New Relic

A Three-Phased Approach to Observability

How to improve the customer experience by moving from reactive to data-driven behaviors

Table of Contents

Introduction

Your customers—regardless if you're in high tech, manufacturing, healthcare, or financial services—expect a seamless experience each time they engage with you. Customer loyalty is precarious at best, and in response, every company is figuring out how to play offense with software. After all, **research shows** that highly engaged customers buy 90% more frequently, spend 60% more per purchase, and have three times more annual value than average customers. That's why you need your infrastructure to scale to your busiest day while continuously delivering delightful, differentiating experiences.

But here's the paradox: How do you support these cutting-edge customer experiences without inadvertently compromising customer experience? The importance of excellent customer experience has been well documented. **Bad customer experiences are more powerful than good customer experiences**. We've all **seen the headlines** about how a few hours of downtime can cost millions in revenue and more in goodwill. The good news: Pushing new code and leveraging modern infrastructure doesn't have to be a high-risk proposition. A well-run **observability practice** enables you to deliver excellent customer experiences with software despite the complexity of the modern digital enterprise.

New Relic defines three phases of observability maturity to chart your path—reactive, proactive, and data-driven. The ultimate goal of any observability practice is an automated, self-healing system that allows your developers to understand the effect of updates on the end user's digital experience in real time and helps your engineers know not just that a failure occurred, but *why*.

This ebook dives into each phase of your observability journey and explains the activities and key performance indicators (KPIs) inherent in each.

A well-run observability practice enables you to deliver excellent customer experiences with software despite the complexity of the modern digital enterprise.

OBSERVABILITY MATURITY:

Reactive Phase

We've all heard the phrase "You can't improve what you can't measure." Accordingly, a strong observability practice starts with putting instrumentation in place to aggregate as much telemetry data about your systems as possible. From there, teams can prioritize eliminating obvious problems.

