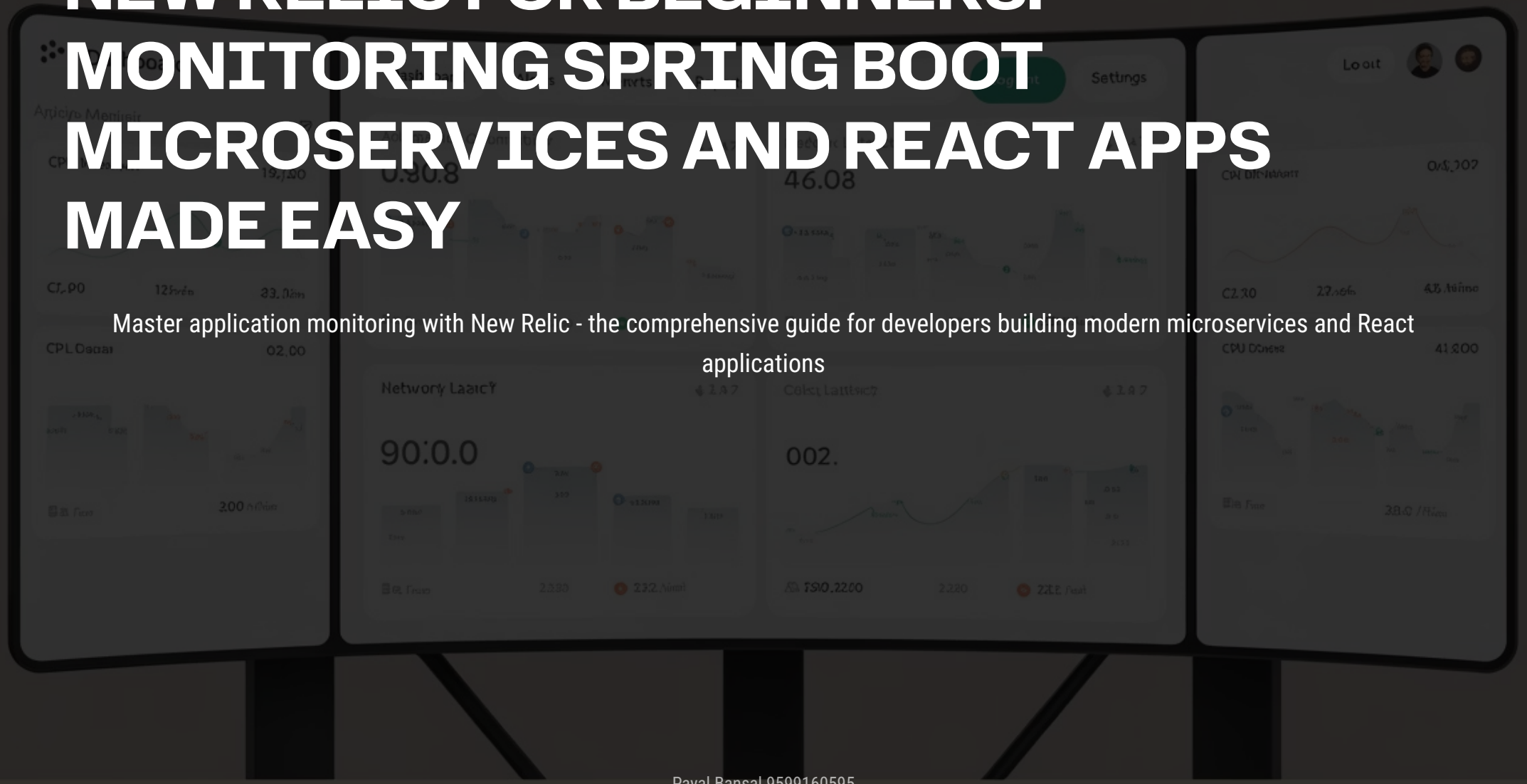
NEW RELIC FOR BEGINNERS: MONITORING SPRING BOOT MICROSERVICES AND REACT APPS MADE EASY

Master application monitoring with New Relic - the comprehensive guide for developers building modern microservices and React applications



CHAPTER 1: INTRODUCTION TO NEW RELIC AND WHY IT MATTERS

New Relic is a powerful monitoring platform that revolutionizes how developers understand and optimize their applications. Think of it as having X-ray vision into your software - you can see exactly what's happening inside your applications, from database queries to user interactions, all in real-time.

In today's complex software landscape, applications are no longer simple monolithic structures. Modern applications consist of multiple microservices, frontend components, databases, and third-party integrations - all working together in a delicate ecosystem. When something goes wrong, finding the root cause can be like searching for a needle in a haystack.

This is where New Relic becomes invaluable. Instead of waiting for users to report issues or spending hours debugging problems in production, New Relic provides a comprehensive dashboard that shows you exactly where your application slows down, crashes, or encounters errors - often before your users even notice.

REAL-TIME VISIBILITY

See exactly what's happening in your applications as it happens, with detailed performance metrics and error tracking.

PROACTIVE PROBLEM DETECTION

Identify and resolve issues before they impact your users, with intelligent alerts and anomaly detection.

END-TO-END MONITORING

Monitor your entire application stack from frontend to backend, including microservices and infrastructure.

Monitoring is especially crucial for microservices architecture and React applications because issues can originate from multiple sources - backend services, frontend code, network connectivity, or external dependencies. Without proper monitoring, diagnosing problems becomes a time-consuming process of elimination. New Relic eliminates this guesswork by providing complete observability into your application ecosystem.

WHAT IS APPLICATION PERFORMANCE MONITORING (APM)?

Application Performance Monitoring (APM) is like having a comprehensive health monitoring system for your software applications. Just as a doctor uses various instruments to check your vital signs, APM tools like New Relic continuously monitor your application's vital signs - response times, error rates, throughput, and resource usage.

