturing the details in the logs that will enable me to diagnose and determine the 'why,' so my users can take the necessary actions that impact the business?"

For application errors, make sure that the message conveys what is going on with that line of code. For example, an error message that says "Transaction Failed" is not as helpful as an error message with a description like "Transaction Failed: Could not create user ${path/to/file:line-number}." However, when the log includes data about the transaction, it helps a developer see why the transaction failed.

Usually, error codes or status codes in programs can also indicate your application's type of problem. Rather than just outputting the error code text or number, accompany it with a short description in the log to save another developer the time and effort of looking it up during troubleshooting.

Logs should provide critical information to your organization. Avoid logging cryptic or non-descriptive messages that only limited members of your team would understand.

Keep logs simple and concise

While it's essential to make sure you have enough information inside the log message, the opposite applies as well. Having excessive data that you don't need in the message can bloat the log storage size and costs, slow down the search logs, and distract from the core issue, making it tougher to debug.

Keep your logs concise so that you are capturing only the most critical information. Logs should contain the "why" while avoiding unnecessary noise.

If an error occurs, you can provide information about the root cause of the error without including every little detail about the environment. For example, suppose an application failed to connect and retrieve data from an internal API. In that case, it may be good to log any error messages from the API or the network state information of the environment. You probably don't need to include how much memory the application uses. Or, how many applications are running.

Don't forget the time

Remember to include a timestamp. This statement may sound obvious, but if you are used to writing logs to a database that automatically includes the date and time, you might not think to add a timestamp in your log messages. Select the most granular level that makes sense and output it inside your 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 your organization.

Another potentially obvious and vital note is to ensure you have synchronized all your systems to the same time. This enables your observability platform to use the timestamp to correlate log events with other telemetry data.

Use a log format that you can parse

If your observability platform can't extract data from your log data, it won't be very helpful. Use a log format that you can parse and keep a consistent log structure so that it is easy to collect and aggregate.

New Relic makes it easy to define custom log parsing rules, but parsing rules can't work their magic if your log data is unintelligible. Take a look at the MELT (Metrics, Events, Logs, and Traces) common format for an example of a parsable log format. If the format you use is entirely custom, setting the log type triggers customer-defined parsing rules.

When you have multiple applications that serve a common purpose, focus on standardizing a log format for all the apps. This makes incorporating data into your observability platform easier, even if the team associated with each app wants visibility into different attributes.

Log Formats in Depth

Let's take a moment to go a bit deeper into log formats. There are three consistent themes to how the text is structured with implications for usability once a log aggregation tool collects data.

The three format categories are:

- **Structured** — The most common structured format 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 common formats like comma-separated values (CSV) and tab-separated values (TSV) are also examples of structured log data.
- **Common** — A common format would not be 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 format or a common format, it is writing logs with a custom format. During log forwarding, it may be necessary to do some parsing to recognize the start and end of an individual log line. Creating customer-defined parsing rules will help make the data more valuable.

Categorize and group your logs

Specifying a data model for your logs enables you to search more effectively. Define and include attributes whenever possible to categorize and group your logs accordingly. It may be worthwhile to read through the Open Telemetry standards for logs by a coalition of industry leaders, including New Relic, as it covers many elements such as naming conventions and field value definitions. While not every framework natively supports a log formatted to exactly these standards, they would be beneficial as a guideline.

The following are a few common attributes that can be useful to have in a log data model.

Resources

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

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

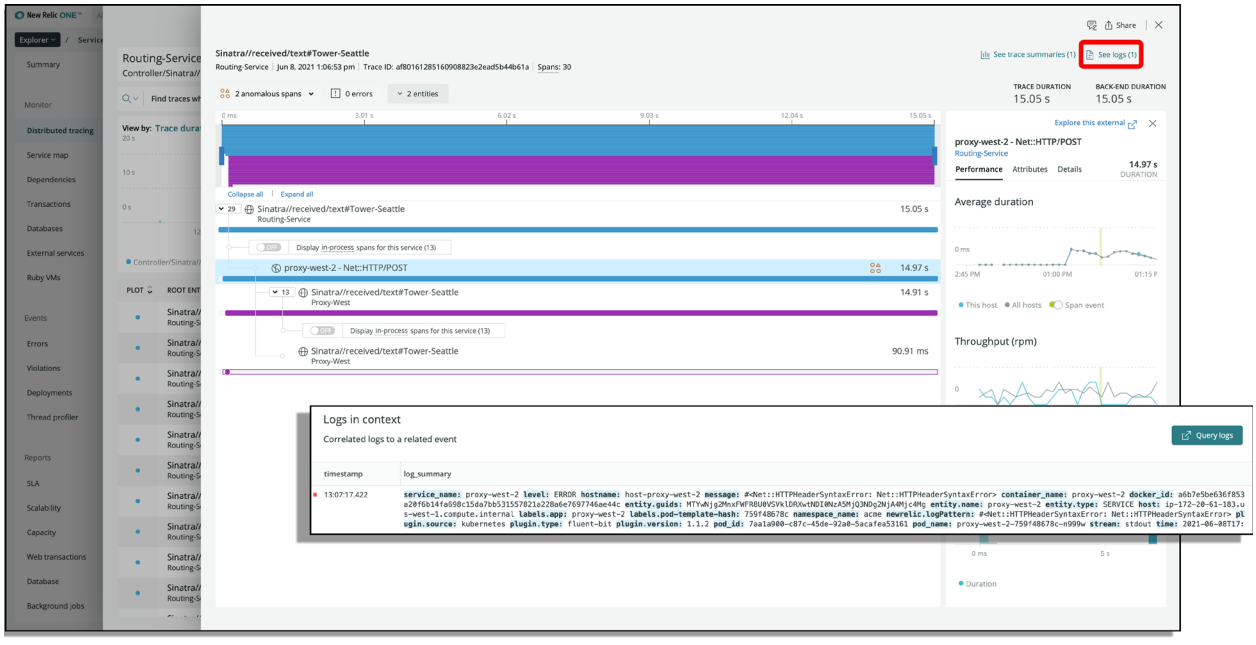
The hostname can be meaningful in logs from classic host-based applications with named environments. A pod or container ID would better organize logs from containerized or orchestrated environments.