Reactive Behaviors and Recommendations

- Instrument and Tune
- Establish Baselines and Set Foundational Alerts

KPIs

Errors, Throughput, Latency, Traffic, Saturation, Utilization, Apdex

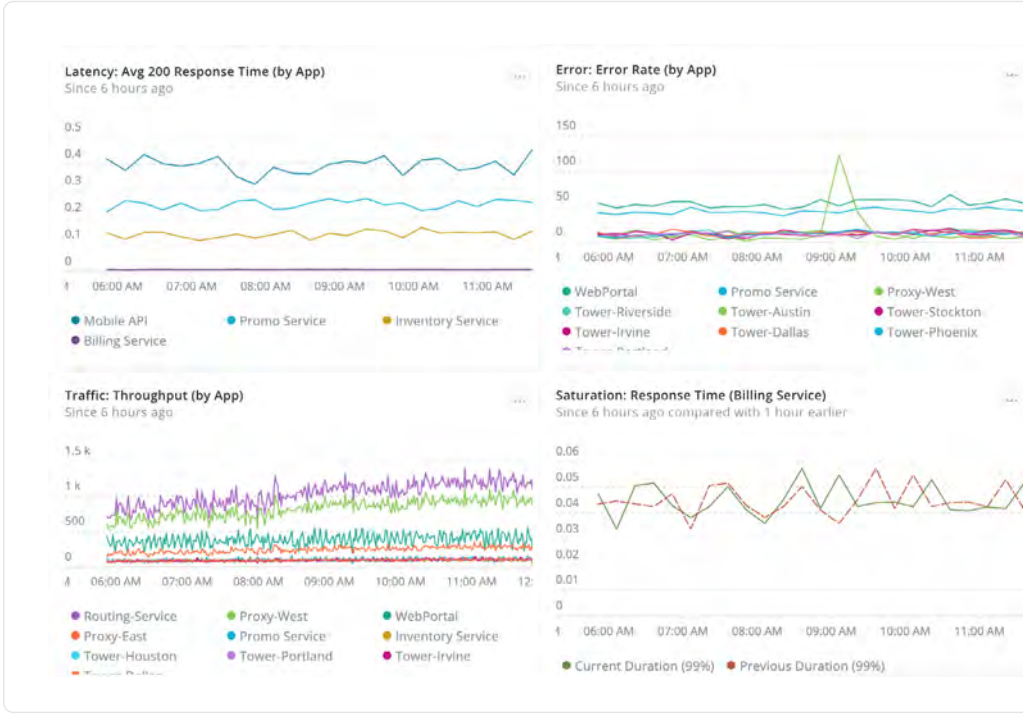
Instrument and tune

When choosing what to monitor, involve business stakeholders and digital leaders from your organization. Your goal is to monitor what matters to your business and not to overload your teams with noise.

Metrics, events, logs, and traces (or **MELT** for short) are the essential data types of observability. Aggregating data from disparate sources, including **logs**, and storing it in one place, gives you the ability to analyze, visualize, and troubleshoot your systems from a single platform. This ability to instrument any data type from any source (proprietary or open source) is critical for observability, as it allows you to get telemetry quickly, no matter the data type, and resolve issues dynamically as they arise.

With all your business-critical observability data in one place, you can reduce the operational burden—and expense—of maintaining other systems for storing, querying, viewing, and alerting on that data.

This ability to instrument any data type from any source (proprietary or open source) is critical for observability.



The Four Golden Signals (as laid out in the Google SRE Handbook) —a useful starting point as you baseline your applications.

Establish baselines and set fundamental alerts

Gathering metrics and performance statistics about your application and infrastructure helps you identify bottlenecks and errors that can lead to instability in your software (and frustration for your users). **Remediate your apps** by addressing your slowest transactions and the most common errors. Then you can establish a baseline of what "normal" is for your services and systems. Analyze obvious errors to create **baseline alert conditions** and adjust sensitivity as needed. Mapping service boundary transaction performance to product features is critical to breaking silos between product, operations, engineering, and service owner teams.

OBSERVABILITY MATURITY:

Proactive Phase

With so many different signals coming from multiple tools, it's difficult to determine and address the root cause of incidents quickly, let alone detect and respond to issues proactively. Increased complexity in your tech and processes leads to the challenge of investigating and interpreting thousands of "unknown unknowns." This stage aligns teams by setting SLOs and optimizing your processes to balance system reliability with speed and address issues before they affect your customers.

Proactive Behaviors and Recommendations

- Set SLOs
- Optimize Alerting Strategy
- Improve Incident Response
- Enable Fast Feedback Loops

KPIs

Availability, Errors, Latency, Traffic, Saturation, Utilization, Deploy Frequency, Lead Time for Changes, MTTD, MTTR, MTBF, Page Load Time

Set SLOs to align teams

Service level objectives (SLOs) provide a powerful mechanism to codify a team's goals in ways that can be measured and shared. They also provide clear boundaries on service expectations that help teams achieve greater velocity and freedom in experimenting with new approaches. If you're unsure where to start, the **four golden signals** (latency, traffic, errors, and saturation) as laid out in **Google's SRE handbook** can help expose issues. Then, find one or two high-level service level indicators (SLIs) representing the overall health of your services. For example, video-streaming services use time to first frame, the time between a user selecting a movie and when the first frame appears.

Optimize your alerting strategy

With data aggregation in place and team objectives clearly defined, you'll need a systematic and reliable way to mobilize on-call teams when things go wrong. Well-defined alerts help you understand your systems' health so that you can respond to performance problems quickly.

Not every SLO needs to become an alert. A strong alert strategy takes SLOs and creates a set of simple, actionable alerts. Many mature organizations set fewer alerts in general and focus those alerts on a core set of metrics that indicate when their customer experience is truly degraded. DevOps teams often use **Apdex** as part of their alerting strategy to align alerts with signs of degraded user satisfaction.

