

OBSERVABILITY

Metrics, Traces & Logs

INTRO

Organizations face rising network complexity. Cloud and hybrid networks. Hundreds if not thousands of devices, applications, APIs. It is multiplied by even more distributed systems, interactions and dependencies. As this complexity escalates, managing the health and performance is paramount to keeping systems up and running – reliably and efficiently.

Most organizations have all the information they need to figure out any issues, but it would take a lot of manual work to get there. And most organizations do not have the time or resources to connect the dots across all the available data. This is especially true for modern environments built around micro-services. These dispersed workloads must be well connected, high performing, and be able to identify bottlenecks, performance issues, or security problems.

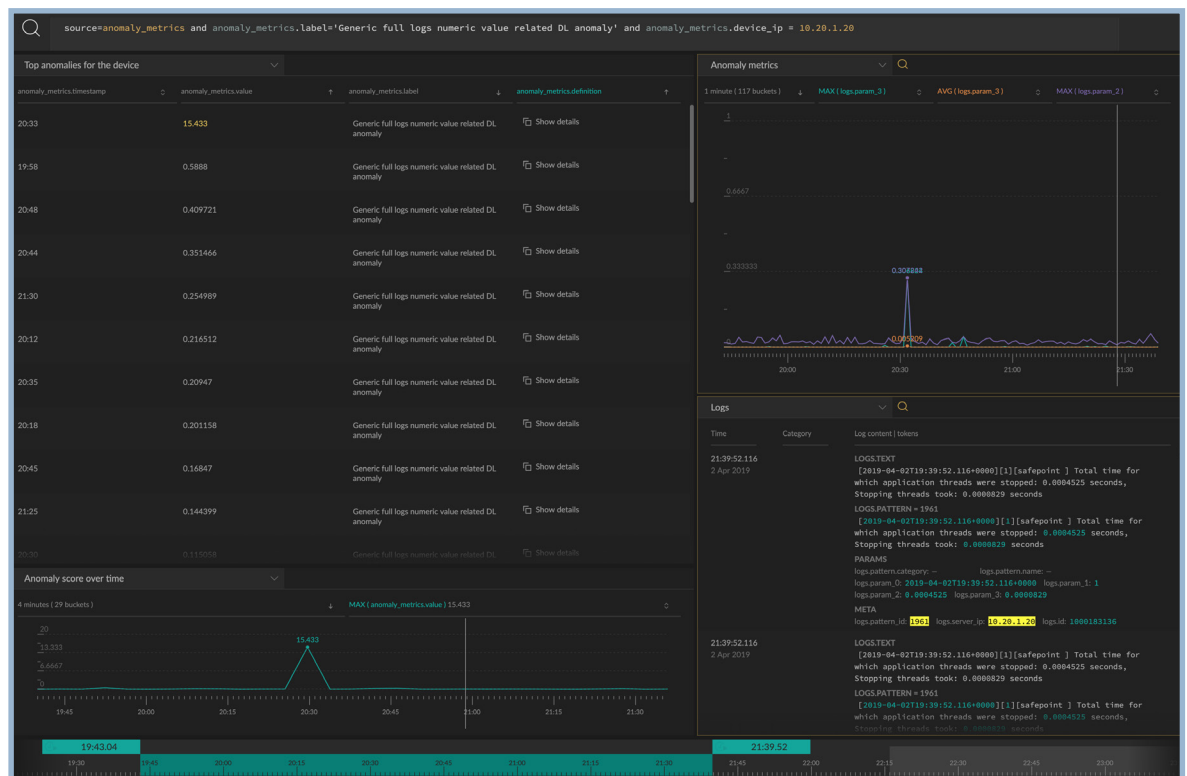
Observability. Whether buzzword or fact, there's no denying that modern organizations should analyze performance indicators, understand their impact on adjacent services or applications, and get notified when something isn't right. Even then, debugging a complex program and determining the root cause of any issues are major hurdles.

In this ebook, we'll discuss why observability matters, the three pillars of observability, and how you can use observability for fast, accurate troubleshooting and application performance management to speed up your development cycle.



MONITORING (& BEYOND)

Monitoring is a critical requirement for building and managing applications. It offers a panoramic view of the overall performance and health of the system and helps you understand where the gaps may be located. Monitoring provides a snapshot in time. This snapshot is great for present or historical views during a specific period when there was a known issue (such as blackbox tests, infrastructure and applications updates, or during the A/B tests).