RESPONSE TIME TRACKING

Monitor how quickly your application responds to user requests. Slow response times directly impact user satisfaction and can indicate performance bottlenecks in your code or infrastructure.

ERROR RATE ANALYSIS

Track the frequency and types of errors occurring in your application. This includes HTTP errors, database connection failures, and application exceptions that could break user workflows.

THROUGHPUT MEASUREMENT

Understand how many requests your application handles over time. This metric helps you plan for scaling and identify traffic patterns that might cause performance issues.

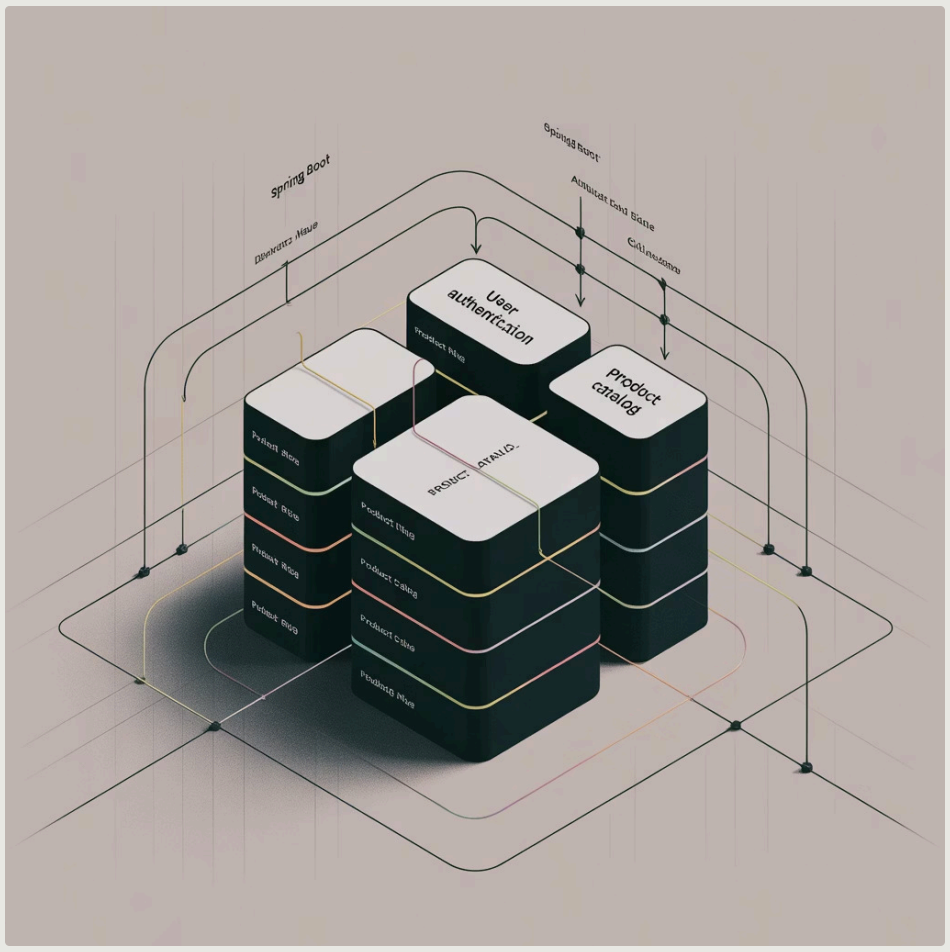
RESOURCE UTILIZATION

Monitor CPU usage, memory consumption, and database performance. Resource bottlenecks often cause application slowdowns and can be expensive if left unaddressed.

APM tools excel at identifying bottlenecks - those specific points in your application where performance degrades. This could be a slow database query that takes 10 seconds to execute, an external API call that frequently times out, or a memory leak that gradually consumes server resources. By pinpointing these issues with surgical precision, APM enables developers to focus their optimization efforts where they'll have the greatest impact.

> "Think of APM as a doctor's checkup for your software - it provides regular health assessments and early warning signs of potential problems."

WHY USE NEW RELIC WITH SPRING BOOT MICROSERVICES?



Spring Boot has become the gold standard for building microservices in the Java ecosystem, and for good reason. It provides powerful features for creating small, independent services that can be developed, deployed, and scaled independently. However, this architectural approach introduces new challenges that traditional monitoring approaches can't handle effectively.

In a microservices architecture, your application is no longer a single, monolithic entity. Instead, it's composed of dozens or even hundreds of small services, each with its own database, API endpoints, and business logic. These services communicate with each other through HTTP calls, message queues, or event streams, creating a complex web of dependencies.

01

SERVICE INDEPENDENCE

Each microservice operates independently, making it difficult to trace issues across service boundaries without proper monitoring.

02

DISTRIBUTED COMPLEXITY

A single user request might touch 10+ microservices, and any one of them could be the source of performance problems.

03

NETWORK COMMUNICATION

Services communicate over the network, introducing latency and potential failure points that didn't exist in monolithic applications.

04

DATA CONSISTENCY

With distributed data storage, maintaining consistency and tracking data flow becomes exponentially more complex.

New Relic addresses these challenges through distributed tracing, a powerful feature that follows a single request as it travels through your entire microservices ecosystem. When a user clicks a button in your React frontend, New Relic can show you the complete journey: which services were called, how long each service took to respond, where errors occurred, and which database queries were executed. This end-to-end visibility is essential for maintaining reliable microservices applications.

Additionally, New Relic provides service maps that visualize the relationships between your microservices, showing you which services depend on each other and how healthy those connections are. This helps you understand the impact of changes and quickly identify which service is causing downstream problems.