Not every SLO needs to become an alert. A strong alert strategy takes SLOs and creates a set of simple, actionable alerts.

Alert Basics

As you design your alert strategy, keep this question in mind: "If the customer isn't impacted, is it worth waking someone up?"

Question	Metrics and KPIs
Are we open for business?	Set up automated pings and alert on availability.
How's our underlying infrastructure?	Manage and troubleshoot your hosts and containers.
How's the health of our application?	Use real end-user metrics to understand the backend. Use metric and trace data from open source tools, and display that information alongside all the other systems and services data you're managing.
How do I troubleshoot a system error?	Search and investigate the root cause of an issue across your application and infrastructure logs.
How's the overall quality of our application?	Use an Apdex score to quickly assess an application's quality.
How are our customers doing?	Monitor front-end and mobile user experiences.
How's our overall business doing?	Focus on **key transactions** within an application and tie them to expected business outcomes to correlate application and business performance.

Improve incident response

Every minute you spend manually interpreting your telemetry data to diagnose and resolve problems affects your company's reputation and bottom line. That's why you need to empower your IT operations teams with intelligence and automation that proactively detects and resolves incidents faster.

Assign responsibility for the health of the application and services to a person or team. They will be the first responder for an incident. Make sure that communications during critical incidents take place in easily accessible and highly visible channels. For instance, if your teams are used to communicating with a tool such as Slack, make sure that incident alerts can be sent to Slack.

Leverage deployment markers to quickly understand how code changes impact reliability.

Enable fast feedback loops

Fast feedback loops for releases reduce mean time to resolution (MTTR) and reliability risks associated with frequent application updates. It's important to track deployments and how the impact of code and infrastructure changes affect customer experience.

Tracking deployments is a valuable way to determine the root cause of immediate, long-term, or gradual degradations in your application. Fast feedback loops are also critical for modern release practices such as canary or **blue/green** deploys.

Fast feedback loops for releases reduce mean time to resolution (MTTR) and reliability risks associated with frequent application updates.

OBSERVABILITY MATURITY:

Data-Driven Phase

Ultimately, the goal of observability is to enable teams to take action with shared data. Connect people with processes and technology performance across the entire organization and tie it to specific business outcomes. To keep your observability data flowing, automate instrumentation for new apps and services. Then, you'll be able to move fast with confidence as a data-driven organization.

Data-Driven Behaviors and Recommendations

- SLO Trend Analysis
- Visualize the Relationship Between Dev, Ops, and Customer
- Automate Instrumentation

