System Operation Monitoring Project on AWS

by

Prosenjit Roy
ID: CSE2103024119

Md. Abdullah Al Fattah
ID: CSE2103024142

Sultana Yeasmin
ID: CSE2103024144

Abu Saleh Mahammad Nazmus Shakib
ID: CSE2103024146

Supervised by
Md. Ashfakur Rahman

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2025

APPROVAL

The project titled "**System Operation Monitoring Project on AWS**" submitted by Prosenjit Roy (CSE2103024119), Md. Abdullah Al Fattah (CSE2103024142), Sultana Yeasmin (CSE2103024144) and Abu Saleh Mahammad Nazmus Shakib (CSE2103024146) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Md. Ashfakur Rahman
Lecturer
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Md. Ashfakur Rahman,** Lecturer**,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Md. Ashfakur Rahman)
Supervisor

Signature

Prosenjit Roy
ID: CSE2103024119

Md. Abdullah Al Fattah
ID: CSE2103024142

Sultana Yeasmin
ID: CSE2103024144

Abu Saleh Mahammad Nazmus Shakib
ID: CSE2103024146

ABSTRACT

In modern digital infrastructure, uninterrupted web service availability and real-time performance monitoring are crucial for ensuring user satisfaction and operational reliability. This project focuses on the design and implementation of a Website Monitoring Tool using open-source technologies Prometheus, Grafana, and Alert Manager on a Linux-based environment. The primary goal of the project is to monitor the operational health of a web application server hosted on a cloud platform such as AWS, although it is equally suitable for any Linux-based server deployment. The Node Exporter is installed on the application server to expose system-level metrics such as CPU usage, memory utilization, disk activity, and network performance. These metrics are then scraped and stored by Prometheus, which acts as the time-series database and monitoring engine. For real-time visualization, Grafana is used to create custom dashboards that allow users and administrators to observe performance trends and detect anomalies instantly. To handle critical situations, Alert Manager is configured with predefined alerting rules (e.g., high CPU usage, server down). Upon meeting the alert conditions, the system can send email notifications via Gmail, ensuring prompt action by system administrators.

This project demonstrates an effective, scalable, and cost-efficient approach to monitoring the health of websites and servers. It leverages the power of automation, open-source tools, and visual analytics to enhance system observability and reliability.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the schedule time.

We are auspicious that we had the kind association as well as supervision of **Md. Ashfakur Rahman**, Lecturer, Department of Computer Science and Engineering, Sonargaon University whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We would like to convey our special gratitude to **Prof. Bulbul Ahamed**, Head of the Department, Computer Science and Engineering, Sonargaon University, for their kind concern and precious suggestions.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

API	Application Programming Interface
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
KPI	Key Performance Indicator
NMS	Network Monitoring System
RAM	Random Access Memory
ROM	Read Only Memory
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
YAML	YAML Ain ‘t Markup Language

TABLE OF CONTENTS

Title			Page No.
DECLARATION ……			iii
ABSTRACT ……			iv
ACKNOWLEDGEMENT ……			v
LIST OF ABBREVIATION ……			vi
CHAPTER 1			1 – 2
SYSTEM OPERATION MONITORING PROJECT ON AWS			
1.1	Introduction ……		1
1.2	Objectives of the Project…….		1
1.3	Scope of the Project ……..		1
1.4	Methodology …….		2
1.5	Project Justification ……		2
1.6	Tools and Technologies Used……		2
1.7	Limitations of the Project……..		2
1.8	Future Scope……		2
CHAPTER 2			3 – 13
RELATED WORK			
2.1	Introduction …….		3
2.2	Existing Monitoring Tools …….		3
	2.2.1	Nagios……	3 – 4
	2.2.2	Zabbix……	4 – 5
	2.2.3	Datadog……..	6 – 7
	2.2.4	UptimeRobot……..	7 – 8
	2.2.5	Monit……..	9 – 10
2.3	Comparison Table……...		10
2.4	Gaps Identified in Existing Solutions……		10 –11
	2.4.1	Complex Configuration and Steep Learning Curves……	11
	2.4.2	High Cost or Usage Limitations in SaaS-Based Tools……..	11

2.4.3 Lack of Real-Time Monitoring with Modular and Lightweight Architecture……... 11
2.4.4 Limited Integration with Modern DevOps and Cloud-Native Environments……. 12
2.5 Advantages of Proposed System……. 12
2.5.1 Open-Source Stack…… 12
2.5.2 Modular Architecture……. 12
2.5.3 Custom Dashboards……... 12 –13
2.5.4 Flexible Alerts…… 13
2.5.5 Scalability…….. 13
2.6 Chapter Summary…… 13

CHAPTER 3 14 –27

SYSTEM DESIGN AND IMPLEMENTATION

3.1 Introduction ……..…..... 14
3.2 System Requirements ……. 14
3.2.1 Functional Requirements …….. 14 –15
3.2.2 Non-Functional Requirements …….. 15 –16
3.3 System Architecture ……... 16
3.3.1 Node Exporter Probes the Target Websites……... 16
3.3.2 Prometheus Scrapes and Stores the Probe Metrics…… 16
3.3.3 Alert Manager Sends Notifications Based on Prometheus Alerts…….. 16 –17
3.3.4 Grafana Visualizes Metrics via Customizable Dashboards…... 17
3.4 Tool Setup & Configuration…… 17
3.4.1 Installing Prometheus ……... 17 –19
3.4.2 Installing Node Exporter …….. 20 –21
3.4.3 Installing Grafana ……. 21 –22
3.4.4 Configuring Grafana Dashboards ……. 22 –23
3.4.5 Installing and Configuring Alert Manager……. 24 –26
3.5 Summary……. 27

CHAPTER 4 28 –36

SYSTEM TESTING AND ANALYSIS

4.1 Introduction ……..……... 28
4.2 Test Environment Setup ……. 28
4.2.1 Hardware Specifications ……... 28
4.2.2 Software Stack …….……. 28 –29
4.2.3 Network Configuration…….. 29
4.2.4 Test Services…….. 29
4.2.5 Installation Method…… 29
4.3 Test Objectives ……... 29 –30
4.4 Test Cases and Execution …….. 30
4.4.1 Test Cases for the Monitoring System…….. 31
4.5 Sample Screenshot – Dashboard View …….. 32
4.6 Sample Screenshot – Probe Metrics (Prometheus) …… 32 –33
4.7 Sample Screenshot – Alert Triggered …… 33
4.8 Sample Screenshot – Email Notification ……... 34
4.9 Metrics Analysis ……..……... 35
4.9.1 Types of Metrics Collected ..……. 35
4.9.2 Sample Metric Queries (PromQL) ..…….. 35
4.9.3 Visualization in Grafana ..……. 35
4.9.4 Performance Observations ..…….. 35
4.9.5 Alert Analysis ..……. 36
4.9.6 Summary of Findings ..……. 36
4.10 Summary …… 36