WHY MONITOR REACT APPLICATIONS WITH NEW RELIC?

FRONTEND PERFORMANCE

React applications run entirely in users' browsers, making frontend performance directly visible to users and crucial for user experience.

REAL USER MONITORING

Unlike backend services running in controlled environments, frontend apps face unpredictable conditions: slow devices, poor network connections, and different browsers.

React applications present unique monitoring challenges because they execute in your users' browsers rather than on your servers. This means performance can vary dramatically based on factors completely outside your control: the user's device capabilities, internet connection speed, browser version, and even what other applications they're running.

New Relic Browser monitoring specifically addresses these frontend challenges by collecting real user data from actual browser sessions. This includes detailed timing information about page loads, JavaScript execution, AJAX requests, and user interactions.

PAGE LOAD MONITORING

Track how long it takes for your React components to render and become interactive. Slow initial page loads are one of the biggest factors in user abandonment.

AJAX PERFORMANCE

Monitor API calls from your React app to backend services, identifying slow or failing requests that impact user experience.

1 2 3 4

JAVASCRIPT ERROR TRACKING

Capture and analyze JavaScript errors that occur in production, including stack traces and the user actions that triggered them.

USER JOURNEY ANALYSIS

Understand how users navigate through your application and where they encounter problems or abandonment points.

React's component-based architecture and virtual DOM provide excellent user experiences when properly optimized, but they can also introduce performance pitfalls. Common issues include unnecessary re-renders, large bundle sizes, memory leaks from uncleared event listeners, and inefficient state management. New Relic helps you identify these React-specific performance problems by providing detailed insights into component render times, bundle loading performance, and JavaScript execution patterns.

Perhaps most importantly, New Relic Browser monitoring gives you the user's perspective. While your application might perform well in your development environment, real users face different conditions. New Relic shows you performance data segmented by browser type, device category, geographic location, and connection speed, helping you understand how your React application performs for your actual user base.

NEW RELIC'S CORE MONITORING FEATURES EXPLAINED

New Relic offers a comprehensive suite of monitoring capabilities designed to give you deep insights into the performance and health of your entire software stack, from your backend servers to your users' browsers. Let's explore some of its core features and why they're essential for modern applications.

APM (APPLICATION PERFORMANCE MONITORING) TRANSACTION TRACES

What it is: Imagine a single customer's request, like adding an item to a shopping cart or submitting a form. APM Transaction Traces follow that request from the moment it hits your application's server, through every single function call, database query, and external service interaction, until the response is sent back. It's like having a detailed map and stopwatch for every journey a request takes through your system.

Why it's valuable: This feature is crucial for quickly identifying performance bottlenecks. If your checkout process is slow, a transaction trace can tell you exactly which part of the code, which database query, or which external API call is causing the delay. This visibility is invaluable for developers trying to optimize their application's speed and reliability, especially in complex microservices architectures.

BROWSER MONITORING

What it is: While APM focuses on your server-side application, Browser Monitoring gives you the "end-user perspective." It tracks how your application performs directly in your users' web browsers. This includes metrics like page load times, how long it takes for a page to become interactive, and most importantly, it captures JavaScript errors that users encounter.

Why it's valuable: Your users don't see your backend; they experience your application through their browser. Browser monitoring reveals the real-world performance under diverse conditions (different devices, networks, and locations). It helps you identify slow loading pages, diagnose frontend issues (like broken buttons or forms caused by JavaScript errors), and understand how performance impacts actual user experience, which is critical for satisfaction and retention.

INFRASTRUCTURE MONITORING

What it is: Infrastructure Monitoring provides visibility into the health and performance of the underlying foundation of your applications. This includes servers (physical or virtual), containers (like Docker or Kubernetes pods), and cloud services (like AWS EC2, Azure VMs, Google Cloud Compute Engine). It collects metrics such as CPU utilization, memory usage, disk I/O, network traffic, and process health.

Why it's valuable: Applications can only perform as well as the infrastructure they run on. If a server runs out of memory or a database server is overloaded, your application will suffer, regardless of how optimized its code is. Infrastructure monitoring helps you proactively identify and resolve resource constraints, predict future capacity needs, and understand how infrastructure issues might be impacting application performance. It's essential for maintaining stable and scalable environments.

