

Application Monitoring using Prometheus and Grafana

[¹] Abhijeet Dhane, [²] Atharva Joshi, [³] Chinmay Borgaonkar, [⁴] Manas Deshpande, [⁵] Prof. Pravin Patil

[¹] [²] [³] [⁴] [⁵] Department of Computer Engineering Pune Institute of Computer Technology, Pune 411043, India

Corresponding Author Email: [¹] dhaneabhijeet30@gmail.com, [²] atharvaj06@gmail.com, [³] cpborgaonkar@gmail.com, [⁴] manasdeshpande4902@gmail.com, [⁵] prpatil@pict.edu@gmail.com

Abstract— In today's fast-paced digital world, ensuring the smooth operation of critical applications has become more important than ever before. With increase in user expectations and ever-increasing competition, even minor downtime or performance issues can lead to significant losses in revenue, reputation and customer trust. To prevent such outcomes, organizations must proactively monitor their applications and detect potential issues before they impact users. Application monitoring involves continuously tracking various metrics associated with application's performance and availability. Traditional monitoring approaches mainly include relying on manually checking the system and periodic reporting which can be time consuming, error-prone and reactive. In contrast, modern monitoring tools leverage automation, machine learning, and real-time analytics to enable faster detection, diagnosis, and resolution of issues. Prometheus and Grafana are two popular open-source tools that have gained widespread adoption in recent years for application monitoring purposes. Prometheus is a powerful metric collection and query engine that enables organizations to capture granular data about application behavior and infrastructure health. Grafana, on the other hand, is a flexible visualization platform that allows users to create custom dashboards, alerts, and reports based on Prometheus (and other) data sources.

Index Terms— Prometheus, Grafana, application monitoring, time series database, visualization tool.

I. INTRODUCTION

An emerging paradigm in enterprise computing is cloud computing. Cloud computing platforms exchange large amounts of data storage, software, and infrastructure to create a sizable resource pool from which customers can draw information services, storage capacity, and processing power as needed. It is necessary to keep an eye on your servers' resources, performance, and problems in addition to the applications they support. Also these modern applications are complex and constantly evolving, making continuous monitoring is essential for efficient operation. Traditional monitoring solutions often rely on heavyweight infrastructure and are difficult to scale. Applications nowadays are becoming more dispersed, dynamic, and complicated. This makes it challenging to follow them. Among the difficulties in keeping an eye on contemporary applications are:

- **Containers:** Applications that are containerized can be dynamically scaled up or down. Metric collection and analysis from containerized applications may become challenging as a result.
- **Cloud-native apps:** These programs are frequently set up across several cloud environments. Because of this, it could be challenging to obtain a complete picture of the functionality and state of cloud-native apps.
- **Microservices:** Applications built using microservices are made up of numerous tiny, independent services. Because of this, it could be challenging to monitor dependencies between services and find the source of issues.

II. BACKGROUND AND RELATED WORKS

In the realm of extreme-scale systems, monitoring and managing these colossal infrastructures pose significant challenges for organizations. Despite the availability of numerous monitoring platforms and analytic tools, the sheer volume and complexity of data generated necessitate substantial efforts from computational center staff or users to analyze results comprehensively. The escalating rates of data collection are stretching the limits of current monitoring infrastructures, highlighting the critical need for careful consideration and design of monitoring solutions in the procurement of extreme-scale systems. Factors such as scalability, high availability, automation, and the integration of dashboards for streamlined operations are emphasized to ensure efficient monitoring and management, anticipating issues and facilitating rapid response.

Efficient monitoring solutions for extreme-scale systems must address several key design considerations. These include scalability and high availability to accommodate the growing demands of computational ecosystems, along with automation to streamline data processing and service orchestration while minimizing complexity for users. Dashboards offering comprehensive views of system health and performance metrics are essential for proactive issue anticipation and rapid troubleshooting. Additionally, actionable insights and analytics, coupled with the ability to customize monitoring solutions to suit specific organizational needs, are crucial for effective management of hybrid computational deployments across both on-site and cloud environments.

A. Existing Solutions and Technologies

Servers were previously observed with the use of the analytics repository Cloudwatch[3]. One-day monitoring periods,

a monthly cap of ten alarms, and a maximum of five actions per alarm were among Cloudwatch's limitations. Moreover, it was not able to export alerting warning data for further research. The user interface (UI) was disorganized because there were many screens to navigate through. The reason it couldn't keep an eye on servers was that it didn't provide any recommendations.