CHAPTER 5 37 –39

LIMITATIONS AND FUTURE WORKS

5.1 Limitations ……..…... 37 –38
5.2 Future Works …….…… 38 –39

CHAPTER 6 40 –41

CONCLUSION

REFERENCE…… 42

LIST OF TABLES

Table No.	Title	Page No.
Table 1.1	Tools and Technologies Used	2
Table 2.1	Comparison Table	10
Table 4.1	Hardware Specifications	28
Table 4.2	Software Stack	29
Table 4.3	Test Cases for the Monitoring System	31
Table 4.4	Summary of Findings	36

LIST OF FIGURES

Figure No.	Title	Page No.
Fig.3.1	Update System Packages	17
Fig.3.2	Create Prometheus User and Directories	18
Fig.3.3	Move Binaries and Set Permissions	18
Fig.3.4	Set proper ownership	18
Fig.3.5	Create a System Service File	19
Fig.3.6	Reload System and Start Prometheus	19
Fig.3.7	Installing Node Exporter	20
Fig.3.8	Start and Enable Node Exporter	20
Fig.3.9	Integration with Prometheus	21
Fig.3.10	Grafana APT Repository	21
Fig.3.11	Create a Configuration File	24
Fig.3.12	Start Alert Manager	25
Fig.3.13	Test Alerting Rules	26
Fig.4.1	Sample Screenshot – Dashboard View	32
Fig.4.2	Sample Screenshot – Probe Metrics (Prometheus)	33
Fig.4.3	Sample Screenshot – Alert Triggered	33
Fig.4.4	Sample Screenshot – Email Notification	34
Fig.5.1	Future System	39

CHAPTER 1

INTRODUCTION TO SYSTEM OPERATION MONITORING PROJECT ON AWS

1.1 Introduction

In today's digital-first world, websites are critical assets for businesses, institutions, and organizations. Downtime or poor performance of a website can lead to revenue loss, damaged reputation, and diminished user trust. Monitoring tools are essential to ensure that websites remain available, fast, and healthy.

This project presents a Website Monitoring Tool that uses Prometheus, Grafana, and Alert Manager, deployed on a Linux (Ubuntu) environment. These open-source technologies offer a robust, scalable, and customizable solution for continuous website and service monitoring. This system ensures that website metrics such as uptime, response time, and performance thresholds are monitored in real-time, with alerting mechanisms for immediate issue detection and resolution.

1.2 Objective of the Project

The primary objectives of this project are:

- To design and implement a reliable, real-time monitoring tool for websites and services.
- To collect and visualize system metrics using Prometheus and Grafana.
- To configure alert rules and deliver notifications via Alert Manager.
- To create a flexible and scalable solution that can be adapted for different server environments.

1.3 Scope of the Project

This monitoring tool can be implemented across various types of web services, from single page applications to enterprise-level websites. It allows monitoring of:

- HTTP status codes (e.g., 200, 404, 500)
- Response time and latency
- Uptime/Downtime tracking
- CPU and memory usage (if desired)
- Alert thresholds for anomalies

It is suitable for:

- System administrators
- DevOps engineers
- IT teams managing production environments

1.4 Methodology

The project uses a modular approach. The methodology includes:

- **Installation of Prometheus**: For metric scraping and storage.
- **Installation of Grafana**: For building dynamic dashboards.
- **Configuration of Alert Manager**: To handle alerts from Prometheus.
- **Integration and testing** of the complete system on a Linux (Ubuntu) server.

1.5 Project Justification

Due to the rising need for service reliability and performance, this project provides a low cost, open-source alternative to commercial monitoring platforms. With custom alerting and powerful visualizations, users gain deep insights into their web service performance, enabling faster issue resolution and better uptime guarantees.

1.6 Tools and Technologies Used

Tool/Technology	Purpose
Prometheus	Metric collection & storage
Grafana	Data visualization
Alert Manager	Alert dispatching
Ubuntu Server	Hosting environment
Node Exporter	For system-level monitoring

Table 1.1 Tools and Technologies Used

1.7 Limitations of the Project

- Initial setup may be complex for beginners unfamiliar with Linux systems.
- Requires manual configuration for each target endpoint.
- Alert delivery depends on network/email configuration.
- Limited mobile integration unless third-party tools is added.

1.8 Future Scope

- Integration with mobile apps for push notifications.
- Auto-scaling configuration for cloud environments.
- Custom dashboards per team/user.
- SMS/Telegram alert integration.
- Dockized version for container-based deployment.

CHAPTER 2
RELATED WORK

2.1 Introduction

Website monitoring is a critical function in modern IT infrastructure management. Numerous tools and platforms—both open-source and commercial—exist to provide monitoring, alerting, and analytics for system performance and availability. This chapter explores related monitoring tools, compares them with the proposed system, and identifies the gaps that this project addresses.

2.2 Existing Monitoring Tools

Several monitoring tools are currently available in the market, each offering unique features for tracking system health, performance, and availability. Below are some of the most widely used tools

2.2.1 Nagios

Nagios is a powerful and flexible open-source IT infrastructure monitoring system. Originally known as NetSaint, Nagios has been a foundational tool in the system and network monitoring domain since its first release. It allows organizations to identify and resolve IT infrastructure problems before they affect critical business processes.[4]

Key Features:

- **Comprehensive Monitoring:** Nagios can monitor network services (SMTP, POP3, HTTP, NNTP, ICMP, SNMP), host resources (processor load, disk usage, system logs), and application-specific parameters.
- **Alerting System:** Provides timely alerts via email, SMS, or custom scripts when problems are detected and when they are resolved.
- **Plugin Architecture:** Supports a modular plugin framework. Thousands of community-contributed plugins are available for monitoring everything from basic uptime to complex enterprise applications.
- **Performance Data Collection:** Supports integration with graphing tools like PNP4Nagios or NagiosGraph for visual performance data analysis.
- **Extensibility:** Can be extended via plugins, custom scripts, and integration with third-party tools like NRPE (Nagios Remote Plugin Executor), NCPA (Nagios Cross Platform Agent), and NSClient++.

Architecture:

Nagios uses a server/agent model. The main server runs checks directly or through agents installed on remote systems. The results are then processed and visualized on a web interface or dashboard.

Configuration:

- Configuration is handled through plain-text files, which can be difficult and time-consuming to set up for large environments.
- Hosts, services, and contact definitions are all managed through .cfg files, which requires familiarity with the Nagios syntax and logic.

Web Interface:

- The default web interface is functional but outdated compared to modern UI standards.
- Several forks and frontends (such as Nagios XI, Thruk, and Centreon) offer improved dashboards and usability.

Strengths:

- Mature and stable, with a large community and extensive documentation.
- Excellent for monitoring complex, hybrid infrastructures.
- Fine-grained control over checks, alert conditions, and dependencies.

Weaknesses:

- Steep learning curve due to complex configuration.
- The core system lacks modern visualization and dashboard features unless extended with other tools.
- Not ideal for dynamic or cloud-native environments without significant customization.

Use Cases:

- Suitable for small to medium-sized businesses needing robust monitoring with high configurability.
- Often used in traditional data center environments where static IPs and well-defined services are monitored.