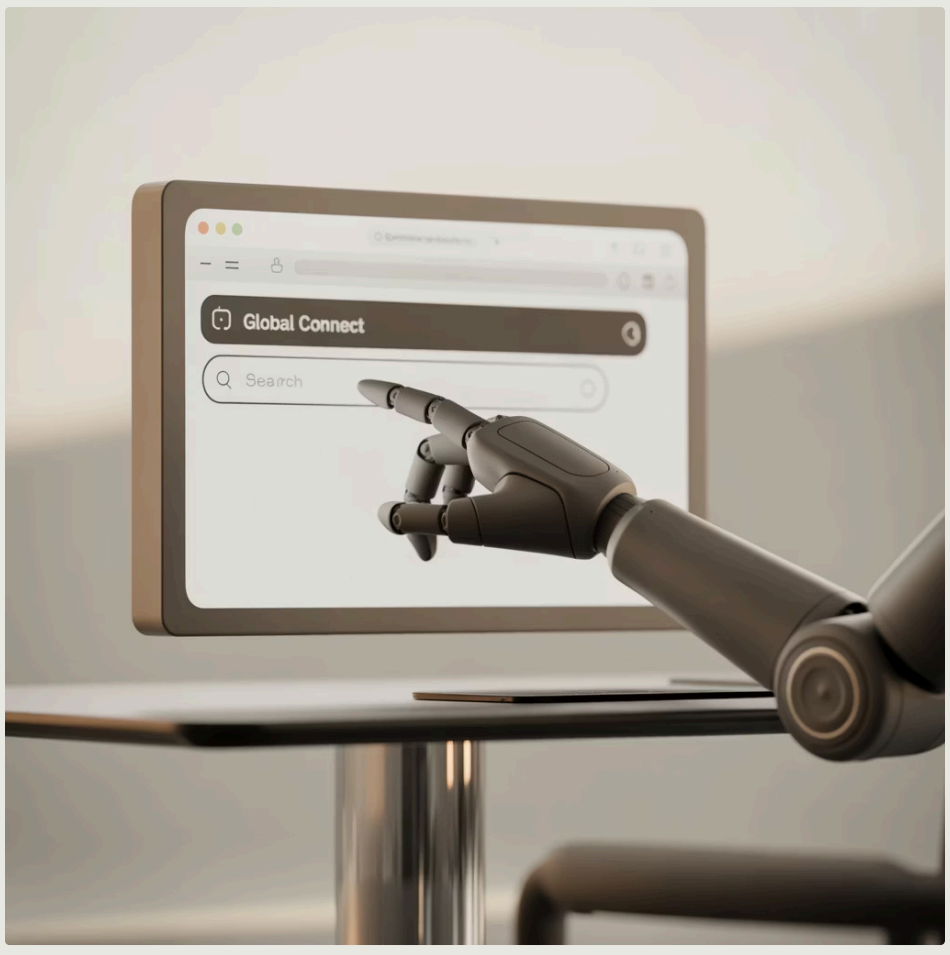
LOG MONITORING

What it is: Log Monitoring centralizes all the log data generated by your applications, servers, and other services into a single, searchable platform. Instead of sifting through countless log files spread across different machines, you can collect, parse, and analyze them from one place. New Relic also correlates these logs with your performance data, connecting specific events or errors in logs to performance spikes or transaction failures.

Why it's valuable: Logs are the narrative of your system's behavior. When something goes wrong, logs often contain the clues you need to diagnose the problem. Centralized log monitoring makes troubleshooting much faster and more efficient. By correlating logs with performance data, you can quickly jump from a slow transaction identified by APM to the exact log entry that explains why it was slow (e.g., a specific error message or a long-running query debug message). This holistic view speeds up problem resolution significantly.

EXTENDING YOUR MONITORING: ADVANCED NEW RELIC CAPABILITIES

Beyond core application and infrastructure insights, New Relic offers specialized features to proactively identify issues, analyze errors, and ensure your team is instantly aware of critical problems. Let's explore three powerful additions to your monitoring toolkit.



SYNTHETIC MONITORING

What it is: Synthetic Monitoring acts like a vigilant, automated user constantly checking your application from various locations around the world. It simulates real user interactions and API calls to ensure your services are available and performing as expected, 24/7, even when real users aren't active.

How it works: New Relic sets up "monitors" that perform predefined actions, such as navigating through a login process, completing a shopping cart checkout, or simply checking if an API endpoint responds. These checks are run at regular intervals from New Relic's global network of locations. If a check fails or experiences a performance degradation (e.g., a page takes too long to load), it triggers an alert.

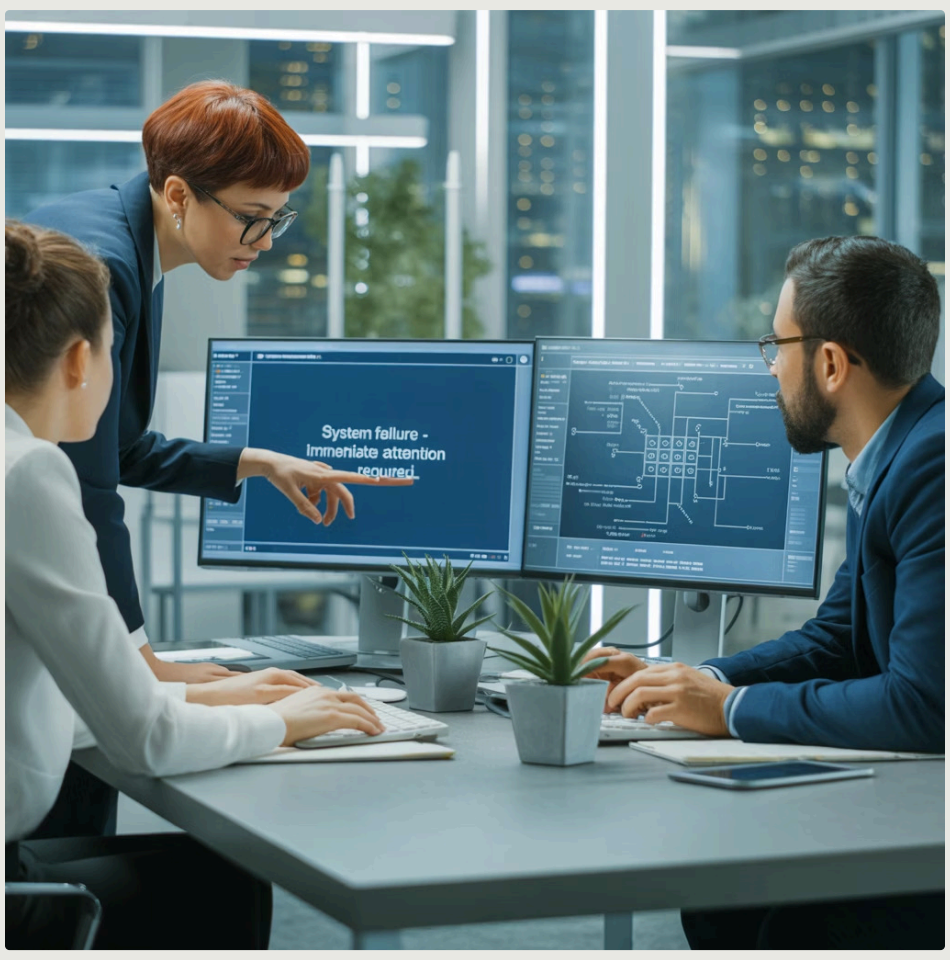
Why it's valuable: This feature is essential for proactive problem detection. Synthetic Monitoring can catch issues like broken links, slow page loads, or even full outages before your actual customers encounter them, providing you with time to fix problems before they impact user experience and revenue. It's particularly useful for monitoring critical user flows and ensuring global availability.