In the study, machine learning is suggested as a method of server monitoring [4]. To classify the server status as healthy or sick, it uses a revolutionary K Nearest Neighbor (KNN) machine learning algorithm based on some physical variables as system memory, RAM, and swap tool. The model is additionally trained using a training dataset. For analyzing the results as a bar graph, Nvd3.js is utilized.

While traditional systems are useful for managers, users, and system administrators because they can provide meaningful performance statistics via the command line, they don't provide a thorough visualisation of system resource usage trends, according to the study "A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster"[7]. Using Slurm to obtain job states, Telegraf to gather CPU metrics, InfluxDB to store time series data, and Grafana to visualize and obtain insights into system resource usage and alerts via widely used industry accepted communication channels, the authors proposed a method for gathering system state and visualizing it. The technique offered centers on the effective elimination of system task data, in combination with additional performance metrics and associated perceptible presentation.

A comprehensive analysis of the effects of utilizing Prometheus and Grafana for HPC systems with CPUs and GPUs in particular has been published in "Jobstats: A Slurm-Compatible Job Monitoring Platform for CPU and GPU Clusters."[8] The complexity of HPC clusters increases with file system slowdowns, CPU overloads, and job failure rates becoming more difficult to handle in these systems. In order to monitor CPU and memory consumption, GPU job statistics, and Node Specific Statistics, four Prometheus exporters are further installed on the Jobstats job monitoring platform, which is then viewed using Grafana. Slurm stores job statistics on an individual basis. The Jobstats monitoring platform, the tools created on it, and the speed at which the time series database processes data are the main contributions of this work.

III. METHODOLOGY

A. Introduction to Prometheus

Prometheus is an open source toolkit for system monitoring and awakening that is utilized as free software for

event monitoring and waking. Either directly or through a central drive gateway for temporary work, Prometheus extracts criteria from instrumented jobs. It keeps all of the scraped samples locally and applies rules to this data in order to generate warnings or total and record new time series from the data. Prometheus' primary characteristics are

- A multidimensional data model that uses key/value pairs and metrics names to link time series data.
- One versatile query language to take advantage of this dimension is PromQL.
- Individual nodes are independent; there is no dependency on a distributed storage..
- A pull model over HTTP is used to collect time series

B. Introduction to Grafana

Users can view their data through charts and graphs that are combined into a single dashboard (or multiple dashboards!) for simpler interpretation and comprehension using Grafana, an open source interactive data visualization platform created by Grafana Labs. Grafana was founded on the ideas of openness and the notion that information ought to be accessible to everyone in the company, not just a select few. As a result, teams are encouraged to be more transparent, creative, and cooperative by creating an environment where data is easily accessible and utilized by anyone who needs it.

Key properties:

- Dashboards: use graphs, heatmaps, geomaps, histograms, and other visualizations to see your data however you'd like.
- Plug-ins: Connect to current data sources with dashboard plug-ins to render your data in real-time on an intuitive API; no data migration is necessary. Additionally, plugins for data sources can be made to retrieve metrics from any custom API.
- Alerts: All of your alerts can be created, consolidated, and managed from a single user interface.
- Transformations: Rename, condense, merge, compute across queries and data sources.
- Remarks: Utilize rich events from various data sources to add annotations to graphs.
- Panel Editor: A standardized user interface for setting up and personalizing your panels.

C. System Architecture

Prometheus has several components which help in the overall monitoring of your application. These components of the architecture are:

- Prometheus Server
- Push Gateway
- Alert Manager
- Prometheus Targets
- Client Libraries

- Prometheus Exporters
- Service Discovery

First, Prometheus scrapes data using the Prometheus server and finds targets using the Service discovery. After that, the scraped data is sent to the dashboard and processed using PromQL and sends alerts to the alert manager, who will send notifications to the user.

1) Prometheus Server

The Prometheus server functions as the "brain" of web or mobile applications, collecting multi-dimensional data in time series and analyzing and aggregating it. This

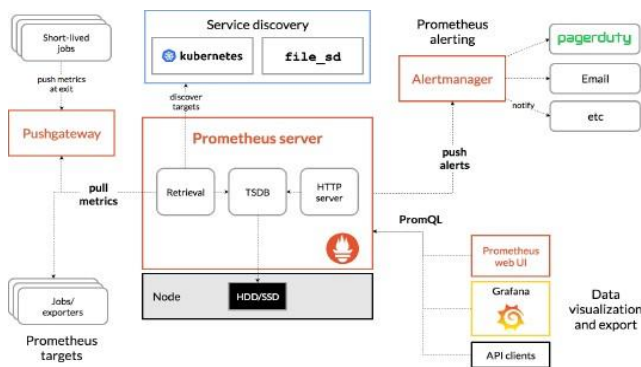


Fig. 1. Architecture

process, known as scraping, involves accessing metrics via metric names or optional key-value pairs, often distinguished by labels. Time series data is collected at successive or fixed intervals, enabling analysis and filtering for further insights.