But monitoring is passive. Except for simple extrapolation, monitoring alone does not give you enough information to predict and plan for future problems. While collecting and storing the data is important, it is of little use to an engineering team unless it can be visualized in a way to tell a relevant story.

This is where “Observability” comes in. Observability gives us highly granular insights into the behavior of systems. It combines those insights with context. And together, proper observability tells a great story.



SEEING IS BELIEVING

A chart is the most basic visualization unit in observability products. Typically charts are organized into dashboards. Or they can be created on the fly to quickly share information while diagnosing an incident or rolling out a new application version.



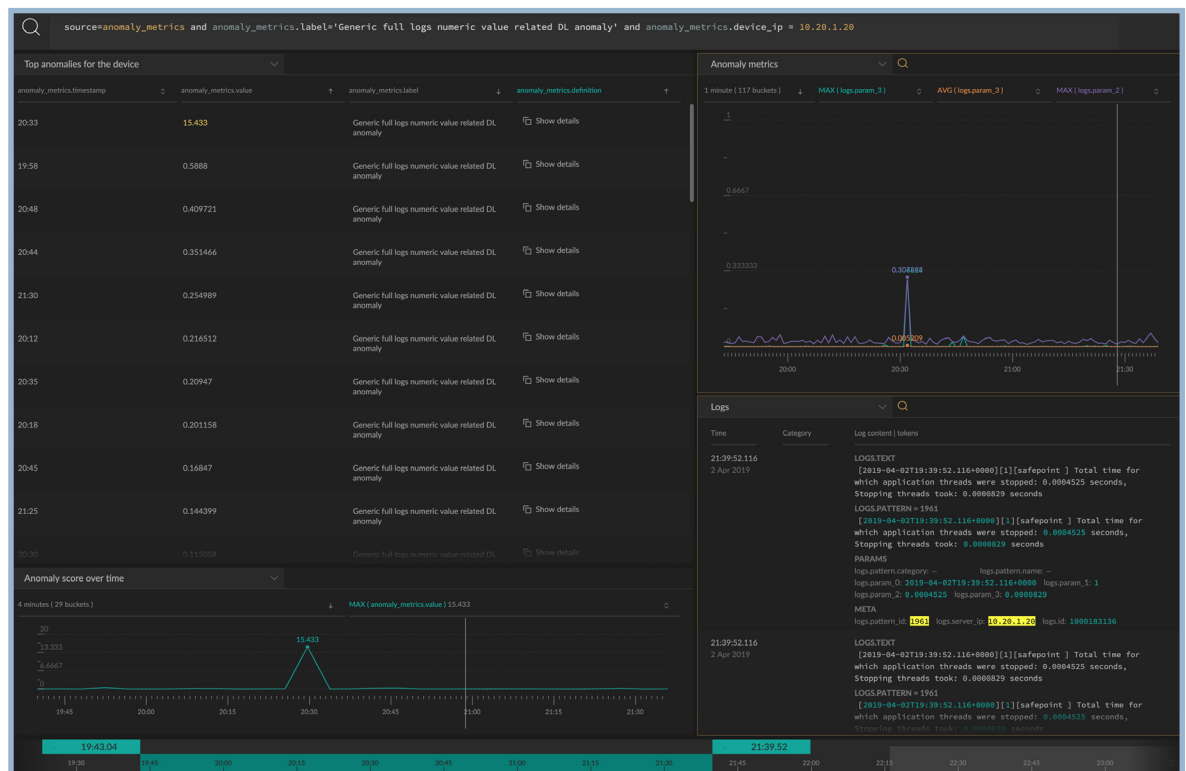
Dashboards and charts are equipped with many tools to help engineers drill down. They can change the arrangement and presentation of their data with stack and fill options. They can toggle between linear and logarithmic chart scales. They can select different time granularities (per-minute, per-hour, per-day)

Additionally, engineers need to be able to view near real-time data or quickly go back into historical data. The point here is that dashboards are great - as long as the output tells a story. The challenge comes when most of the products today treat unstructured data like plain text, which we all know is hard to use to visualize a story.



METRICS

Metrics are measures of the system's performance. Examples include response time, transactions per second, memory usage, and uptime. The information they provide is general but important. Any investigation needs to start by identifying an issue at the macro level. Is performance slower than it should be? Are users unable to log in at times? Metrics let the software team determine the existence and severity of problems and set priorities.