ERROR ANALYTICS

What it is: Error Analytics is your application's built-in detective for understanding why things go wrong. It automatically tracks, collects, and analyzes every application error, exception, and crash report generated by your code, giving you a clear picture of what's breaking and how often.

How it works: Instead of just reporting a single error, New Relic groups similar errors together, identifies their root causes, and highlights trends. It shows you which errors are most frequent, which parts of your application are generating the most errors, and how many users are affected. You can see the full stack traces and context around each error, making debugging significantly easier.

Why it's essential: This feature transforms chaotic error logs into actionable insights. It helps you quickly identify and prioritize the most impactful bugs, understand if recent code deployments introduced new errors, and track whether your fixes are truly working. By understanding error patterns, development teams can improve code quality and prevent recurring issues, ultimately leading to a more stable and reliable application.



ALERTS AND NOTIFICATIONS

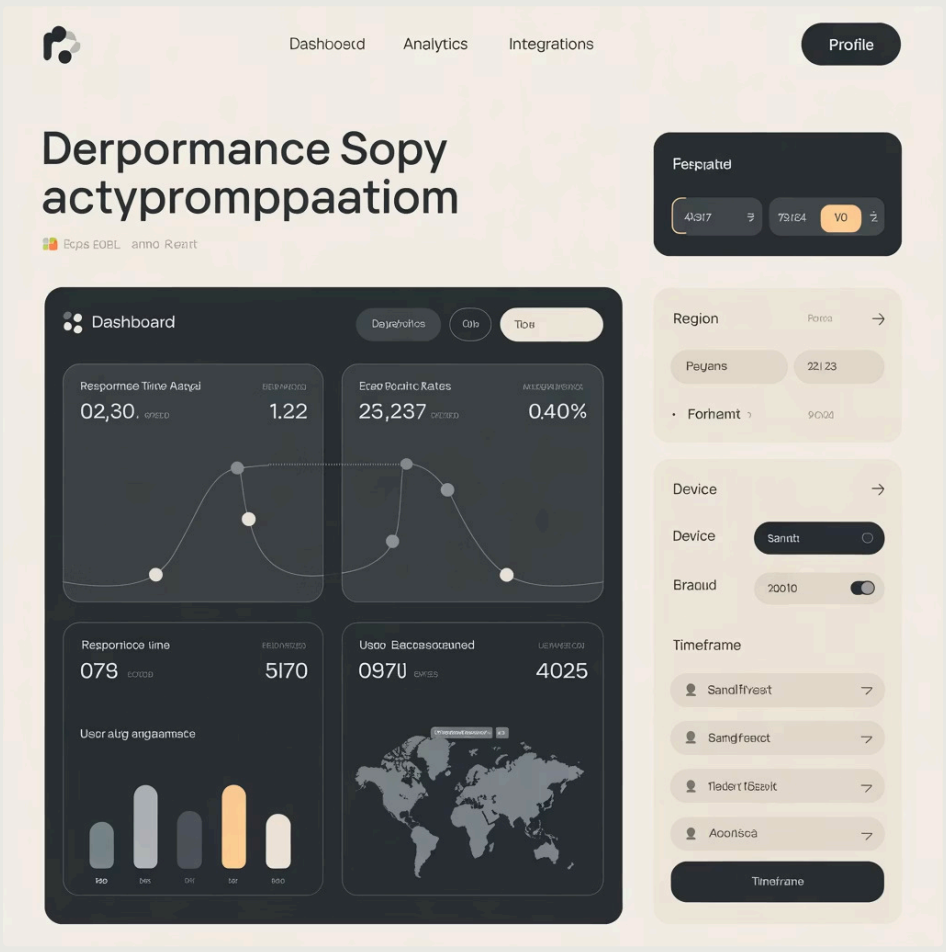
What it is: New Relic's Alerts and Notifications system is your automated alarm bell, designed to inform the right people, at the right time, when critical performance issues or anomalies occur in your application or infrastructure. It moves you from reactive firefighting to proactive problem resolution.

How it works: You define alert conditions based on specific metrics (e.g., "CPU usage > 90% for 5 minutes," "page load time > 3 seconds," or "error rate > 5%"). New Relic also offers anomaly detection, which can alert you to unusual behavior even if it doesn't cross a fixed threshold. When an alert triggers, it sends notifications through various channels, including email, Slack, PagerDuty, webhooks, or custom integrations, ensuring your team is instantly aware.

Why it's crucial: Timely alerts are vital for minimizing downtime and maintaining application health. They enable your team to respond to issues before they escalate or severely impact users. By integrating with collaboration tools like Slack or incident management platforms like PagerDuty, New Relic ensures that your operations team, developers, and on-call personnel can coordinate rapidly and efficiently, drastically reducing the Mean Time To Resolution (MTTR) for any incident.

OPTIMIZING USER EXPERIENCE AND COMPLEX SYSTEMS

New Relic extends beyond basic monitoring with powerful features designed to give you deep insights into actual user experiences and the intricate flow of requests across distributed architectures. These capabilities are crucial for modern applications, especially those built with frameworks like React and Spring Boot microservices.