The server automatically pulls metrics from targets, eliminating the need for manual pushing of metrics for analysis. This simplifies the client's task as they only need to expose metrics through an HTTP endpoint with "/metrics," returning the complete set of metrics.

2) Prometheus PushGateway

The Prometheus server alone would not scrape all kinds of metrics; some require extra mechanics. Prometheus Gateway is the intermediary source used for metrics from those jobs which can not be scraped by usual methods. The push gateway always exposes the data collected to the Prometheus for any reason, and we can not delete that information manually from the Gateway's API. When multiple instances of a job use an instance label for differentiating their metrics in the push gateway, the metrics remain in the push gateway even after the original entity deletes. It happens because the lifecycle of an instance in the push gateway is different from the lifecycle of the actual process. On the other hand, in Prometheus classical pull, metrics also delete when the original instance gets deleted.

3) Alertmanager

Alertmanager is responsible for managing the alerts sent by the clients. It checks for duplication, groups the signals,

and routes them to the correct application like email, Pagerduty, Opsgenie, etc. It also checks for when it should keep alerts off and when not. You can do various things with Alerts received from a client (Prometheus Server) to the Alertmanager. You can group similar types of notifications that prevent you from seeing similar notifications repetitively. You can mute notifications as well. In the alert manager, you can configure that all alerts related to the same type of instance are compiled in a single alert and then sent to the alert manager.

4) Prometheus Targets

Prometheus targets represent how Prometheus extracts metrics from a different resource. In this case, Prometheus collects metrics directly. But in some instances, like in unexposed services, Prometheus has to use exporters. Exporters are some programs that extract data from a service and then convert them into Prometheus formats. In the multi-target export pattern, the metrics export via a network. They do not have to run on the parent machines of the metrics, and they can query from multiple targets. Blackbox and SNMP exporters use multi-target exporter patterns.

5) Client Libraries

As we all know, Prometheus collects data in time-series formats that are multi-dimensional. So clients are always asked to send in this form specifically. Prometheus provides various client libraries, some are official, and some are unofficial. If you can control the source code, client libraries provide the client-specific instrumentation and metrics collection. The client library sends the current state of all tracked metrics to the server whenever Prometheus scrapes your instances' HTTP endpoint. Client libraries for instrumenting your own code are available in Go, Java/JVM, C/.Net, Python, Ruby, Node.js, Haskell, Erlang, and Rust, among other prominent languages and runtimes. Software like Kubernetes and Docker already include Prometheus client libraries. There are hundreds of integrations available for third-party software that offers metrics in a non-Prometheus format. HAProxy, MySQL, PostgreSQL, Redis, JMX, SNMP, Consul, and Kafka are examples of exporters.

6) Prometheus Exporters

As mentioned above, in most cases, metrics are self-exposed by the service. In such cases, Prometheus automatically collects metrics. In other cases, Prometheus needs to scrape metrics. Exporters are third-party tools that help scrape metrics when it is not feasible to extract metrics directly. Some exporters are official, while others are not officially declared in the Prometheus Github organization. Prometheus exporters can go into various categories such as database exporters, hardware related, issue trackers, storage, HTTP, APIs, logging, miscellaneous alert managers, and other official Prometheus exporters.

7) Service Discovery

In the Prometheus Targets section, we discussed using static-config files to configure the dependencies manually. This process is ok when you have simple uses with the config file, but what if you have to do this in a large amount? Especially when some instances are added or removed every single minute? This is where service discovery comes into play. Service discovery helps in providing Prometheus the information of what to scrape in whichever database you want. Prometheus' common

service discovery resources are Consul, Amazon's EC2, and Kubernetes out of the box.

D. Data Collection Methods

1) Exporters and Agents:

- **Exporters:** Exporters are small agents or libraries that expose metrics from various systems and services in a format that Prometheus can scrape.
- **Types of Exporters:** Prometheus provides a range of officially supported exporters for popular systems such as Node.js, Docker, MySQL, PostgreSQL, Redis, and more.
- **Custom Exporters:** Users can also write custom exporters using Prometheus client libraries to monitor specific applications or services.
- **Usage:** Exporters are deployed alongside the applications or services they monitor, exposing metrics via HTTP endpoints that Prometheus scrapes at regular intervals.