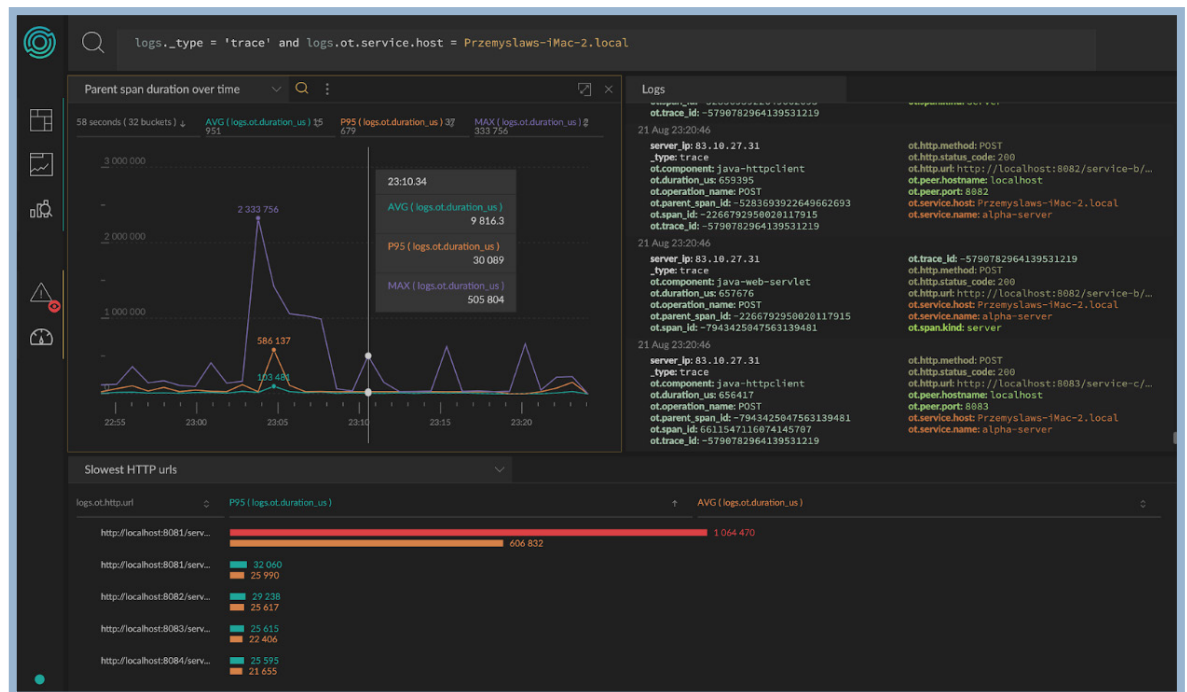


For example, when we monitor an HTTP service, one of the key metrics we want to monitor is the requests per second. A sudden change in traffic is a good indicator of problems. It can be malicious behavior, wrong service configuration, or issues with other parts of the system. Metrics give us this visibility.



TRACING

Tracing can take many forms depending on the tools used. Enabling a trace may or may not require modifying the code. It may cover only the most significant events or may record them all. Unlike logging, tracing is systematic and requires little effort by developers. A ton of information can be generated, but enabling it selectively (e.g., per source file) can make it more manageable and useful.



Traces can tell you if the system is behaving good or badly, or if any issues are happening. Traces allow you to get into the details of particular requests. You can find which request has had issues and where, where the bottlenecks are, and so forth.

For example, a metric might reveal that a certain type of query is consistently slow. A trace will show the code path which is running inefficiently, and perhaps even the exact function where there's a long delay. The hardest part is determining exactly why there's a bottleneck in that function and what changes are necessary.

