International Journal of Computing and Artificial Intelligence

E-ISSN: 2707-658X
P-ISSN: 2707-6571
www.computersciencejournals.com/ijcai
IJCAI 2025; 6(2): 60-63
Received: 12-05-2025
Accepted: 17-06-2025

Topalidi Anna
Specialist Degree, Moscow State University of Geodesy and Cartography, Moscow, Russia

Corresponding Author:
Topalidi Anna
Specialist degree, Moscow State University of Geodesy and Cartography, Moscow, Russia

Monitoring and visualization of web application performance metrics: Using Prometheus, Grafana, and new relic in ruby projects

Topalidi Anna

DOI: https://www.doi.org/10.33545/27076571.2025.v6.i2a.177

Abstract
This article examines the practice of monitoring performance metrics in web applications developed using the Ruby programming language. It explores key approaches to organizing the collection, visualization, and interpretation of metrics with the goal of enhancing system stability and observability. Special attention is given to the tools Prometheus, Grafana, and New Relic, their architectural features, integration capabilities with Ruby applications, and differences in data representation models. The study also investigates alerting configuration scenarios and event handling as the final stages in building an effective monitoring system. Methods for alert definition, notification delivery mechanisms, and approaches to automating anomaly response are analyzed.

Keywords: Monitoring, performance metrics, visualization, alerting, event handling, Prometheus, Grafana, New Relic, Ruby

Introduction
In a situation that is dominated by high competition and rapidly growing user expectations, it is imperative that web applications deliver consistent and up to standard performance. Even minimal response time delays can lead to reduced client satisfaction and cause financial losses for organizations. To tackle these problems, it is important to monitor key performance indicators continuously, evaluate system functionality, and take immediate corrective actions upon observing any anomalies. In this regard, active performance monitoring has become a key determinant in the maintenance and development of current web services.

In practice, monitoring encompasses not only the collection and storage of metrics, but also their visualization, alert configuration, and automated event handling. Within the Ruby development ecosystem, tools such as Prometheus, Grafana, and New Relic are increasingly adopted. Each provides distinct approaches to application observability, scalable monitoring, and accessible presentation of performance data. When implemented appropriately, these tools enable DevOps teams and developers to identify bottlenecks, anticipate load patterns, and ensure reliable system operation around the clock. The goal of this research is to analyze the approaches to monitoring performance metrics in web applications using Prometheus, Grafana, and New Relic, with a particular focus on data visualization, alert configuration, and event handling in Ruby-based projects.

Main part. Approaches to monitoring web application performance metrics
The maintenance of peak performance requires a wide approach to the gathering, analysis, and interpretation of various data. Functionality monitoring is an essential foundation for guiding technical decision-making, detecting real-time anomalies, and planning for future scalability. The assessment of an application includes not just its inherent attributes, but also extrinsic metrics related to user interaction and the general health of the underlying infrastructure. System-level metrics reflect the state of the host environment and include CPU utilization, memory consumption, number of open file descriptors, and network throughput (table 1).

Table 1: Common performance metrics in web applications [1-3]

Metric	Description	Example threshold	Equivalent metric in New Relic
http_request_duration	Time taken to process an HTTP request	≤ 200 ms	Response time
http_requests_total	Total number of HTTP requests received	Depends on traffic	Throughput (requests per minute)
memory_usage_bytes	Amount of memory used by the application	≤ 70% of available	Memory usage
error_rate	Proportion of requests resulting in errors	< 1%	Error rate
db_query_duration	Average duration of database queries	≤ 100 ms	Database query time

Within the Ruby ecosystem, the Prometheus exporter tool plays a significant role by enabling the export of application metrics in a format compatible with modern monitoring systems. When using the Rails framework, it is common practice to collect custom metrics directly from middleware components or background workers. Examples include measuring view rendering times, SQL query durations, or the behavior of background job processors. In large-scale applications, distributed tracing systems are often employed alongside metrics to visualize the relationships between requests, which is particularly valuable in microservice architectures.

A core aspect of any monitoring system is its metric collection strategy. The two most widely adopted models are push and pull. In the push model, agents embedded within the application or host actively send metrics to an external monitoring service. A notable example is New Relic, where the Ruby agent aggregates and transmits performance data to a cloud-based platform. In contrast, the pull model involves the monitoring system, such as Prometheus, periodically querying predefined endpoints to retrieve the latest metrics [4]. Each approach has distinct advantages: push is generally easier to integrate into cloud and hybrid environments, while pull offers greater transparency and control in self-managed or on-premise deployments. A typical monitoring architecture for Ruby-based applications is adapted from publicly available Prometheus-related design patterns (fig. 1).