BROWSER MONITORING (REAL USER MONITORING - RUM)

What it is: Browser Monitoring, also known as Real User Monitoring (RUM), captures performance and error data directly from your actual users' browsers. Unlike synthetic monitoring which simulates user actions, RUM provides insights into the real-world experiences of your users, considering their device, browser, network conditions, and location.

How it works: New Relic injects a small JavaScript snippet into your web pages (e.g., in a React application). This snippet collects a wealth of data including page load times, AJAX request performance, and Core Web Vitals (like Largest Contentful Paint, First Input Delay, and Cumulative Layout Shift). It also tracks all JavaScript errors that occur, providing full stack traces to pinpoint the exact line of code causing issues. Session traces allow you to visualize a single user's journey and interactions through your application.

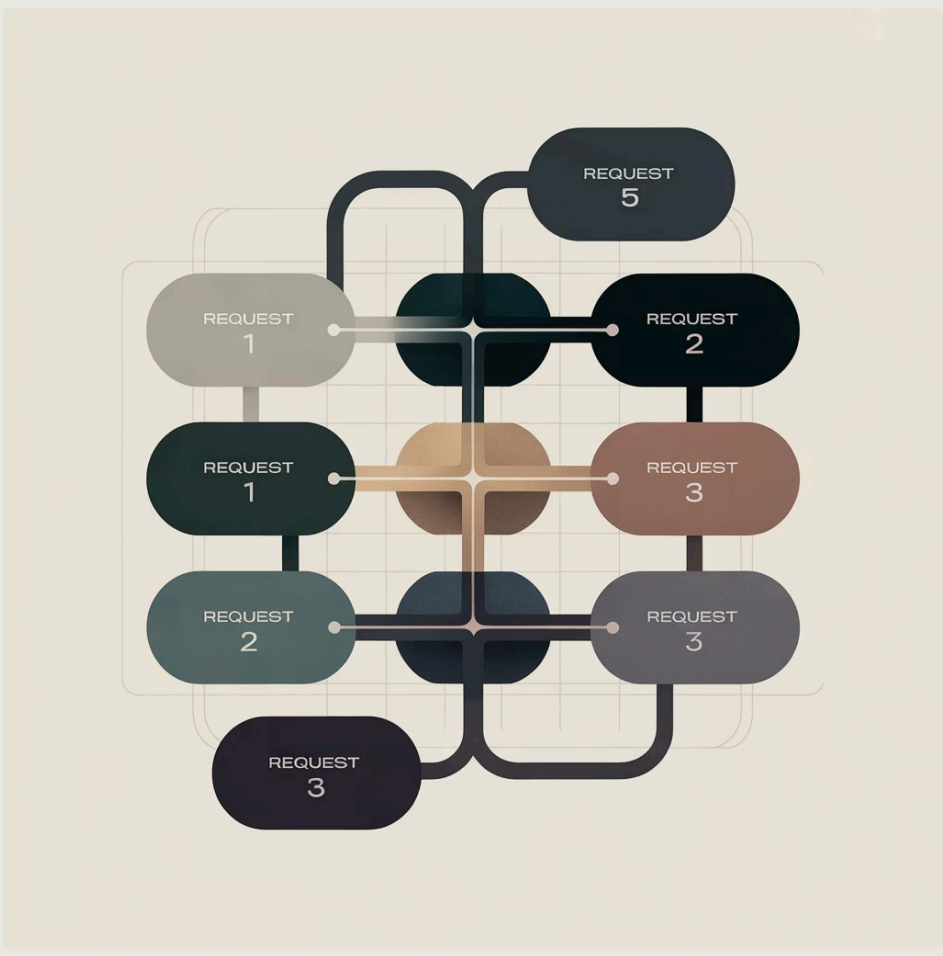
Why it's invaluable: RUM is essential for understanding the actual user experience. For a React app, it can reveal slow component rendering, inefficient data fetching, or JavaScript errors impacting specific user segments. It helps identify performance bottlenecks that synthetic checks might miss, such as issues unique to mobile users or specific geographic regions. This allows you to prioritize fixes that have the greatest impact on user satisfaction and business outcomes.

DISTRIBUTED TRACING

What it is: Distributed Tracing provides an end-to-end view of a single request as it flows through multiple services in a distributed system. In a microservices architecture, a single user action can trigger a cascade of requests across many independent services (e.g., different Spring Boot applications). Tracing helps you follow this complex journey.

How it works: New Relic automatically instruments your services (e.g., Spring Boot microservices) to add unique identifiers to each request. As the request travels from one service to the next, these identifiers are propagated, linking all operations (called 'spans') together into a single 'trace'. The trace visualization then displays a waterfall chart, showing the latency and operations performed by each service, as well as communication between them.

Why it's crucial: For complex Spring Boot microservices, Distributed Tracing is indispensable for debugging and performance optimization. It allows you to quickly identify which service or database call is causing a bottleneck within a multi-service transaction. Instead of guessing, you can see exactly where latency accumulates, enabling rapid diagnosis of issues like slow database queries, inefficient API calls between services, or network delays. This significantly reduces the Mean Time To Resolution (MTTR) for problems in highly distributed environments.



CHAPTER 2: SETTING UP NEW RELIC FOR SPRING BOOT MICROSERVICES

Ready to implement monitoring? Follow these essential steps to get New Relic working with your Spring Boot applications.

CREATE YOUR NEW RELIC ACCOUNT

Navigate to newrelic.com and sign up for a free account. New Relic offers a generous free tier that's perfect for getting started with monitoring your applications.

DOWNLOAD THE JAVA AGENT

The New Relic Java agent is a small program that automatically instruments your Spring Boot application. Download the latest version from your New Relic dashboard.

CONFIGURE YOUR APPLICATION

Integrate the Java agent into your Spring Boot application by modifying your startup command or Dockerfile. The agent will begin collecting performance data immediately.