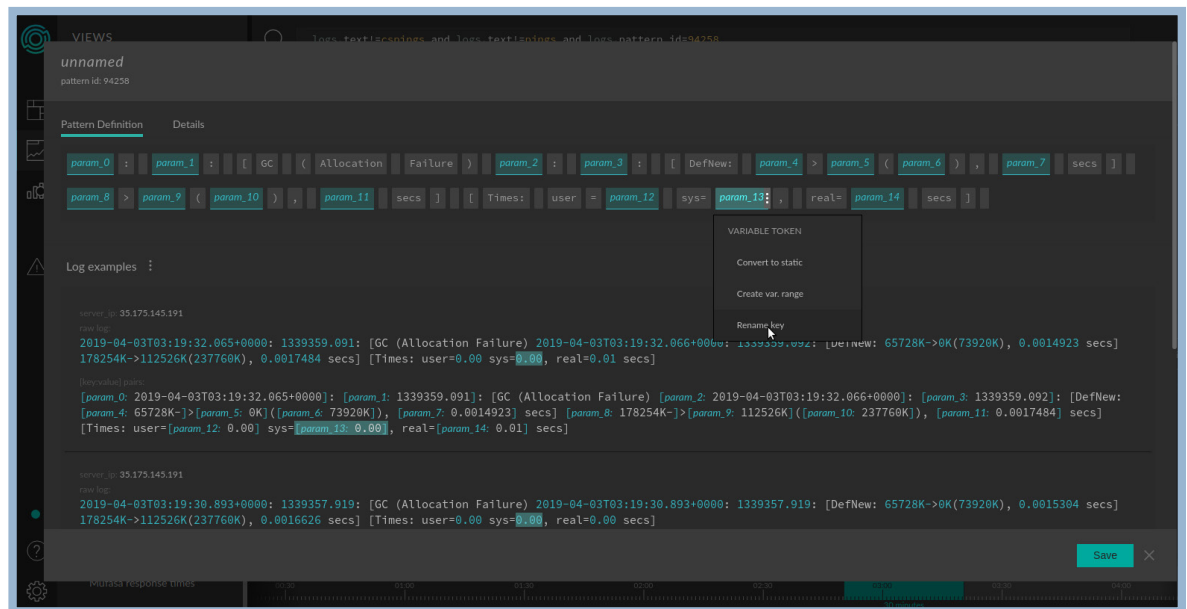
For example, sometimes our systems are being slowed down by external components like databases. Tracing helps us to find out if the JDBC driver or a specific query that we call is not a root cause.



LOGS

Logs contain the most detailed information, but they are the hardest to manage. As log data has grown exponentially, so too has the need to manage it. And a key part of data management has been to find answers to problems and resolve them as fast as possible.

Most of the log content comes from logging statements in the code. Developers usually write human-readable messages, such as “no match found for key” so logs consist predominantly of unstructured data. The volume of log data can be vast, especially when many instances of a service are running concurrently. Unstructured data is one of the biggest challenges. The bottom line is that most organizations have all of the data at hand, but they do not have the time or resources to sift through it all manually to get the relevant observability.



For example, sometimes we need to deny HTTP traffic to some IPs because we want it to remain within the organization. If we log the client's IP and monitor those logs, we can spot if the HTTP server has been configured properly.



WHY LOGSENSE

With LogSense, we embed tracing with the logs, giving you a way to understand what has happened and perhaps more importantly, why something happened. This is great for development teams and DevOps teams to understand what caused the issues and how to fix them efficiently, which ultimately makes them run much faster.

Tracing allows us to get very precise information about what was happening with a request. For example, if there were services that had separate requests coming in: maybe there was a message queue on the way and some other services, and sometimes those typologies are very complex. This allows us to find the bottlenecks. Even when a bottleneck is found, the cause of the bottleneck is not always clear. This is when logs come in handy.

Legacy tracing or APM solutions assume that the developer will write the custom logs using their API. Custom APIs are very inefficient because you need developers to spend time on changing the applications to work better with tracing. In fact, normally they are already using some logging solutions. For example, in Java they might be using log4j or logback. In Python, there's Python logging.


With LogSense, our implementation of tracing (with logs) combines information about the trace with the logs. So each log that is being admitted within a single trace, it's including information about which trace this was, which means you are no longer dealing with tens of thousands of logs for a given trace and you are dealing only with those specific logs that are related to that particular trace. And this approach allows you to find the root cause of the problem very quickly.

CONCLUSION

By applying the lessons learned from root cause analysis, development and engineering teams can refine their systems to avoid repeating the same issues over and over, and allow them to focus on innovating.

High observability means having not just a mountain of data, but rather high-quality information on any ongoing issues. LogSense provides intelligent processing of the vast amounts of diagnostic data, making it more usable.

Observability hype is on the rise – and rightfully so. It allows teams to take greater ownership of uptime and performance, giving you a quantifiable way to measure success.





LogSense is a cloud-based log management solution that combines powerful log parsing capabilities with easy-to-use analytics to help developers make sense of logs – all of them.

With LogSense automated pattern discovery, users get instant observability across custom and historical logs, without requiring pre-defined criteria or rules. This clear visualization means smarter insights and faster resolutions to promote agile application development, reduce support burden, and minimize security risks. LogSense integrates directly with existing workflows and can be configured for on-premises, cloud, or hybrid environments.

Sales

TOLL-FREE

+1 877-LOGSENSE

+1 877-564-7367

EMAIL

sales@logsense.com

www.logsense.com