2) Instrumenting Servers:

- **Instrumentation Libraries:** Prometheus provides client libraries for various programming languages (e.g., Go, Java, Python, Ruby, etc.) that allow developers to instrument their applications with custom metrics.
- **Metrics Exposition:** Instrumented applications expose metrics via HTTP endpoints in a format that Prometheus can scrape.
- **Types of Metrics:** Metrics can include application-specific metrics such as request latency, error rates, throughput, resource utilization, and more.
- **Customization:** Developers can customize the instrumentation to expose metrics that are relevant to their application's performance and behavior.

E. Data Processing and Visualization

1) Querying With Promql

Prometheus Query Language (PromQL) is a powerful query language used to retrieve and analyze collected metrics. It enables users to perform various operations on time-series data to gain insights into system behavior and performance. Some common use cases of PromQL include:

- **Calculating Aggregates:** PromQL allows users to calculate aggregates such as averages, sums, counts, and percentiles over a specified time range. For example, users can calculate the average response time of a Node.js application over the past hour or the 90th percentile of CPU usage on Windows servers.

calculate aggregates such as averages, sums, counts, and percentiles over a specified time range. For example, users can calculate the average response time of a Node.js application over the past hour or the 90th percentile of CPU usage on Windows servers.

- **Filtering Metrics:** PromQL supports filtering metrics based on label values, enabling users to focus on specific subsets of data. For instance, users can filter metrics related to a particular service, instance, or job to analyze performance trends or identify outliers.
- **Mathematical Operations:** PromQL provides various mathematical operators such as addition, subtraction, multiplication, and division, allowing users to perform arithmetic operations on metrics. This enables users to calculate derived metrics or compute ratios between different metrics.
- **Temporal Functions:** PromQL includes temporal functions such as rate, increase, and delta, which allow users to calculate rates of change or differences between consecutive data points. These functions are useful for monitoring trends and detecting anomalies in metric data.

2) Grafana Dashboard Design

Grafana provides a user-friendly interface for designing custom dashboards to visualize Prometheus metrics. Dashboards in Grafana can be tailored to display real-time and historical data from multiple data sources, including Node.js applications, Docker containers, MySQL databases, and Windows servers. Key features of Grafana dashboard design include:

- **Data Source Integration:** Grafana supports integration with various data sources, including Prometheus, InfluxDB, Graphite, Elasticsearch, and more. Users can configure data sources in Grafana and query metrics from multiple sources within the same dashboard.
- **Visualization Options:** Grafana offers a wide range of visualization options, including line graphs, bar charts, heatmaps, gauges, and tables. Users can choose the most suitable visualization type based on the nature of the data and the insights they want to convey.
- **Panel Customization:** Grafana allows users to customize individual panels within a dashboard by adjusting settings such as colors, axis labels, legend placement, and data aggregation functions. This flexibility enables users to create visually appealing and informative dashboards tailored to their specific monitoring needs.
- **Dashboard Templating:** Grafana supports dashboard templating, allowing users to create dynamic dashboards that adapt to changes in data sources or query parameters. Templating enables users to create reusable dashboard templates and dynamically filter

data based on user input or predefined variables.

- **Annotations and Alerts:** Grafana allows users to add annotations and alerts to dashboards to highlight important events or conditions. Annotations can be used to mark downtime periods, deployment events, or performance incidents, while alerts can be configured to notify users when predefined thresholds are exceeded.

F. Alerting Mechanisms

3) Alertmanager Configuration

Alertmanager is a component of Prometheus used for handling alerts generated by the Prometheus server. It allows users to define alerting rules and configure notification channels for alert delivery. Key features of Alertmanager configuration include:

- **Alerting Rules:** Alertmanager allows users to define alerting rules based on predefined thresholds or using complex expressions involving multiple metrics. Users can specify conditions under which alerts should be triggered, such as when the response time of a Node.js application exceeds a certain threshold or when CPU utilization of Docker containers reaches critical levels.
- **Notification Channels:** Alertmanager supports various notification channels such as email, Slack, PagerDuty, and webhook integrations. Users can configure multiple notification channels and specify different notification settings for each channel.
- **Silence and Inhibition:** Alertmanager provides features for silencing and inhibiting alerts to prevent unnecessary notifications during maintenance windows or when certain conditions are met. Users can silence alerts for specific time periods or inhibit alerts based on other alert states or labels.

4) Defining Alert Rules

Alert rules specify conditions under which alerts should be triggered by the Prometheus server. Users can define alerting rules using Prometheus's expression language and specify thresholds or conditions based on metrics collected by Prometheus. Some common examples of alert rules include:

- **Node.js Application Response Time:** Alert rule to notify administrators when the response time of a Node.js application exceeds a certain threshold, indicating degraded performance or potential issues with the application.
- **Docker Container CPU Utilization:** Alert rule to trigger alerts when CPU utilization of Docker containers reaches critical levels, indicating resource contention or overload on the host system.
- **MySQL Query Latency:** Alert rule to alert administrators when MySQL query latency exceeds

acceptable limits, indicating potential database performance issues or query optimization opportunities.