2.2.2 Zabbix

Zabbix is a highly scalable, enterprise-grade open-source monitoring solution designed to track the status of servers, virtual machines, network devices, applications, databases, and cloud services in real-time. Developed by **Zabbix LLC**, it supports both agent-based and agentless monitoring, and is widely adopted in large-scale IT infrastructures.[5]

Key Features:

- **All-in-One Monitoring**: Provides unified monitoring for networks, servers, cloud environments, containers, services, databases, and more.
- **Agent-Based and Agentless Monitoring**: Uses Zabbix agents for detailed metrics or SNMP, IPMI, SSH, and HTTP for agentless checks.
- **Advanced Visualization**: Offers graphs, charts, maps, and dashboards to visualize real-time and historical performance data.

- **Built-in Notification and Escalation System**: Supports multi-step notifications and escalation rules through email, SMS, script execution, or third-party integrations like Slack or Telegram.
- **Auto-Discovery and Auto-Registration**: Automatically detects network devices and registers new hosts based on rules.
- **Template-Based Monitoring**: Uses predefined or custom templates for efficient and consistent monitoring configuration.
- **Database Storage**: All configuration and monitoring data are stored in a relational database like MySQL or PostgreSQL.

Architecture:
Zabbix follows a centralized server-based architecture, consisting of the following components:

- **Zabbix Server**: The core of the system that collects and processes data.
- **Zabbix Agent**: Installed on hosts to collect local metrics.
- **Zabbix Proxy (optional)**: Used to offload processing in large or distributed environments.
- **Frontend**: A modern, responsive web interface written in PHP.

Configuration:

- Configuration can be done entirely through the web GUI, making it easier than editing plain-text files.
- Supports massive automation through templates, macros, and low-level discovery (LLD).

Web Interface:

- Fully responsive, feature-rich, and actively maintained.
- Supports real-time dashboards, event correlation, historical data access, and user-defined widgets.

Strengths:

- Highly customizable with excellent visualization.
- Scalable architecture suitable for both small and large infrastructures.
- Strong support for automation and dynamic monitoring environments.
- Built-in support for problem detection using triggers and dependencies.

Weaknesses:

- Requires significant system resources, especially for large setups.
- Complex to configure optimally—steep learning curve for beginners.
- May require fine-tuning of database and polling intervals for performance.

Use Cases:

- Best suited for medium to large enterprises needing detailed, real-time monitoring.
- Used in environments where visualization, automation, and flexibility are priorities.

2.2.3 Datadog

Datadog is a commercial SaaS-based monitoring and analytics platform designed for modern, dynamic cloud-based infrastructures. It offers real-time observability across the entire technology stack, integrating metrics, logs, traces, and other telemetry data from multiple sources into unified dashboards.[6]

Key Features:

- **Infrastructure Monitoring**: Tracks host metrics, container metrics (e.g., Docker, Kubernetes), and cloud infrastructure performance (AWS, Azure, GCP).
- **Log Management**: Aggregates logs from different services and enables full-text search, filtering, and analytics.
- **APM (Application Performance Monitoring)**: Supports distributed tracing for identifying bottlenecks in microservices-based applications.
- **Dashboards**: Customizable and interactive dashboards with real-time metrics and visualization.
- **AI-Powered Alerts**: Offers anomaly detection and forecasting using machine learning.
- **Integrations**: Over 600+ built-in integrations for technologies like Apache, MySQL, NGINX, Kubernetes, Jenkins, and more.
- **Security Monitoring**: Detects threats and security misconfigurations across infrastructure and applications.

Architecture:

Datadog uses a SaaS-based agent model. Lightweight agents are installed on servers or containers to collect system metrics and logs. Data is securely transmitted to Datadog's cloud infrastructure where it's processed, analyzed, and visualized.

- **Agent**: Installed on hosts to collect metrics, traces, and logs.
- **API**: Allows programmatic interaction for pushing custom metrics and automating workflows.
- **Cloud Backend**: All data is processed in Datadog’s cloud, relieving users from managing backend infrastructure.

Configuration:

- **Configuration Simplicity**: Initial setup is straightforward through UI or YAML-based configuration files.
- **Tagging**: Supports dynamic tagging of resources for easy filtering, grouping, and visualization.
- **Infrastructure as Code**: Works with tools like Terraform and Ansible for automated deployment and configuration.

Web Interface:

- The web interface is modern, intuitive, and user-friendly.
- Offers real-time metric correlation across infrastructure, logs, and APM data.
- Supports sharing dashboards and reports among team members and stakeholders.

Strengths:

- Easy to deploy and scale, especially in containerized and cloud-native environments.
- Intuitive and modern UI with powerful dashboarding capabilities.
- Provides a single pane of glass for metrics, logs, and traces.
- Minimal setup time with rich out-of-the-box integrations.

Weaknesses:

- **Cost**: High subscription and per-host pricing can be prohibitively expensive for small teams, individual users, or academic projects.
- **Vendor Lock-in**: As a proprietary SaaS solution, users are dependent on Datadog's infrastructure and data storage.
- Requires internet connectivity and continuous data transfer to the cloud, which may not suit air-gapped or restricted environments.

Use Cases:

- Large enterprises and DevOps teams monitoring production-scale, distributed systems.
- Teams operating in dynamic environments like Kubernetes, serverless, and hybrid cloud infrastructures.
- Businesses that need comprehensive observability in a single platform without maintaining infrastructure.

2.2.4 UptimeRobot

UptimeRobot is a cloud-based Software-as-a-Service (SaaS) monitoring solution that primarily focuses on website and service uptime monitoring. Launched in 2010, it has grown in popularity due to its simplicity, intuitive user interface, and quick setup process. UptimeRobot is ideal for users who need basic availability monitoring without the complexity of server-side installations or advanced configurations.[7]

Key Features:

- **Website and Port Monitoring**: UptimeRobot checks website availability at regular intervals (as frequently as every 1 minute in the paid tier) using HTTP(s), keyword checks, ping, port, and even heartbeat monitoring for custom apps.
- **Alerting System**: Sends alerts via multiple channels including email, SMS, voice calls, push notifications (via apps like Pushover, PagerDuty, Slack, Microsoft Teams), and webhooks when downtime or status changes occur.
- **Dashboard and Reporting**: Provides a modern web dashboard and mobile apps with real-time status, historical uptime logs, and response time graphs. Public status pages are also supported for transparency with users or clients.
- **Heartbeat Monitoring**: Can track scheduled jobs, CRON jobs, or background tasks by expecting periodic pings from them—useful for detecting silent failures.
- **Multi-location Checks**: To minimize false positives, UptimeRobot confirms downtime from multiple global checkpoints before sending alerts.

Architecture:
Unlike Nagios' server/agent model, UptimeRobot is entirely agentless and cloud-hosted. No installation or server-side setup is required. All monitoring is conducted from UptimeRobot's distributed global infrastructure, making it highly accessible and easy to deploy for non-technical users.