KPIs

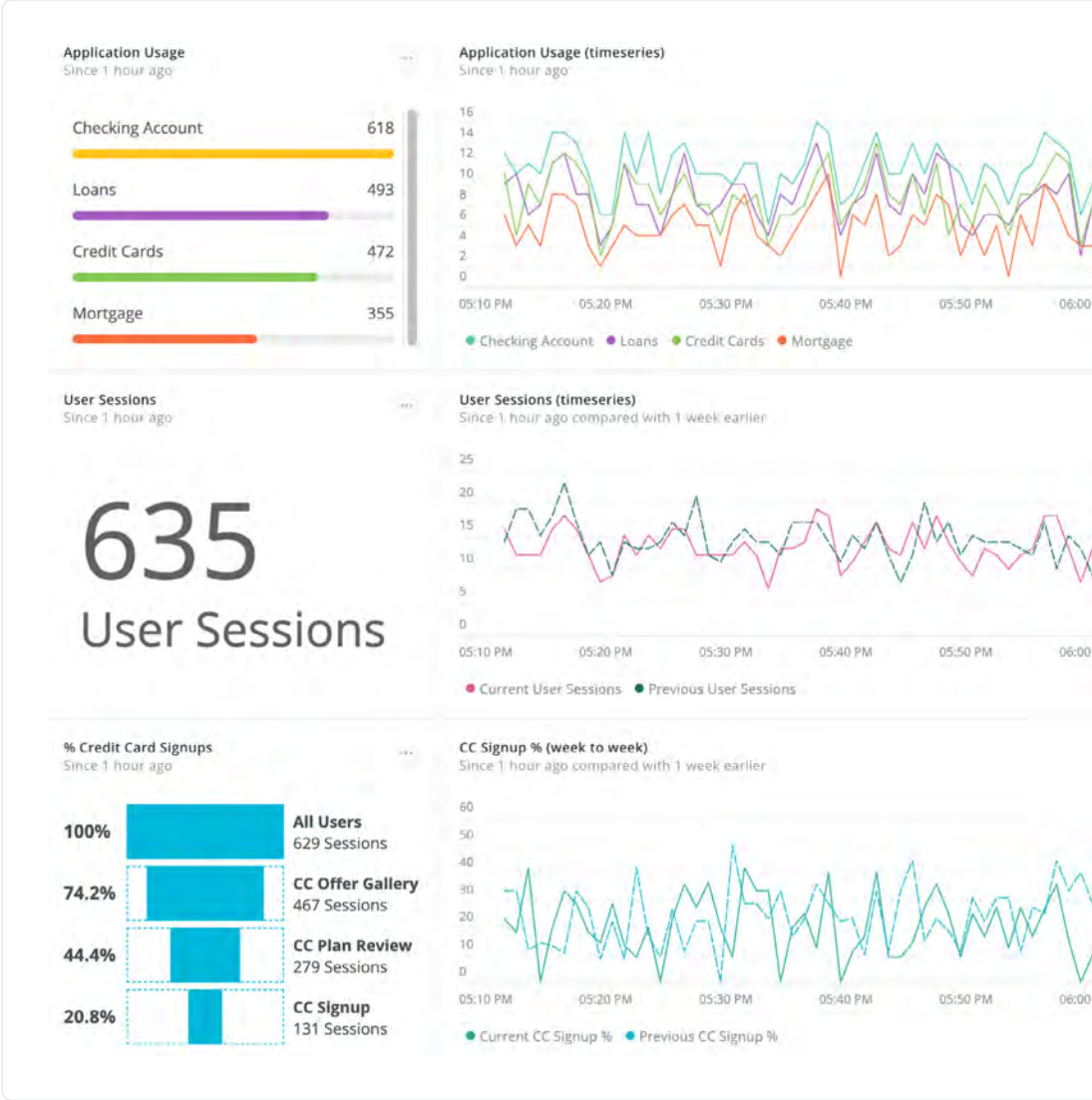
Number and MTTR of Change-Related Incidents, Developer Satisfaction, Customer Satisfaction, SLO and SLA Compliance, Utilization, Error Rate, Active Alerts, Number of Support Tickets

SLO trend analysis

Defining and measuring SLOs allows you to decompose your software into a set of key indicators that you can use to validate the expected behavior of services and track how incidents affect service delivery. Leverage the work you've done to define SLOs (above in the proactive phase) to find hot spots that need focus. Team dashboards are a good reference point for describing SLO attainment over time.

Visualize the relationships among developers, operations, and customers

A curated business performance dashboard will give your teams an overview of how users are experiencing your app. Knowing how your customers use—or abandon—your apps will help you triage future work. A strong observability culture democratizes data beyond the typical backend users, making it available to groups such as customer service, support, product, sales, marketing, and business executives. Having this data front and center are powerful motivators for teams.



Connect the dots to understand how application and infrastructure performance impact business outcomes.

Automate instrumentation

Automation can make you faster, and it helps to eliminate fat-finger errors—and more importantly, it improves observability. You'll spend less time instrumenting systems and reduce toil as your application and infrastructure stacks inevitably grow. To instill this discipline, consider putting a 40% cap on the aggregate "ops" work for all SREs, such as tickets, on-call, and manual tasks. An SRE team must spend the remaining time developing apps. Over time, your SRE team should end up with very little operational load and almost entirely engage in development tasks, because the service runs and repairs itself. Leverage observability strategies that enable your teams to minimize toil and maximize automation.