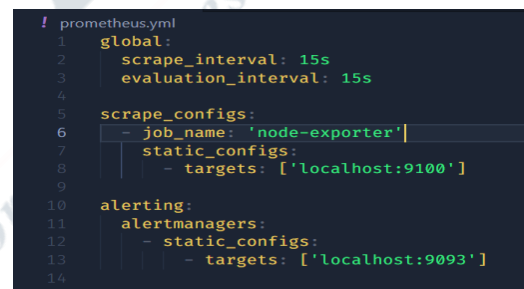
- **Windows Server Disk Space:** Alert rule to notify administrators when disk space on Windows servers is running low, indicating a potential risk of disk space exhaustion and service disruption.

By defining alert rules and configuring notification channels in Alertmanager, users can proactively monitor their infrastructure and respond to potential issues in a timely manner, ensuring the reliability and availability of their systems.

IV. IMPLEMENTATION

A. Setting up Prometheus and Grafana

To set up Prometheus and Grafana for server monitoring, it's essential that the server to be monitored and the monitoring system are within the same network. Before getting started with Prometheus, application to be monitored needs to be integrated with relevant metrics, choice of which needs to be brainstormed thoroughly. The application must be configured to expose the intended metrics to be scraped by Prometheus at a suitable endpoint. As we operated a Node.js server, we utilized the Prometheus client library for JavaScript, known as prom-client, to create and export custom metrics. This library



```

1 prometheus.yml
2 global:
3   scrape_interval: 15s
4   evaluation_interval: 15s
5
6 scrape_configs:
7   - job_name: 'node-exporter'
8     static_configs:
9       - targets: ['localhost:9100']
10
11 alerting:
12   alertmanagers:
13     - static_configs:
14       - targets: ['localhost:9093']

```

Fig. 2. Sample code of prometheus.yml file

facilitated access to a registry, where all metrics were registered and prepared for export. Selecting appropriate metric exporters for Prometheus is pivotal, as they form the backbone of our observability strategy alongside Grafana. Choice of metrics and relevant exporters is left to the users due to its dependence on the nature of application and OS of the server that needs to be monitored. We employed various exporters to cater to different metrics. For instance, we utilized Prometheus counter metrics to export the number of payments, and Custom Prometheus gauge metrics to capture payment success rates and MySQL metrics. Additionally, we exported HTTP metrics, as well as CPU and memory utilization metrics, which we identified as relevant for our server. Once the application is configured to export metrics, the next step involves setting up Prometheus, Alertmanager, and Grafana to monitor our server effectively.

B. Prometheus

Prometheus is configured through a YAML file, typically named prometheus.yml. This file includes various sections such as global configuration, alerting rules, and scrape configs. It also requires a rules-file generally (rules.yml) where all the alerting rules are expressed in promQL, which are evaluated periodically to check system health and trigger alerts.

Fig 2 depicts the sample code of prometheus.yml file. Global section defines global configuration options that apply to the entire Prometheus instance. Like scrape-interval sets the interval at which Prometheus will scrape (collect) metrics from the configured targets and evaluation-interval determines how often Prometheus will evaluate rules and update the alerting status. Scrape-config defines the configurations for scraping metrics from specific targets. Like in the above configuration, Prometheus scrapes metrics from a Node Exporter on local- host:9100 and is configured to send alerts to Alertmanager running on localhost:9093. Alerting defines configurations related to alerting. Alertmanager specifies the Alertmanager instances that Prometheus will send alerts to. Static-config allows statically configuring the targets for Alertmanager in- stances. Like in the code depicted in Fig 2, target is set to localhost on port 9093, indicating that the alerts will be sent to the Alertmanager running on this address.

```

alertmanager > ! alertmanager.yml
1  global:
2    resolve_timeout: 5m
3
4  route:
5    group_by: ['alertname', 'severity']
6    group_wait: 30s
7    group_interval: 5m
8    repeat_interval: 12h
9    receiver: 'email-notifications'
10
11
12
13 receivers:
14 - name: 'email-notifications'
15   email_configs:
16   - to: 'xyz@gmail.com'
17     from: 'abc@gmail.com'
18     smarthost: 'smtp.gmail.com:587'
19     auth_username: 'abc@gmail.com'
20     auth_password: '**** *'
21     auth_identity: 'abc@gmail.com'

```

Fig. 3. Sample code of alertmanager.yml file

C. Alertmanager

Alertmanager provides a yaml configuration file which includes (global, routes, receiver, etc) configuration settings.