Configuration:

- Configuration is done entirely through a web GUI or API.
- Adding a new monitor takes just seconds: specify the URL or IP, choose the check type (HTTP, Ping, Port, etc.), and define alert contacts.
- No need for text-based config files like in Nagios (.cfg), significantly reducing setup time and complexity.

Web Interface:

- UptimeRobot has a clean, modern, and mobile-friendly web interface.
- Includes visual graphs for response time, uptime history, and availability statistics.
- Public status pages can be customized and shared externally a feature not natively available in Nagios Core.

Strengths:

- **Ease of Use**: Extremely user-friendly and fast to configure.
- **Cloud-Based**: No infrastructure or maintenance required on the user's part.
- **Public Status Pages**: Helpful for showing uptime to customers or stakeholders.
- **Multi-channel Alerts**: Supports a wide range of notification integrations.

Weaknesses:

- **Limited Depth**: Only monitors at the application (website) level—no support for system metrics like CPU, memory, or disk usage.
- **Free Tier Limitations**: The free version is limited to 50 monitors at 5-minute intervals and lacks advanced features like SSL expiry monitoring and custom HTTP headers.
- **No Extensibility**: Does not support custom plugins, scripting, or infrastructure integration like Nagios.
- **No On-Premises Option**: All data resides in the cloud; not ideal for highly regulated or air-gapped environments.

Use Cases:

- Ideal for website owners, SaaS providers, bloggers, and eCommerce businesses who need to ensure public-facing services are up.
- Commonly used by small businesses and startups that need a lightweight monitoring solution without a dedicated IT team.
- Best suited for uptime and basic performance monitoring rather than in-depth infrastructure analysis.

2.2.5 Monit

Monit is a lightweight, open-source utility designed primarily for monitoring and managing Unix-based systems. Unlike comprehensive monitoring tools such as Nagios, Monit focuses on process supervision, service availability, and automatic recovery, making it especially useful for system administrators looking to ensure that specific services remain operational.

Originally developed to minimize downtime by automating service restarts, Monit excels in monitoring local system resources and processes with minimal configuration and system overhead.[8]

Key Features:

- **Process Monitoring and Recovery**: Monit can monitor system processes, services, and daemons (e.g., Apache, MySQL, Nginx) and automatically restart them if they crash or become unresponsive.
- **File and Directory Monitoring**: Capable of tracking file changes, such as checksum validation, timestamp modifications, and file size growth (useful for log files or configuration tracking).
- **System Resource Monitoring**: Monit tracks CPU usage, memory consumption, load averages, and disk space. Alerts are generated when thresholds are exceeded.
- **Network Testing**: Supports simple TCP and HTTP checks to verify that services are not only running but also reachable and functioning correctly.
- **Built-in Web Interface**: Provides a lightweight, secure web-based interface for monitoring status and logs in real-time. This is included out-of-the-box without the need for third-party tools.
- **Event-based Alerting**: Sends alerts via email or custom scripts when errors occur, and notifications when services are restarted or return to normal.

Architecture:

Monit is a **standalone daemon** that runs on the monitored server. It does **not** require agents or a central monitoring server like Nagios or Prometheus. It checks services at regular intervals using its internal scheduler and acts upon its own decisions based on pre-defined rules.

Configuration:

- Configuration in Monit is relatively simple and human-readable, making it accessible to users with limited experience in system monitoring.
- Monit supports grouping of services and dependency tracking.

Web Interface:

- The Monit Web UI is minimalistic but functional, showing service status, system resource usage, and log messages.
- It allows starting/stopping/restarting services and viewing logs directly from the browser.

- Runs by default on port 2812 and can be secured with basic HTTP authentication and SSL.

Strengths:

- Lightweight, fast, and uses minimal system resources.
- Easy to install and configure.
- Automatic self-healing capabilities (restart services upon failure).
- Ideal for individual servers or lightweight infrastructure setups.
- No external dependencies or databases required.

Weaknesses:

- Not suitable for large-scale or distributed infrastructure monitoring.
- Lacks built-in support for long-term performance metrics, graphs, or historical data analysis.
- Limited alerting channels compared to tools like Nagios or Prometheus.
- No centralized dashboard for managing multiple nodes.

Use Cases:

- Perfect for single-server deployments or containers where simple service uptime and auto-restart are required.
- Commonly used in embedded systems, IoT devices, and small to medium web servers.
- Acts as a first line of defense against service failures in non-complex systems.

2.3 Comparison Table

Tool/Platform	Open Source	Web Monitoring	Alerting	Dashboards	Complexity	Cost
Nagios	Yes	Yes	Yes	Limited	High	Free
Zabbix	Yes	Yes	Yes	Yes	Medium-High	Free
Datadog	No	Yes	Yes	Yes	Low	Paid
UptimeRobot	No (Free Tier)	Yes	Yes	No	Very Low	Limited Free
Monit	Yes	Basic	Yes	No	Low	Free
Proposed System	Yes(All tools)	Yes	Yes	Yes	Medium	Free

Table 2.1 Comparison Table

2.4 Gaps Identified in Existing Solutions

Despite the availability of numerous monitoring tools—ranging from open-source frameworks to commercial SaaS platforms—several critical limitations remain that affect their usability, scalability, and cost-efficiency. The following are key gaps observed in the current landscape of monitoring solutions:

2.4.1 Complex Configuration and Steep Learning Curves

- Open-source tools like Nagios, Zabbix, and Icinga offer extensive monitoring capabilities but require intricate and manual configuration.
    - Example: Nagios depends heavily on .cfg files and plugin scripting, which are not intuitive for beginners.
    - Zabbix has a more modern web UI but demands proper database setup, agent configuration, and dependency mapping for full functionality.
- The configuration burden increases with scale, leading to high initial setup times and maintenance overhead.
- Lack of automation or built-in configuration templates for common services adds to deployment complexity.

2.4.2 High Cost or Usage Limitations in SaaS-Based Tools

- Commercial solutions like Datadog, UptimeRobot, Pingdom, and New Relic provide polished interfaces and easy setup but come with:
    - Tiered pricing models that may limit the number of hosts, checks, or data retention in free or basic plans.
    - Pay-per-metric or pay-per-host models, which can become prohibitively expensive in large-scale or budget-conscious environments.
    - Vendor lock-in, which may hinder customizability or integration with other open-source tools.
- These services are less suitable for institutions, educational projects, or small businesses seeking cost-effective solutions with long-term sustainability.

2.4.3 Lack of Real-Time Monitoring with Modular and Lightweight Architecture

- Many traditional tools (e.g., Nagios, MRTG) poll at fixed intervals (e.g., every 5 minutes), leading to delays in detecting fast-changing issues.
- Some open-source tools lack real-time dashboards or require third-party integrations (e.g., Nagios + PNP4Nagios) to achieve near real-time visualization.
- Heavyweight frameworks often consume more system resources and are tightly coupled to specific backends or databases, making them less suitable for lightweight or embedded deployments.
- There is a shortage of tools that are:
    - Modular (supporting independent components like exporters, visualizers, and alerting systems).
    - Lightweight (minimal memory/CPU usage).
    - Extensible (easy to integrate with custom scripts, APIs, or cloud-native architectures).