10

Observability Best Practices

1

Collect Data

Aggregate as much telemetry data about your systems as possible—metrics, events, logs, and traces, whether open source or proprietary—and store that data in a single platform.

GOAL

Collect and explore all of your telemetry data in a unified platform.

2

Establish Baselines

Analyze obvious errors to create baseline alert conditions and adjust sensitivity as needed.

GOAL

Put out obvious fires and build a foundation to prioritize action.

3

Create SLOs

Set SLOs, perhaps starting with the four golden signals of latency, traffic, errors, and saturation and a few relevant metrics that measure end-user experiences.

GOAL

Codify team goals in a way that can be measured.

4

Set Alerts

Define a simple alert strategy that prioritizes a core set of metrics indicating the customer experience is degraded.

GOAL

Focus on alerts for the most critical SLOs.

5

Speed Up Incident Response

Empower your IT operations teams with intelligence and automation that proactively detect and resolve incidents faster.

GOAL

Optimize the feedback loop between issue detection and resolution.

6

Track Deployments

Monitor application deployments so you can assess the impact of code changes on performance.

GOAL

Determine the root cause of immediate, long-term, or gradual degradations in your application.

7

Automate

Keep telemetry data flowing by automating instrumentation for new apps and services.

GOAL

With shared telemetry data, your teams can move fast with confidence.

8

Analyze

Create dashboards to measure SLO attainment over time.

GOAL

Validate service delivery expectations and find hot spots that need focus.

9

Be Customer-Centric

Give your teams an overview of how customers use—or abandon—your apps.

GOAL

Improve how you triage and prioritize future work.

10

Minimize Toil

Instill the discipline of automation by capping SRE "maintenance work" at 40%.

GOAL

SRE teams end up with very little operational load and engage in development tasks because the service largely runs and repairs itself.

Observability in New Relic

Now that you've learned about the three phases of observability maturity, here are some hands-on tips for getting started in New Relic.

Reactive phase

- Install the appropriate monitoring agents for applications, the infrastructure that supports them, and the frontend user experiences they enable.
- Collect data from popular **cloud-based services**, including Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, and **critical telemetry data from any source**, including OpenTelemetry, Envoy, Dropwizard, Prometheus, and more.
- Access **on-host integrations** through the Infrastructure agent.
- Establish **key transactions** in New Relic APM, giving you a custom level of monitoring, or unique **alert policies.**

Proactive phase

- **Set up New Relic Synthetics** to interact with the entire system as an external user would, giving your teams high-level checks for performance, availability, and user experience.
- Implement **New Relic AI** to streamline and enhance your production incidents.
- Use the **New Relic APM** deployment markers feature to **record deployments** for each application. **Deployment markers** in team dashboards enable you to analyze the before-and-after impact that software releases have on performance.

Data-driven phase

- Take advantage of the **SLO/R application** in New Relic One to help define SLIs and set SLO targets. Consider reporting the SLO attainment. Your teams should use how you're tracking against your SLO target as an indicator of when you need to slow down to focus on reliability work and push on new features. You can also use SLO trends to facilitate better organizational decision making.
- Use this **tutorial** to learn how your customers experience your apps **across different cohorts**, such as by geographic location or device type. Segment and assess end-user cohorts and ensure you are optimizing customer experience in-line with overall business strategy.
- Use **New Relic One dashboards** to build flexible, interactive visualizations from data anywhere in the New Relic One platform.
- Use the **Synthetics REST API** to create and manage New Relic Synthetics monitors.
- Configure alert settings and **set up alert conditions**.
- **Write a build script or plugin** that captures dependencies and tasks to complete the agent install and provide them with sensible default configurations.