Fig 3 depicts the sample code of alertmanager.yml file. Global section defines global configuration options that apply to the entire Alertmanager instance. Resolve-timeout sets the timeout duration for resolving alerts. If an alert remains active for longer than this duration, it will be considered resolved. Route specifies the routing configuration for incoming alerts. group-by defines how alerts should be grouped. Alerts with the same name and severity will be grouped together. group- wait determines how long Alertmanager should wait before grouping alerts together. group-interval specifies the interval at which

Alertmanager should group. repeat-interval sets the interval at which Alertmanager should resend notifications for the same alert. Receivers in alert manager is a configuration entity that defines how alerts are routed and where notifications are sent when triggered by an alerting system. Each receiver has a unique name and is associated with specific notification settings, such as email, Slack, PagerDuty, or custom webhook endpoints, etc. Alerts can be routed to different receivers based on various criteria, such as severity, alert name, or specific conditions defined in the routing configuration. Code depicted in Fig 3 explains how to configure route and receiver for conveying alerts through emails. SMTP server and port used for sending emails are dictated by smarthost configuration. Auth-username specifies the username used for authenticating with the SMTP server. Its typically the same as the email address. Auth-password would normally contain the pass- word associated with the auth-username for authentication. Its securely configured, typically encrypted or stored in a secure manner. Auth-identity specifies the identity used for authentication, which is often the same as the auth-username.

D. Grafana

For creating a dashboard, firstly we need to specify the appropriate datasource (Prometheus Data source). Then we need to run a prometheus query consisting of either raw metrics (exported metrics) or performing operations on these raw metrics of which results can be visualized in (time

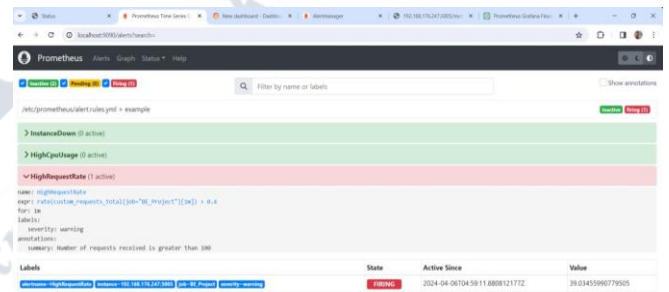


Fig. 4. Alert manager dashboard indicating that an alert has been fired.

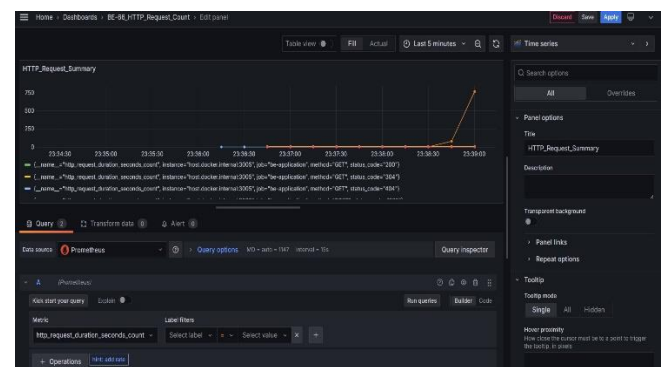


Fig. 5. Grafana dashboard creation UI

series graphs, bar graph, histogram, stats, etc). Further grafana continues to periodically refresh its dashboard to visualize the updated metrics. There are more customization options available for creating dashboards which can be explored.

V. RESULTS AND DISCUSSION

In this section, we present the results of our application monitoring using Prometheus and Grafana. We discuss the performance evaluation, integration challenges, complexity, and alerting and notification aspects encountered during the implementation and usage of these tools.

A. Performance Evaluation

Prometheus allowed us to monitor various key metrics essential for understanding the health and performance of our application. These metrics included:

- Custom Requests: Tracking the total number of custom requests served by the application, providing insights into user interaction and system usage patterns.
- Payment Success Rate: Monitoring the rate of successful payments processed by our payment gateway, crucial for assessing the reliability and efficiency of our payment processing system.
- MySQL Performance Metrics: Observing metrics such as MySQL connections, query execution time, bytes sent

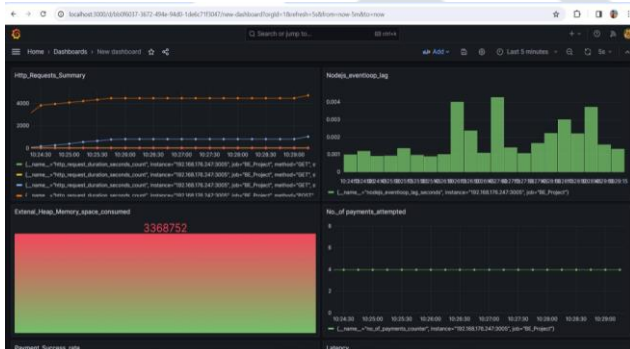


Fig. 6. Grafana metrics dashboard

and received, and queries per second, helping us optimize database performance and ensure smooth operation.