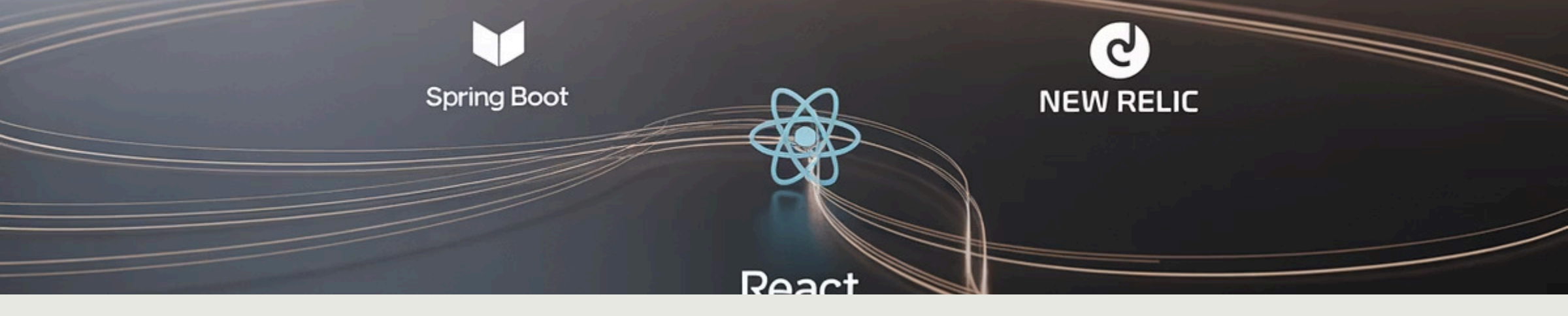
QUICK SETUP OPTIONS

```
# Option 1: Command Line
java -javaagent:newrelic.jar \
  -jar your-spring-boot-app.jar

# Option 2: Docker
FROM openjdk:11
COPY newrelic.jar /app/
COPY your-app.jar /app/
CMD ["java", "-javaagent:/app/newrelic.jar",
     "-jar", "/app/your-app.jar"]
```

Once configured, the New Relic agent automatically begins collecting detailed performance metrics from your Spring Boot application. You'll immediately see data about response times, throughput, database queries, and external service calls in your New Relic dashboard. The agent is designed to have minimal performance impact while providing maximum visibility into your application's behavior.

Pro Tip: The New Relic Java agent works seamlessly with Spring Boot's auto-configuration. No code changes are required - simply add the agent and start monitoring!



IMPLEMENTING DISTRIBUTED TRACING: SPRING BOOT + REACT + NEW RELIC STEP-BY-STEP

Distributed tracing is essential for understanding how requests flow through microservices and how different parts of your application interact. This guide provides a step-by-step approach to setting up distributed tracing using New Relic for Spring Boot microservices and a React frontend.

---

1. PREREQUISITES AND SETUP

Before you begin, ensure you have the following:

- **New Relic Account:** A New Relic account (free tier is sufficient to start).
- **Java Development Kit (JDK):** Version 11 or higher for Spring Boot.
- **Node.js & npm/yarn:** For your React application.
- **Spring Boot Application:** A working Spring Boot microservice.
- **React Application:** A working React frontend that interacts with your Spring Boot backend.
- **New Relic Java Agent:** Download the latest `newrelic.jar` from your New Relic dashboard.

---

2. SPRING BOOT CONFIGURATION

Integrate the New Relic Java agent into your Spring Boot application and configure it for distributed tracing.

ADD NEW RELIC JAVA AGENT

Place the `newrelic.jar` in a known location (e.g., in the root of your application or a dedicated `lib` folder). Configure your application to start with the agent:

```
# Option 1: Command Line
java -javaagent:/path/to/newrelic.jar \
    -Dnewrelic.config.app_name="YourSpringBootService" \
    -
Dnewrelic.config.license_key="YOUR_NEW_RELIC_LICENSE_KEY"
\
    -jar your-spring-boot-app.jar

# Option 2: Dockerfile
FROM openjdk:17-jdk-slim
COPY newrelic.jar /app/newrelic.jar
COPY target/your-spring-boot-app.jar /app/your-spring-boot-
app.jar
WORKDIR /app
CMD ["java", "-javaagent:/app/newrelic.jar", \
    "-Dnewrelic.config.app_name=YourSpringBootService", \
    "-
Dnewrelic.config.license_key=YOUR_NEW_RELIC_LICENSE_KEY",
\
    "-jar", "your-spring-boot-app.jar"]
```

CONFIGURE `NEWRELIC.YML` (OPTIONAL, BUT RECOMMENDED)

For more detailed configuration, create a `newrelic.yml` file in the same directory as `newrelic.jar` or provide its path via `-Dnewrelic.config.file=/path/to/newrelic.yml`. Essential settings include:

```
common: &common
  license_key: 'YOUR_NEW_RELIC_LICENSE_KEY'
  app_name: 'YourSpringBootService'
  distributed_tracing:
    enabled: true
  # ... other configurations
```

SET UP TRACE CORRELATION

New Relic automatically injects and propagates W3C Trace Context headers (`traceparent` and `tracestate`) for supported HTTP clients. Ensure your services pass these headers when making calls to other services. For Spring Boot, this is largely automatic with popular clients like RestTemplate or WebClient, but custom interceptors might be needed for non-standard clients.

---

3. REACT APPLICATION SETUP

Integrate the New Relic Browser agent into your React application to capture frontend performance and correlate it with backend traces.