2.4.4 Limited Integration with Modern DevOps and Cloud-Native Environments

- Legacy tools were designed for static infrastructure (e.g., physical servers and VMs), and adapting them to dynamic cloud environments (Kubernetes, containers, auto-scaling) often requires complex customization or external adapters.
- Few tools support service discovery, auto-registration, and metrics aggregation out-of-the-box in cloud-native ecosystems.
- Tools without native support for Prometheus exporters, Grafana dashboards, or API-based alerting fall short in modern CI/CD pipelines and microservice monitoring.

2.5 Advantages of Proposed System

The proposed Website Monitoring Tool using Prometheus, Grafana, and AlertManager offers several significant advantages over traditional monitoring solutions [1]. Built entirely on open-source technologies, it ensures high flexibility, extensibility, and cost-effectiveness. Below are the key advantages with detailed explanations:[10]

2.5.1 Open-Source Stack

The entire monitoring system is based on open-source tools—Prometheus, Grafana, and AlertManager—which are free to use and backed by active community support and contributions.

- No licensing fees, reducing total cost of ownership.
- Frequent updates and improvements due to strong community and corporate backing (e.g., CNCF).
- Access to large repositories of plugins, exporters, and templates.

2.5.2 Modular Architecture

Each component of the system operates independently and communicates via well-defined interfaces.

- Prometheus handles metrics collection and time-series data storage.
- Grafana provides the visualization and dashboarding layer.
- AlertManager manages alert processing and routing.

This modularity allows:

- Replacing or upgrading individual components without disrupting the entire system.
- Easy integration with other tools like Loki (for logs), Thanos (for long-term storage), or Elasticsearch.
- Simplified troubleshooting and system maintenance.

2.5.3 Custom Dashboards

Grafana offers powerful and flexible dashboard creation tools.

- Supports a wide range of data sources and visualizations (graphs, heatmaps, gauges, tables).
- Allows filtering, templating, and querying using **PromQL** or other data languages.

- Dashboards can be customized per team, per service, or per metric, enabling tailored views for different stakeholders (e.g., DevOps, Managers, QA).
- Role-based access control (RBAC) can be implemented for dashboard permissions.

2.5.4 Flexible Alerts

AlertManager enhances the alerting capabilities of Prometheus by supporting complex routing and notification strategies.

- Alerts can be grouped to avoid notification fatigue (e.g., multiple alerts for the same host).
- Supports alert silencing during maintenance periods or known outages.
- Multi-channel notifications: email, Slack, Microsoft Teams, PagerDuty, SMS, and webhooks.
- Routing rules allow directing different types of alerts to different teams or channels.

2.5.5 Scalability

The system is suitable for a wide range of deployment scales, from small websites to enterprise-level infrastructure.

- Horizontal scalability: Prometheus can be run in a federation model to collect metrics from multiple instances or data centers.
- Easily adaptable to containerized environments like Docker or Kubernetes.
- Designed for high performance, capable of ingesting millions of metrics per second with appropriate configuration.
- Grafana can connect to multiple Prometheus instances or even third-party data sources, allowing for distributed monitoring.

2.6 Chapter Summary

This chapter reviewed several existing website monitoring tools and platforms, comparing their features, complexity, and cost. The comparison highlighted that while many tools are powerful, they may be complex or costly. The proposed system fills this gap by offering a balanced, open-source, scalable, and easy-to-manage solution.[12]

CHAPTER 3
SYSTEM DESIGN AND IMPLEMENTATION

3.1 Introduction

This chapter explains the design and implementation of the website monitoring system using Prometheus, Grafana, and Alert Manager on a Linux (Ubuntu) environment. It includes an overview of system requirements, architecture, configuration steps, and deployment of each tool used in the system.

3.2 System Requirements

The system requirements define the core functionalities and operational characteristics expected from the proposed Website Monitoring Tool built using Prometheus, Grafana, and Alert Manager. These requirements are categorized into Functional Requirements and Non-Functional Requirements to clearly distinguish system behavior from performance and technical constraints.

3.2.1 Functional Requirements

These requirements describe what the system should do and outline the specific functionalities it must support.

- **Website Monitoring**
  The system must actively monitor specified websites and web applications by periodically checking:
  - HTTP response status codes (e.g., 200 OK, 404 Not Found, 500 Internal Server Error)
  - Response time (latency in milliseconds)
  - Availability/Uptime (whether the website is reachable)

- **Metrics Storage**
  - Collected metrics must be stored using Prometheus in a time-series database format for querying and analysis.
  - Metrics retention should be configurable to allow short-term or long-term analysis.[11]

- **Dashboard Visualization**
  - Metrics should be visualized using Grafana dashboards.
  - Dashboards must allow filtering by website, time range, and metric type (e.g., latency, downtime).
  - Visualizations may include line graphs, bar charts, uptime heatmaps, and statistical panels.[3]

- **Alerting System**
  - The system must generate alerts based on defined thresholds, such as:

        - Website is unreachable (e.g., HTTP 5xx or timeout).
        - Response time exceeds a specified threshold (e.g., 1 second).

    - Alerts must be:
        - Configurable via Prometheus alert rules.
        - Routed and grouped through Alert Manager.[9]

- **Notifications**
    - Alerts must be sent via:
        - Email
        - Third-party services such as Slack, Discord, Microsoft Teams, or any platform supporting webhook integration.

    - Notification messages should contain:
        - Affected website
        - Type of issue (e.g., down, slow).
        - Timestamp and severity level

- **System Configuration**
    - Administrators must be able to:
        - Add/edit/remove monitored websites.
        - Adjust alert thresholds.
        - Configure notification channels.

3.2.2 Non-Functional Requirements

These define how the system should perform and operate, rather than specific behaviors.

- **Real-Time Monitoring**
    - Data collection and alerting must occur at near real-time intervals (e.g., every 15–60 seconds).
    - Delays between issue detection and alert notification should be minimal.
- **Scalability**
    - The system should support monitoring of:
        - Dozens to hundreds of endpoints without performance degradation.
        - Horizontal scaling through federation or multiple Prometheus instances if necessary.
- **Reliability & Fault Tolerance**
    - The system should continue operating under partial failure conditions.
    - Recovery mechanisms (e.g., retry logic, service restarts) should be supported.
- **Open-Source Based**

    - The solution must be built entirely on open-source technologies to minimize licensing and infrastructure costs.
    - Components should have active community support and documentation.
- **Security**
    - Access to Grafana dashboards and system configurations should be secured with authentication.
    - Communication between components (especially webhooks) should use encrypted protocols (HTTPS).
- **Maintainability**
    - Configuration files and deployment scripts should be documented and modular.
    - Logs and system metrics should be accessible for debugging and optimization.