- HTTP Request Durations: Analyzing HTTP request durations via histograms for different status codes and methods, aiding in identifying performance bottlenecks and improving response times.
- Node.js Event Loop Lag: Measuring the lag of the event loop in Node.js, indicating the efficiency of event-driven processing and potential performance issues.

Visualization of metrics as seen in Fig 6 and prompt alerting after implementing Prometheus and Grafana allowed us to gain insights into the impact of monitoring on our

application's performance. We observed improved visibility into system behavior, timely detection of anomalies, and enhanced resource utilization management. Monitoring resource utilization metrics such as CPU usage, memory usage, and file descriptors helped us understand the impact of monitoring on system resources. This enabled us to optimize resource allocation and identify potential bottlenecks proactively.

B. Case Studies

- Amadeus IT Group:
 - Industry: Travel Technology
 - Use Case: Amadeus IT Group adopted Prometheus and Grafana by spinning up four clusters on-premises and deploying Prometheus on top of them. The company manually added these four data sources in Grafana for visualization. This implementation allowed Amadeus IT Group to monitor its infrastructure effectively and gain insights into system performance and resource utilization.
- Digital Ocean:
 - Industry: Cloud Services
 - Use Case: Digital Ocean found Grafana to be a natural fit for its monitoring needs. With close integration with Prometheus, new dashboards could be built quickly. This integration enabled support and platform teams to easily access visual metrics for any server in the fleet. The intuitive UI, powerful query editor, and beautiful visualizations of Grafana empowered teams throughout the company to build and share dashboards, fostering collaboration and facilitating data-driven decision-making.
- JP Morgan Chase and Co.:
 - Industry: Financial Services
 - Use Case: JP Morgan Chase and Co. utilized Prometheus and Grafana to monitor their technical landscape. By leveraging trade volumes, synthetic transactions, and alerts created by SRE precepts, they built a comprehensive monitoring tool using Grafana. This solution enabled them to record trends and proactively highlight issues, ensuring the stability and reliability of their technical infrastructure.

C. Challenges Encountered

- Network Setup and Configuration: One of the significant challenges encountered during integration was setting up the network and configuring IP addresses for Prometheus and Grafana instances. Ensuring proper communication and connectivity between different components required careful planning and configuration. However, thorough documentation and community support helped in

over-coming these challenges.

- **Complexity:** Configuring and maintaining Prometheus and Grafana setups required a deep understanding of metrics collection, storage, visualization, and alerting. Managing complex configurations, especially in large-scale deployments, demanded meticulous attention to detail and continuous optimization efforts.
- **Alerting and Notification:** Configuring SMTP settings for email alerts posed challenges, especially in environments where dedicated email servers were not readily available. To address this, we leveraged Google's SMTP servers, which are accessible for developers and provide a reliable email delivery mechanism. However, configuring SMTP settings and ensuring seamless integration with Prometheus Alert Manager required careful coordination to ensure proper authentication and security measures were in place.

D. Lessons Learned

- **Proper planning and setup:** Effective monitoring with Prometheus and Grafana requires careful planning and setup. We learned the importance of defining clear monitoring objectives, selecting relevant metrics to track, and establishing robust data collection mechanisms. Additionally, configuring Prometheus and Grafana setups to align with specific business requirements and infrastructure complexities proved crucial for deriving actionable insights from monitoring data.
- **Continuous Optimization:** Continuous optimization of monitoring configurations is essential for maintaining the effectiveness of Prometheus and Grafana setups over time. We discovered the value of regularly reviewing and fine-tuning monitoring configurations to adapt to changing business needs, evolving system architectures, and shifting performance requirements. By optimizing alerting thresholds and dashboard layouts we ensured that the monitoring systems remained aligned with operational goals and provided actionable insights.