Orchestrated environments or PaaS environments will often populate logs with a great deal of metadata automatically. This is great for organization, but it is also important to annotate the logs with qualifiers that a system can't know about. For example, product version numbers, staging vs. production environments, test branches, or A/B testing versions are useful. Remember that log aggregation means all logs from multiple sources will be collected into the same system. Without the right metadata, you won't be able to identify a real error log in production from a transaction that failed as part of a test run.

Another resource that can help identify the source of the log is its log forwarding. 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.

Logs in context

Logs in context is a New Relic feature that can add application information to logs automatically. Application performance management data is provided by the New Relic APM agent to your logging framework and included in your application logs. The result is that logs in context automatically correlates log data with associated application events and traces. APM errors and distributed traces will directly link 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. This means you can bring application and log data together and troubleshoot much more quickly.



Logs filtered to show errors in context of trace

To learn more about what this achieves and how to implement it, check out the documentation on configuring logs in context.

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. Having a severity attribute helps to filter out or discard less critical information so that you can look for critical errors solely.

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 for you to control how applications generate logs, but the log management system can also discard unwanted data. Once in New Relic One, you can surface outliers using machine learning-driven patterns based on the log level. Color-coded log levels also give you a visual indicator to focus your attention on the most critical areas.

Please note that you should use log levels with care, particularly when using the debug log level. Debug can be very helpful when you need to capture very verbose messages associated with a specific behavior, but if you use it unnecessarily, it can create a significantly higher volume of logs and slow your ingestion and search functions without providing any additional value. It may be beneficial for larger teams and projects to communicate a standard for log levels so that grouping, categorizing, and logging methods are consistent.

Use logging tools and frameworks

Use existing, well-tested logging tools and frameworks. Instead of spending time and resources implementing a logging solution from scratch, using an established framework can save time and trouble. Using a consistent logging framework simplifies adoption in engineering teams, normalizes the log output, and ensures that you can uniformly enable logs in context. Be careful when introducing logging frameworks and test the performance impact they have, just as you would do with any new code.

Reference large values, don't include them

In some cases, you need a larger chunk of data from your 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. Keep your logs as lightweight as possible and access the data separately.

Share useful views, queries, and alerts

Finally, create and share standard visualizations, queries, and alerts for your team's logs. This can give broader insight into the current state of your organization and increase cross-team visibility and communication. That is the power of full-stack observability. Why not have more eyes on what is going on?

What Not to Log

Even though you may want to log anything and everything that can be useful, let's also go through some exceptions and pitfalls you should try to avoid.

Sensitive information

You must handle sensitive information with care. It's essential to protect regulated data such as personally identifiable information (PII) and credit card numbers. Consider laws such as the European Union's General Data Protection Regulation (GDPR) and the United States' Health Insurance Portability and Accountability Act (HIPAA).

The Open Web Application Security Project's logging guidelines specify what should not be in logs, such as access tokens, passwords, sensitive information, and information individuals want to remain private.

When you keep logs on a private server or database, it can be easy to log PII accidentally, such as names and email addresses. It usually does not enhance observability, but if you must track a specific user's actions or events, it is helpful to use anonymous identifiers. Although your log data is safe in an observability platform like New Relic's, it is important to be very cautious about transmitting PII outside your organization.

Source code and proprietary data

Besides regulatory and compliance information, there may be other information you want to avoid storing within your logs. This can include source code from applications or protected data within your organization.

Even if you store your logs securely, access to them might not be. Information that can reveal trade secrets or unreleased or unannounced projects and features do not belong inside logs. So, be sure to keep them out of your logs, especially if you store your logs externally on a third-party's service.

Duplicate information

Finally, adding duplicate information to your logs won't break things, and having too much information is usually better than not having enough. However, including a lot of identical information can create unnecessary logs, leading to higher costs without serving a purpose.

Conclusion

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

With these best practices in place, your logs can give you the detail you need to keep everything running smoothly for your customers, have deeper visibility into your entire stack to fix issues quicker, and speed up your development.

To get started using your logs for observability and get the most out of log management, sign up for a free account and try New Relic log management.