3.3 System Architecture

3.3.1 Node Exporter Probes the Target Websites

- Node Exporter is a Prometheus exporter for hardware and OS-level metrics such as CPU, memory, disk usage, and network statistics.
- However, to probe websites (URL availability, response time, etc.), Prometheus typically uses a tool like Blackbox Exporter, not Node Exporter.
- The exporter performs HTTP, HTTPS, ICMP (ping), and TCP checks on the defined targets.
- It collects:
    - HTTP response codes (e.g., 200, 500),
    - Response latency,
    - SSL certificate expiry,
    - Uptime metrics.

3.3.2 Prometheus Scrapes and Stores the Probe Metrics

- Prometheus is the core monitoring engine responsible for data collection and time-series storage.
- It periodically scrapes metrics from exporters (like Blackbox Exporter or Node Exporter) at predefined intervals (e.g., every 15s or 60s).
- All scraped data is stored in a time-series database, optimized for performance and scalability.
- Prometheus can also apply recording rules to precompute complex metrics and alerting rules to evaluate conditions like:
    - “Website down for more than 2 minutes”
    - “SSL certificate will expire in 7 days”

3.3.3 Alert Manager Sends Notifications Based on Prometheus Alerts

- Alert Manager handles all alert notifications generated by Prometheus.
- It supports:

    - Alert grouping to prevent flooding users with multiple alerts from the same incident.
    - Deduplication to avoid sending duplicate notifications.
    - Silencing of known or maintenance-period alerts.
    - Routing rules to direct different types of alerts to the appropriate teams or contacts.
- Alerts can be sent via:
    - Email
    - Slack
    - Telegram
    - Microsoft Teams
    - Webhooks
    - PagerDuty, OpsGenie, and more

3.3.4 Grafana Visualizes Metrics via Customizable Dashboards

- Grafana is the visualization layer of the monitoring system.
- It queries Prometheus for real-time and historical data using PromQL (Prometheus Query Language).
- Users can build custom dashboards with graphs, gauges, bar charts, tables, and other widgets.
- Dashboards can be:
    - Shared or embedded
    - Filtered by time ranges, services, hosts, etc.
    - Configured to show alert status alongside metrics
- Grafana supports alerting rules too, which can be used in parallel with or independently of Prometheus alerts.

3.4 Tool Setup & Configuration

3.4.1 Installing Prometheus

Prometheus is the core component of the monitoring system. It is responsible for scraping metrics from targets, storing them in a time-series database, and exposing them via a powerful query language (PromQL).
Below are the steps for installing and configuring Prometheus on a Linux (Ubuntu) system:[2]

Step 1: Update System Packages
Update the system package.

```
sudo apt update && sudo apt upgrade -y
```

Fig 3.1 Update System Packages

Step 2: Create Prometheus User and Directories
Create a dedicated system user and necessary directories:

```
sudo useradd --no-create-home --shell /bin/false prometheus

sudo mkdir /etc/prometheus
sudo mkdir /var/lib/prometheus
```

Fig 3.2 Create Prometheus User and Directories

Step 3: Move Binaries and Set Permissions

```
sudo cp prometheus /usr/local/bin/
sudo cp promtool /usr/local/bin/

sudo cp -r consoles /etc/prometheus
sudo cp -r console_libraries /etc/prometheus
sudo cp prometheus.yml /etc/prometheus/
```

Fig 3.3 Move Binaries and Set Permissions

Set proper ownership:

```
sudo chown -R prometheus:prometheus /etc/prometheus
sudo chown prometheus:prometheus /usr/local/bin/prometheus
sudo chown prometheus:prometheus /usr/local/bin/promtool
sudo chown -R prometheus:prometheus /var/lib/prometheus
```

Fig 3.4 Set proper ownership

Step 4: Create a System Service File
Create the service file for Prometheus:

```
sudo nano /etc/systemd/system/prometheus.service
```

Paste the following content:

```
[Unit]
Description=Prometheus Monitoring
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
ExecStart=/usr/local/bin/prometheus \
  --config.file=/etc/prometheus/prometheus.yml \
  --storage.tsdb.path=/var/lib/prometheus/ \
  --web.console.templates=/etc/prometheus/consoles \
  --web.console.libraries=/etc/prometheus/console_libraries
```

Fig 3.5 Create a System Service File

Save and exit the file.

Step 5: Reload System and Start Prometheus

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable prometheus
sudo systemctl start prometheus
```

Verify status:

```
sudo systemctl status prometheus
```

Fig 3.6 Reload System and Start Prometheus

Step 6: Access Prometheus Web UI

Prometheus runs on port **9090** by default. You can access the UI via:

http://<your-server-ip>:9090

This UI allows querying metrics, visualizing time series data, and exploring targets

3.4.2 Installing Node Exporter

Node Exporter is a Prometheus exporter for hardware and OS metrics exposed by *nix systems, such as CPU usage, memory statistics, disk I/O, and network statistics. It is an essential component for system-level monitoring in a Prometheus-based monitoring stack.[3]

Step-by-Step Installation on Ubuntu:

Step 1: Download the Latest Node Exporter Binary

Visit the official Prometheus download page or use wget to download the latest version directly:

Step 2: Extract the Archive

Step 3: Create a System User (Optional but Recommended)

Step 4: Create a system Service File

Step 5: Add the following contents:

Add the following contents:

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=default.target
```

Fig 3.7 Installing Node Exporter

Step 6: Start and Enable Node Exporter

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
```

Fig 3.8 Start and Enable Node Exporter

Step 7: Verify Node Exporter is Running

Default Port:

- Node Exporter listens on port 9100 by default.
- Ensure your firewall allows traffic on this port for Prometheus to scrape metrics.

Integration with Prometheus:
To collect metrics from Node Exporter, add the following job to your prometheus.yml file:

```
- job_name: 'node_exporter'
  static_configs:
    - targets: ['localhost:9100']
```

Fig 3.9 Integration with Prometheus

After editing, restart Prometheus to apply changes

3.4.3 Installing Grafana

Grafana is the visualization layer in the proposed monitoring stack, used to display metrics from Prometheus in customizable, interactive dashboards. It supports various data sources and is essential for turning raw time-series data into actionable insights.
Step-by-Step Installation on Ubuntu (20.04/22.04)[3]

Step 1: Update System Packages
Ensure your system is up to date:
Step 2: Add Grafana APT Repository
Grafana is not available in the default Ubuntu repositories. Add Grafana's official repository:

```
sudo apt install -y software-properties-common
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
echo "deb https://packages.grafana.com/oss/deb stable main" | sudo tee /etc/apt/sources.list.d/gra
```

Fig 3.10 Grafana APT Repository

Step 3: Install Grafana
Step 4: Enable and Start the Grafana Service
Step 5: Verify Service Status
You should see an active (running) status.

Accessing the Grafana Web Interface

- Open a browser and navigate to: http://<your-server-ip>:3000
- Default credentials:
  - Username: admin
  - Password: admin
- You will be prompted to set a new password upon first login.

Initial Configuration
After logging in:

1. Add Data Source:
   - Navigate to Settings → Data Sources → Add data source.

- Select Prometheus, enter the URL (http://localhost:9090 if running locally), and click Save & Test.

2. Import Dashboards:
    - Grafana supports importing ready-made dashboards.
    - Go to + → Import, enter a dashboard ID from Grafana's official dashboard repository, and configure it to use your Prometheus data source.

3.4.4 Configuring Grafana Dashboards

Grafana is a widely-used open-source platform for monitoring and observability. In the proposed system, Grafana is configured to visualize the time-series metrics collected by Prometheus, enabling users to monitor system health, performance, and alerts in real-time through interactive dashboards.[3]

Step-by-Step Configuration:

Step 1: Accessing the Grafana Interface

After Grafana is installed and started, it is typically accessible at: http://<server-ip>:3000

- Default login credentials:
    - Username: admin
    - Password: admin (You will be prompted to change this on first login)

Step 2: Adding Prometheus as a Data Source

- Navigate to "Configuration" > "Data Sources"
- Click "Add data source"
- Select Prometheus
- Enter the URL of the Prometheus server (e.g., http://localhost:9090)
- Click "Save & Test" to verify the connection

Step 3: Creating a New Dashboard

- Go to "Create" > "Dashboard"
- Click "Add new panel"
- In the query section, choose Prometheus as the data source
- Enter a PromQL query (e.g., node_cpu_seconds_total or up)
- Select the preferred visualization type: Graph, Gauge, Table, Bar Gauge, etc.
- Customize panel title, axes, legends, units, and thresholds

Step 4: Using Dashboard Variables (Optional but Recommended)

- Create dynamic dashboards by adding **template variables** (e.g., for hosts, services)
- Go to **"Dashboard Settings" > "Variables"**
- Add a variable (e.g., host) and define a query like:
  Label values(instance)
- Use $host in PromQL queries to filter data based on user selection

Step 5: Importing Prebuilt Dashboards

Grafana supports importing community dashboards from its official repository.

- Go to "Dashboards" > "Import"
- Enter a dashboard ID from Grafana Dashboards Repository
    - Example: 1860 for node exporter full
- Paste the ID or JSON, then link it to your Prometheus data source

Step 6: Dashboard Sharing and Permissions

- **Dashboard Sharing:**
- Grafana dashboards can be shared via a public link, snapshot, or embedded in web pages.
- Use the “Share” button on the top-right corner of a dashboard to copy a link or generate a snapshot.
- Snapshots can include live or static data, useful for sharing without exposing the entire Grafana instance.
- **User Access Control:**
- Grafana supports role-based access control (RBAC) for users and teams.
- Roles include Viewer, Editor, and Admin, each with different permissions.
- Viewer: Can view dashboards but cannot edit.
- Editor: Can create and modify dashboards.
- Admin: Full access, including user management.
- **Organization and Teams:**
- Multiple organizations can be set up in a single Grafana instance.
- Users can be grouped into teams for easier permission management and dashboard sharing.
- **Folder-Level Permissions:**
- Dashboards can be organized into folders, each with configurable access rules.
- Set permissions on folders to control who can view or edit dashboards within them.

Step 7: Saving and Reusing Dashboards

- Click “Save Dashboard”, give it a meaningful name and folder
- Dashboards can be exported/imported for version control or multi-server use

Visual Output Example:

A typical Grafana dashboard may display:

- CPU, Memory, and Disk usage trends
- System uptime and service status (via up metric)
- Alert statuses and history
- HTTP response times and error rates (if web service metrics are enabled)

3.4.5 Installing and Configuring Alert Manager

Alert Manager is a key component of the Prometheus monitoring stack. It handles alert notifications generated by Prometheus and manages features such as deduplication, grouping, inhibition, silencing, and routing to different receivers (e.g., email, Slack, webhook, etc.).[9]

Step-by-Step Installation and Configuration:

Step 1: Download and Install Alert Manager

- Go to the official Prometheus download page.
- Download the appropriate version for your operating system.
- Move the binary to a system-wide location:

Step 2: Create a Configuration File

```
global:
  resolve_timeout: 5m
  smtp_smarthost: 'smtp.gmail.com:587'
  smtp_from: 'youremail@gmail.com'
  smtp_auth_username: 'youremail@gmail.com'
  smtp_auth_password: 'your_app_password'

route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 5m
  repeat_interval: 3h
  receiver: 'email-notifications'

receivers:
  - name: 'email-notifications'
    email_configs:
      - to: 'admin@example.com'
```

Fig 3.11 Create a Configuration File

Step 3: Start Alert Manager

Service File Content:

```
[Unit]
Description=AlertManager
After=network.target

[Service]
User=prometheus
ExecStart=/usr/local/bin/alertmanager \
  --config.file=/etc/alertmanager/alertmanager.yml \
  --storage.path=/var/lib/alertmanager/

[Install]
WantedBy=multi-user.target
```

Then run:

```
sudo systemctl daemon-reexec
sudo systemctl start alertmanager
sudo systemctl enable alertmanager
```

Fig 3.12 Start Alert Manager

Step 4: Verify Installation

Access the Web Interface:

- Open your browser and navigate to: http://<your-server-ip>:9093
- The Alert Manager web UI should load, showing current alert groups, statuses, and routes.

Check Alert Groups and States:

- Confirm that no alerts are firing initially unless configured.
- If test alerts are enabled, verify they appear as Firing, Resolved, or Silenced.

Check Configuration Validity:

- On the UI, go to "Status" > "Config" to view the active Alert Manager configuration.
- Validate that email receivers, Slack routes, or webhook configurations are listed correctly.

Review Silences:

- Go to “Silences” tab to check if any scheduled maintenance windows or alert silences are active.
- Add a test silence to confirm functionality.

Log Verification:

- Check the Alert Manager service logs for any startup or configuration errors
  erify Port Accessibility:
- Ensure port 9093 is open and accessible via firewall or security group settings (e.g., ufw, iptables, or cloud firewall rules).

Send Test Alert:

- Temporarily add a simple alert rule in Prometheus to simulate an alert and verify it appears in Alert Manager.

Step 5: Integrate with Prometheus

In this step, the configured exporters are integrated with Prometheus by updating the Prometheus configuration file (prometheus.yml). This involves defining scrape_configs that specify the targets from which Prometheus should collect metrics. Each job corresponds to a specific exporter (e.g., Node Exporter, Blackbox Exporter), and includes target addresses, scrape intervals, and any necessary relabeling rules.

Once the configuration is updated, Prometheus must be reloaded or restarted to apply the changes. After integration, the Prometheus web interface can be used to verify that the exporters are correctly discovered and their statuses are reported as "UP." This step ensures that Prometheus begins collecting real-time metrics from all designated monitoring endpoints, forming the foundation for subsequent visualization and alerting.

Step 6: Test Alerting Rules