VI. CONCLUSION AND FUTURE SCOPE

The adoption of Prometheus and Grafana for application monitoring represents a significant advancement in the field of IT operations and DevOps practices. Throughout this research paper, we have explored the capabilities, features, and real-world applications of these powerful monitoring tools. Case studies show how Amadeus IT Group, Digital Ocean, and JP Morgan Chase and Co., among other top companies, successfully used Prometheus and Grafana to monitor their infrastructure, increase operational effectiveness, and improve service reliability. With its flexible query language,

rapid data collection systems, and strong time-series database, Prometheus gives businesses the capacity to monitor a wide range of metrics across a variety of infrastructures. Because of its integrated integration with Grafana, a flexible dashboarding and visualization platform, users can easily track key performance metrics, create intelligent visualizations, and identify abnormalities.

Looking ahead, the future of application monitoring with Prometheus and Grafana holds immense promise. With ongoing advancements in cloud-native technologies, machine learning, and automation, we anticipate further enhancements in scalability, predictive analytics, and self-healing capabilities. As organizations continue to embrace digital transformation and strive for operational excellence, Prometheus and Grafana will remain indispensable tools in their arsenal for monitoring and managing complex IT environments effectively.

In conclusion, Prometheus and Grafana give businesses the ability to continuously monitor, assess, and improve their infrastructure and applications, helping them to remain ahead of the curve in a dynamic and fiercely competitive digital environment.

REFERENCES

- [1] L. Chen, M. Xian and J. Liu, "Monitoring System of OpenStack Cloud Platform Based on Prometheus," 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), Chongqing, China, 2020, pp. 206-209, doi: 10.1109/CVIDL51233.2020.0-100.
- [2] Rawoof F. M., Tajammul M., "A Survey on Remote On-Premise Server Monitoring", 2022 Journal of Emerging Technologies and Innovative Research (JETIR) Volume 9, Issue 2.
- [3] Kumar A. K., Vinutha B. S., Vinayaditya B.V., "REAL TIME MONITORING OF SERVERS WITH PROMETHEUS AND GRAFANA FOR HIGH AVAILABILITY", Apr 2019, International Research Journal of Engineering and Technology (IRJET).
- [4] Bose S., Rakesh K.R., "Server status Monitoring using Advanced Machine Learning Algorithms", 6 June 2020, International Journal of Creative Research Thoughts(IJCRT) Volume 8.
- [5] O. Mart, C. Negru, F. Pop and A. Castiglione, "Observability in Kubernetes Cluster: Automatic Anomalies Detection using Prometheus," 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Yanuca Island, Cuvu, Fiji, 2020, pp. 565-570, doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00071.
- [6] N. Sukhija and E. Bautista, "Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus," 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of

- People and Smart City Innovation (Smart- World/ SCALCOM/UIC/ATC/CBDCOM/IOP/SCI), Leicester, UK, 2019, pp. 257-262, doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00087.
- [7] Nicolas Chan. 2019. A Resource Utilization Analytics Platform Using Grafana and Telegraf for the Savio Supercluster. In Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning) (PEARC '19). Association for Computing Machinery, New York, NY, USA, Article 31, 1–6.
- [8] I. Josko Plazonic, Jonathan Halverson, and Troy Comi. 2023. Jobstats: A Slurm-Compatible Job Monitoring Platform for CPU and GPU Clusters. In Practice and Experience in Advanced Research Computing (PEARC '23). Association for Computing Machinery, New York, NY, USA, 102–108. <https://doi.org/10.1145/3569951.3604396>
- [9] Zhang Yu-Long, Cao Heng, Hu Jing-Feng, Wang Jin-Cheng, “Design and implementation of remote monitoring system for welding machine based on web”, 2018
- [10] A. Kaushik, “Use of Open Source Technologies for Enterprise Server Monitoring Using SNMP”, IJCSE, Vol. 2, No. 7, pp. 2246-2252, 2010.
- [11] J. Swarna, C. S. Raja, D. Ravichandran, “Cloud Monitoring Based on SNMP”, Journal of Theoretical and Applied Information Technology, Vol. 40, No. 2, pp. 188-193, 2012.
- [12] Andreas witting and Michael witting, Amazon web services in actions, ISBN- 1617292885, 17/10/2015.
- [13] G. Suci, V. Suci, R. Gheorghe, C. Dobre, F. Pop, and A. Castiglione. “Analysis of Network Management and Monitoring Using Cloud Computing”, Computational Intelligence and Intelligent Systems, Springer, pp. 343-352, 2016
- [14] Ikram Hawramani, Cloud computing for complete beginners: Building and scaling high performance web servers on the amazon cloud.
- [15] Chakraborty M, Kundan AP. Grafana. In: Monitoring Cloud-Native Applications [Internet]. Berkeley, CA, Apress; 2021. [cited 15th August 2022] Available from: <https://doi.org/10.1007/978-1-4842-6888-96>