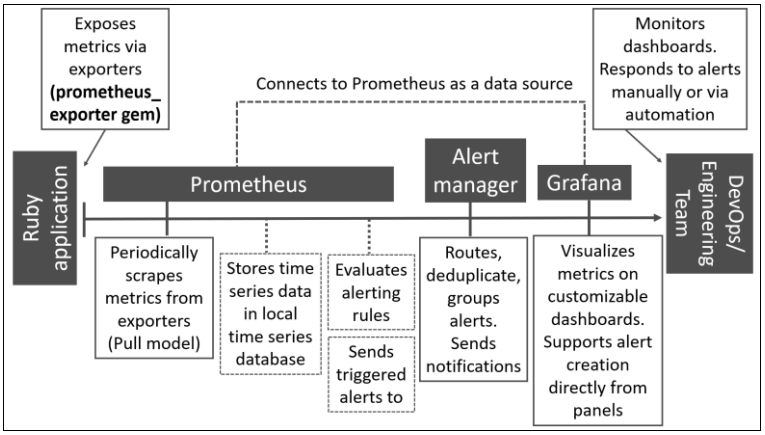

Fig 1: Monitoring architecture for a Ruby application [5]

Effective monitoring is not possible without a clear understanding of threshold values for each metric. Absolute metric values alone do not always provide sufficient insight. It is crucial to consider trends over time as well as the interrelationships among different metrics. An increase in latency, for instance, might be attributed to a surge in user traffic, degradation of an external API, or internal memory leaks. The primary function of monitoring is to identify such anomalies early and guide the diagnostic process.

An essential component of modern monitoring systems is alert configuration; however, detailed implementations of alerting and event handling are discussed in a separate chapter. It is worth noting here that a robust monitoring system must support the definition of flexible alerting conditions and maintain a history of observations for retrospective analysis.

Performance monitoring of web applications emerges as a multilayered process that encompasses data collection, metric interpretation, and alignment with business logic. In Ruby-based systems, this requires a well-designed monitoring architecture and flexible instrumentation strategy that captures both application behavior and the state of the surrounding infrastructure.

Monitoring and visualization tools: Prometheus, Grafana, and New Relic

Modern monitoring tools for web products go far beyond simple metric gathering, with full data visualization capabilities and analytical processes. The choice of a supervision tool depends on several factors, along with the size of the project, the type of infrastructure, and the level of analysis needed. Sought-after solutions in Ruby applications are Grafana, Prometheus, and New Relic. These tools have different approaches to data handling and operate at different levels of abstraction.

Grafana does more than the typical role of being just another metric aggregation dashboard; it is a powerful visualization tool with the ability to connect with many data sources, such as but not restricted to, Prometheus, InfluxDB, and Elasticsearch. The platform allows one to build dashboards with graphs, heatmaps, numerical panels, and time-series displays, all programmatically customizable to accommodate specific monitoring needs (fig. 2).

Fig 2: Example of a Grafana dashboard querying Prometheus metrics [6]

For Ruby-based applications, Grafana's ability to segment visualizations by layer: from basic request statistics to the behavior of background jobs and microservice components, is particularly valuable. Dashboards in Grafana can be configured to highlight anomalies, resource utilization levels, and temporary spikes in user activity. Besides its visualization capabilities, Grafana supports alert creation directly from graph panels.

Prometheus is a high-performance time-series database designed for autonomy and independence from external storage systems. It was developed with automation in mind and operates using a pull-based model, where metrics are collected by querying predefined endpoints. One of its primary advantages is full data transparency: all stored metrics are queryable using PromQL, a powerful and flexible query language that supports aggregation, filtering, and correlation [7].

New Relic is a cloud-based observability platform that prioritizes ease of use and rapid deployment. Unlike Prometheus and Grafana, which require local infrastructure setup and ongoing maintenance, New Relic is delivered as a Software-as-a-service solution and integrates with applications through the installation of a Ruby agent. This agent automatically collects key metrics, transaction traces, dependency data, and SQL performance information.

Visualization in New Relic is designed with practical use cases in mind; the interface provides preconfigured dashboards that display latency, error rates, throughput, and system load, with intuitive filtering by time window, service, or environment. Another benefit is the availability of tracing and performance monitoring capabilities inherent to the method, allowing developers to identify slow or problematic areas of code without the need for manual instrumentation or the setup of exporters.

Alert configuration and event handling: implementation of practical scenarios

The production of a strong monitoring system requires more than just the collection and display of data. To enable timely responses to anomalies and reduce the effect of system failure on end users, it is essential to include thorough alerting capabilities in addition to clearly defined event-management procedures. Notifying capabilities are the essential bridge between passive system monitoring and active problem resolution. This process converts raw numerical data into useful notifications that signal drops in performance, operational anomalies, or violations of expected system behavior. Notably, a well-designed alerting system must have the ability to differentiate between important events and minor fluctuations, thus avoiding warning fatigue and the deluge of useless information.

