



Monitoring and Observability

Complex Systems are Unpredictable

- Inherent complexity of large software systems and unexpected behaviors.
- Reducing downtime, ensuring customer satisfaction, and maintaining system reliability.
- Monitoring: Watching out for known issues.
- Observability: Investigating unknown system behaviors.

Monitoring scenarios

- **Scenario 1: Performance Metrics** - When you need to track system performance, like CPU usage, memory usage, or response times.
- **Scenario 2: System Health** - To ensure system availability and health, such as monitoring server uptime or application health statuses.
- **Scenario 3: Security Monitoring** - For detecting potential security threats or breaches, like unauthorized access attempts.
- **Scenario 4: User Engagement** - checking how users are interacting with your system, detect trends. This is helpful for understanding your users better.
- Example: An e-commerce website monitoring their server health during a Black Friday sale.

Monitoring in Leadspotr

- Most API interactions with Leadspotr don't require high computational load, except for the part of the API that generates and sends the PDF report. This could be a place where we could add monitoring in the form of measuring the time it takes to process this and send a report/message automatically when the processing exceeds a given target time. Also, cloud providers offer memory/cpu usage logs so you can check how well the cloud resources are being used.
- System/application health checks can be run on standard endpoints for health checks. Another thing you could do is regular consistency checks (i.e. ensure that

each quiz and each user is part of a valid organization, that each submission has a submitted date etc).

- Regular access checks: report on endpoints being called more than usual, logging hacking attempts (i.e. someone trying to access a wp-admin endpoint), or many failed authentications in a row.

Observability scenarios

- **Scenario 1: Debugging Complex Issues** - When diagnosing issues not immediately apparent from initial alerts or metrics.
- **Scenario 2: Understanding System Interactions** - To gain insights into how different components of a system interact with each other.
- **Scenario 3: Continuous Improvement** - For continuously improving system performance and user experience based on trends and patterns.
- Example: Troubleshooting an intermittent issue where in some cases an API endpoint is not called.

Observability in Leadspotr

- Debug an issue where a PDF report is not sent.
- Go through an OAuth flow (login via GitHub) to fix an issue with scopes.
- Noticing that lots of people start a quiz but do not finish it: might be an indication of a showstopper bug.
- Other common bugs may occur due to access restrictions, these are hard to test but making sure the user doesn't get random "access denied" errors has a big impact on usability of your application.

Tools for Monitoring and Observability

Logs: The First Line of Insight

- Logs are records of events or transactions that occur within a system.
- Logs capture everything from user actions to system errors.
- Different types of logs: error logs, access logs, transaction logs.

Why is Logging Important?

- Logs are crucial for troubleshooting issues and identifying system errors.
- Analyzing logs helps in understanding how users interact with your application.
- Access logs can reveal user engagement patterns.
- Logs serve as an audit trail for system activities, aiding in compliance and security.
- In financial systems in particular, transaction logs are essential for audit trails.

Best Practices for Logging

- Opt for structured logs in a format like JSON for easier parsing and analysis.
- You could standardize this so that each log is a string + JSON data.
- Logs should be treated as a stream (see the 12-factor app).
- Implement log levels like DEBUG, INFO, WARN, ERROR to categorize log severity.
- You can define the level at which you want to log in your application at the start, read it from an environment variable, or even read it from a database.
- Not all logs need to be kept forever; determine what needs long-term storage and what doesn't.
- Don't write sensitive data like passwords or personal information in logs.
- Implement filters or masks to ensure confidential data is not accidentally logged.

Metrics: Quantifying System Health

- Metrics are numeric data points, collected over intervals, representing system health.
- Common metrics: CPU usage, memory usage, response times, error rates, etc.
 - Example of metrics in Google Cloud for Learntail.
- Best practice: use the tools that are built into most cloud providers, set up alerts, keep historical data for trend analysis.

Tracing: Following a Request's Journey

- What is tracing? Monitoring individual requests as they traverse multiple components.
- Tracing can help diagnose latency issues, understanding system interactions.
- Just like metrics, tracing tools are also built into cloud providers.

Telemetry

- Telemetry is the automated communication process of measurements and other data from remote sources.
- Benefits: real-time insight, proactive issue detection, and automated analytics.
- Goes beyond system metrics and also includes sales numbers, website visits, youtube video views, etc.
- Data dashboards built with for example Streamlit

Summary and Key Takeaways

- The importance of monitoring and observability for complex systems.
- Tools like logs, metrics, tracing, and telemetry provide deep insights.
- Proactive monitoring reduces system failures and enhances performance.