ADD NEW RELIC BROWSER AGENT

Go to New Relic, navigate to "Browser" > "Add more data", and follow the instructions to get your unique JavaScript snippet. Embed this snippet at the very beginning of your React application's `index.html` file, ideally within the `<head>` tag.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- New Relic Browser Agent Script -->
    <script type="text/javascript">
      (function(a,b,c,d,e,f){...}) // Your New Relic Browser script
    </script>
    <!-- End New Relic Browser Agent Script -->
    <meta charset="utf-8" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.
</noscript>
    <div id="root"></div>
  </body>
</html>
```

CONFIGURE FOR SPA ROUTING

For Single Page Applications (SPAs), you need to tell New Relic when route changes occur to ensure proper monitoring of soft navigations. Use the `newrelic.browser.setCurrentRouteName()` API call for client-side routing:

```
// Example with React Router v6
import { useEffect } from 'react';
import { useLocation } from 'react-router-dom';

function NewRelicSPARouter() {
  const location = useLocation();

  useEffect(() => {
    if (window.newrelic) {
      window.newrelic.setCurrentRouteName(location.pathname);
    }
  }, [location]);

  return null;
}

// In your main App.js or Router component
<Router>
  <NewRelicSPARouter />
  <!-- Your routes -->
</Router>
```

CORRELATE FRONTEND AND BACKEND TRACES

When your React application makes HTTP requests to your Spring Boot backend, the New Relic Browser agent automatically injects trace headers into these requests. The New Relic Java agent on the backend will then pick up these headers, linking the frontend and backend traces into a single, comprehensive distributed trace.

---

4. MICROSERVICES COMMUNICATION

Understand how traces propagate through your microservices architecture.

HOW TRACES FLOW BETWEEN SERVICES

Distributed tracing works by propagating a unique trace ID and span ID across services as a request flows through them. Each service adds its own "span" to the trace, representing its contribution to the overall request. These spans are then linked together by their parent-child relationships.

HTTP HEADERS FOR TRACE PROPAGATION

New Relic supports W3C Trace Context, which uses two primary headers:

- `traceparent`: Contains the trace ID, parent ID (current span ID), version, and flags.
- `tracestate`: Contains vendor-specific information (e.g., New Relic's account, app, and transaction IDs).

When Service A calls Service B, Service A's agent adds these headers to the outgoing request. Service B's agent reads them from the incoming request, creating a child span and continuing the trace.

DATABASE AND EXTERNAL API TRACING

New Relic Java agent automatically instruments common database drivers (e.g., JDBC) and HTTP clients (e.g., OkHttp, Apache HTTP Client) used for external API calls. This means database queries and calls to third-party services will appear as segments within your distributed traces, giving you end-to-end visibility.

---

5. PRACTICAL EXAMPLES

Here are some code snippets for common tracing scenarios.

ACCESSING TRACE CORRELATION IDS (SPRING BOOT)

You might want to log the current trace ID for debugging or custom correlation with other systems.

```
import com.newrelic.api.agent.NewRelic;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

    private static final Logger logger = LoggerFactory.getLogger(MyController.class);

    @GetMapping("/data")
    public String getData() {
        String traceId = NewRelic.getAgent().getTracingMetadata().getTraceId();
        logger.info("Current Trace ID: {}", traceId);
        // ... business logic
        return "Data retrieved, trace ID: " + traceId;
    }
}
```

CUSTOM SPANS/SEGMENTS (SPRING BOOT)

Use custom instrumentation to add specific details to your traces, especially for critical business logic or long-running operations.

```
import com.newrelic.api.agent.Trace; // Use this for method-level tracing
import com.newrelic.api.agent.NewRelic; // Use this for manual segment creation
import org.springframework.stereotype.Service;

@Service
public class MyService {

    @Trace(dispatcher = true) // Marks this method as a transaction or a major segment
    public String processOrder(String orderId) {
        // Automatically creates a segment for this method
        NewRelic.addCustomParameter("orderId", orderId);

        // Manual segment for a specific sub-process
        NewRelic.getAgent().getTracer().segmentFunction(() -> {
            // Simulate a time-consuming operation
            try { Thread.sleep(100); } catch (InterruptedException e) {}
            return "Sub-process complete";
        }, null);

        return "Order " + orderId + " processed.";
    }
}
```

DEBUGGING SCENARIOS WITH NEW RELIC UI

Once traces are flowing, use the New Relic UI:

1. **Distributed Tracing:** Navigate to the "Distributed Tracing" section to see a map of your services and individual traces.
2. **Trace Details:** Click on any trace to see its waterfall view, which visualizes the exact path and timing of the request across all services and components.
3. **Error Tracking:** Correlate errors in your logs with specific traces to quickly pinpoint the root cause.

---

6. BEST PRACTICES

Follow these tips for optimal distributed tracing.

CONSISTENT NAMING CONVENTIONS

Use clear and consistent naming for your applications (`app_name` in `newrelic.yml`) and custom spans. This makes it easier to navigate and understand your traces in the New Relic UI.

MONITOR PERFORMANCE OVERHEAD

While New Relic agents are highly optimized, extensive custom instrumentation can introduce some overhead. Monitor agent CPU and memory usage, and instrument only the most critical parts of your code.

LEVERAGE CUSTOM ATTRIBUTES

Add custom attributes (e.g., `userId`, `transactionId`) to your transactions and spans. These attributes are invaluable for filtering, faceting, and finding specific traces for debugging.

TROUBLESHOOTING TIPS

- **No Traces?** Double-check agent installation, license key, and network connectivity to New Relic endpoints.
- **Missing Spans?** Verify that distributed tracing is enabled in `newrelic.yml` and that W3C headers are propagated correctly.
- **Uncorrelated Traces?** Ensure both frontend and backend agents are installed and configured, and that requests from frontend to backend include the necessary trace headers.

By following this guide, you'll gain deep visibility into your Spring Boot and React applications, enabling faster debugging and a better understanding of your microservices architecture.