The development of alerting rules typically begins with defining thresholds and trigger conditions. These thresholds are not always static; in many cases, dynamic models that compare current values with historical baselines or moving averages are preferable. A sudden increase in response time under normal load conditions may be a more significant indicator of a problem than simply exceeding a predefined limit. Contextual awareness is also essential, since acceptable metric values can vary depending on time of day, environment type (production vs. staging), or expected seasonal traffic. As a result, mature monitoring systems often rely on relative changes and composite conditions involving multiple metrics

Alerts are defined in the context of Prometheus by the PromQL query language, which defines a certain level of configuration. Alerts involve an expression, the specified evaluation time, and a set of annotation items that provide context for the observed condition. Once an alert is triggered, it is forwarded to Alertmanager, which handles notification routing and suppression logic. Alertmanager supports grouping similar alerts, reducing redundancy in notifications, suppressing lower-priority alerts in the presence of more critical ones, and defining silence periods during which notifications are withheld [8]. These features are particularly valuable during planned maintenance windows or nighttime hours, when certain anomalies may be expected and should not generate unnecessary alerts.

Response strategies to alerts warrant particular attention. These may be manual, automated, or hybrid in nature. In manual response scenarios, notifications are delivered to designated personnel who then assess the situation and take

corrective action. This procedure is suitable for mission-critical systems where minimizing false positives is a priority. Automated responses are appropriate when predefined remediation workflows can be executed. This can include restarting a service, switching to a standby node, scaling out background workers, or triggering a recovery script. Such actions are especially effective in scalable environments like Kubernetes, where infrastructure can be managed declaratively and linked to the alerting system via API-driven automation.

Implementing a complete observability pipeline, ranging from metric collection to alerting, event generation, and response, substantially increases the resilience of web applications alleviates operational burden on engineering teams. However, it is important to recognize that alerting is not a static system. It requires continuous refinement, rule revision, and adaptation to evolving operational conditions. Upgrades to application libraries, changes in system architecture, or shifts in user behavior can render previously defined thresholds obsolete and trigger a surge of false positives. As a result, effective alert management necessitates regular audits and close alignment with both development and operations workflows.

In this light, alert configuration and the design of event response strategies are integral to any comprehensive monitoring system. They transform passive observation into active reliability engineering, enabling teams to respond swiftly and appropriately to emerging threats. In the context of Ruby-based projects, the use of tools such as Alertmanager, Grafana Alerting, and New Relic Policies facilitates the customization of monitoring strategies to suit real-world operational environments and ensures visibility and control across all layers of the application architecture.

Conclusion

The measurement of performance metrics is a fundamental element in delivering the stability and predictability of web applications. Due to mounting system loads and architectural complexities, it is of paramount importance to correctly acquire and store data. It is also significant to perform meaningful analysis and effectively visualize the data. Grafana, Prometheus and New Relic represent different approaches to observability, ranging from highly customizable, infrastructure-level solutions to efficient, automated solutions with ease of setup as the prime focus. The choice of the particular tool would largely depend on the project's architectural design, the team's proficiency, and the goals the system would need to meet within the firm.

A good monitoring system consists of multiple interdependent elements: systematically structured metric retrieval, advanced data visualization, accurate alerting configuration, and automated remedial actions. Together, these elements support not just passive observation, but active engagement with the system, enabling teams to reduce risk and maintain high levels of reliability. In practice, the integration of monitoring into day-to-day development and operations significantly enhances the quality of technical decision-making and accelerates management response in the face of incidents.

References

1. Sidorov D, Kuznetcov I, Dudak A. Asynchronous programming for improving web application performance. ISJ Theoretical & Applied Science. 2024;138(10):197-201. DOI: 10.15863/tas.2024.10.138.23. EDN: VUUJRS.
2. Instrumentation. Prometheus. https://prometheus.io/docs/practices/instrumentation/ (accessed: 12.05.2025).
3. Thresholds. Grafana. https://grafana.com/docs/k6/latest/using-k6/thresholds/ (accessed: 13.05.2025).
4. Singh J, Ghai K. Comparing New Relic with other Performance Monitoring Tools. 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions). IEEE. 2022:1-5. DOI: 10.1109/ICRITO56286.2022.9964706.
5. Overview. Prometheus. https://prometheus.io/docs/introduction/overview/ (accessed: 16.05.2025).
6. Grafana support for Prometheus. Prometheus. Available from: https://prometheus.io/docs/visualization/grafana/ (accessed: 18.05.2025).
7. Aung T, Zaw HT, Maw AH, Mon MT. Comprehensive Analysis: Monitoring Apache Kafka with Grafana, JMX Exporter, and Prometheus. 2024 5th International Conference on Advanced Information Technologies. IEEE. 2024:1-6. DOI: 10.1109/ICAIT65209.2024.10754944.
8. Yuan C, Zhang W, Ma T, Yue M, Wang PP. Design and implementation of accelerator control monitoring system. Nuclear Science and Techniques. 2023;34(56). DOI: 10.1007/s41365-023-01209-z. EDN: UMNOII.