```
groups:
- name: example-alert
  rules:
  - alert: HighCPUUsage
    expr: node_cpu_seconds_total{mode="idle"} < 20
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "High CPU usage detected on {{ $labels.instance }}"
      description: "CPU usage is over 80% for more than 1 minute."
```

Fig 3.13 Test Alerting Rules

3.5 Summary

This chapter presented a detailed overview of the tools and technologies used in the proposed Website Monitoring Tool—namely Prometheus, Grafana, and Alert Manager—alongside a comparison with other existing monitoring solutions such as Nagios, Zabbix, and Monit. Each tool's functionality, architecture, strengths, and limitations were analyzed to justify the selection of the chosen stack.

The proposed system stands out due to its open-source nature, modular architecture, and advanced visualization and alerting capabilities. It enables scalable and flexible monitoring tailored to a variety of environments, from small-scale setups to enterprise-grade deployments.

The chapter concluded with a discussion of the main advantages offered by the system, including cost-effectiveness, custom dashboards, flexible alert routing, and ease of integration. These qualities make the Prometheus-Grafana-Alert Manager stack a robust and modern solution for real-time website and infrastructure monitoring.

CHAPTER 4
SYSTEM TESTING AND ANALYSIS

4.1 Introduction

System testing and analysis are crucial stages in the development and deployment of any monitoring solution. This chapter focuses on validating the performance, reliability, and accuracy of the Website Monitoring Tool using Prometheus, Grafana, and Alert Manager. The testing process ensures that the system functions as intended, correctly captures website metrics, visualizes data accurately, and triggers alerts under defined conditions.

The testing covers various aspects, including data collection, alert triggering, dashboard rendering, and system behavior under different scenarios (e.g., normal operation, service failure, high traffic). Functional, performance, and integration testing methodologies are used to assess whether all components interact seamlessly and deliver consistent results.

This chapter also presents an analysis of system behavior based on test cases and simulated events. Results and observations are recorded to demonstrate the robustness and practicality of the monitoring solution in a real-world environment.[10]

4.2 Test Environment Setup

To validate the functionality and performance of the proposed **Website Monitoring Tool**, a controlled test environment was established. This environment simulates real-world scenarios to ensure that Prometheus, Grafana, and Alert Manager work cohesively for website and server monitoring tasks.

4.2.1 Hardware Specifications

The system was deployed on a virtualized Linux environment with the following hardware configurations:

Component	**Specification**
Processor	Intel Core i5 (4 cores)
RAM	8 GB DDR4
Storage	120 GB SSD
Network Interface	1 Gbps Ethernet
Virtualization Tool	Oracle VirtualBox / VMware

Table 4.1 Hardware Specifications

4.2.2 Software Stack

The monitoring solution was built on an open-source Linux distribution (Ubuntu) along with the following key components:

Software	**Version/Details**
Operating System	Ubuntu 22.04 LTS (64-bit)
Prometheus	v2.51.1

Grafana	v10.2.3
Node Exporter	v1.7.0
Web Server for Testing	Apache 2.4 / Nginx
Alert Manager	v0.27.0

Table 4.2 Software Stack

4.2.3 Network Configuration

- IP Scheme: Static IP assigned (e.g., 192.168.1.100 for monitoring server)
- Port Mappings:
    - Prometheus: 9090
    - Grafana: 3000
    - Alert Manager: 9093
    - Node Exporter: 9100

4.2.4 Test Services

To simulate a real website environment, the following services were deployed and monitored:

- Apache web server (HTTP)
- SSH service (for remote monitoring)
- Local system metrics via Node Exporter
- Sample HTML site hosted on /var/www/html

4.2.5 Installation Method

All components were installed using official package repositories or binaries from trusted sources. Configuration was done manually to ensure custom setup flexibility:

- Prometheus scraped targets were defined in prometheus.yml.
- Grafana was configured to use Prometheus as a data source.
- Alerts were written in YAML format and loaded by Alert Manager.

4.3 Test Objectives

The primary objective of testing the Website Monitoring Tool using Prometheus, Grafana, and Alert Manager is to ensure that the system functions as expected under various conditions and reliably performs its monitoring, alerting, and visualization tasks. The testing phase aims to validate the system's stability, accuracy, scalability, and responsiveness.

The specific test objectives are as follows:

- **Verify Accurate Metric Collection**:
  To ensure Prometheus correctly collects and stores time-series data from configured targets (e.g., web servers, databases, system metrics).

- **Ensure Dashboard Functionality:**
  To confirm Grafana accurately visualizes metrics in real-time and supports custom queries, filters, and dashboard sharing without data loss or latency.
- **Validate Alert Triggering**
  To test whether Alert Manager generates and sends notifications based on alert rules when thresholds are breached (e.g., high CPU usage, server down).
- **Check Alert Routing and Grouping**
  To ensure that alerts are grouped and routed correctly based on labels, severities, or sources to the appropriate communication channels (e.g., email, Slack).
- **Assess System Stability Under Load**
  To observe system behavior and performance under varying loads, ensuring it remains responsive and does not miss metrics or alerts under stress.
- **Verify Integration Between Components**
  To test seamless interaction between Prometheus, Grafana, and Alert Manager—ensuring data flows consistently and accurately from one component to another.
- **Test Fault Recovery and Resilience**
  To simulate failures (e.g., service crashes, network interruptions) and verify whether the system can recover, resume monitoring, and log/report issues accurately.
- **Validate Configuration Reloading**
  To test the system's ability to dynamically reload configuration files (e.g., Prometheus YAML config, Alert Manager routes) without requiring a restart or causing downtime.
- **Evaluate Security and Access Control**
  To verify user authentication, role-based access (if implemented in Grafana), and ensure secure data handling and alert transmission.

4.4 Test Cases and Execution

This section outlines the test cases used to validate the functionality and performance of the Website Monitoring Tool using Prometheus, Grafana, and Alert Manager. Each test case was designed to ensure the system components work both independently and as a cohesive unit. Testing focused on metrics collection, alert generation, visualization accuracy, and system reliability.

4.4.1 – Test Cases for the Monitoring System

Test Scenario	Expected Outcome	Actual Outcome	Status
Prometheus server startup	Prometheus starts and UI available at localhost:9090	As expected	Pass
Node Exporter availability	Metrics available at localhost:9100/metrics	As expected	Pass
Target service down	Prometheus marks service as DOWN	As expected	Pass
Alert rule condition met	Alert Manager receives alert and triggers notification	As expected	Pass
Alert silencing	Alert is suppressed when silence rule is active	As expected	Pass
Grafana dashboard loads	Grafana dashboard displays metrics accurately at localhost:3000	As expected	Pass
Data refresh in dashboard	Real-time metrics update without manual refresh	As expected	Pass
Email alert delivery	Email is sent when alert is triggered	As expected	Pass
Service recovery alert	Alert Manager sends recovery notification after service restores	As expected	Pass
Resource threshold monitoring (CPU > 80%)	Alert triggered when CPU usage exceeds threshold	As expected	Pass

Table 4.3 Test Cases for the Monitoring System

Execution Environment:

- OS: Ubuntu 22.04 LTS
- Browser: Mozilla Firefox / Google Chrome
- Prometheus: v2.x
- Grafana: v10.x
- Alert Manager: v0.25.x
- Node Exporter: v1.6.x

4.5 Sample Screenshot – Dashboard View

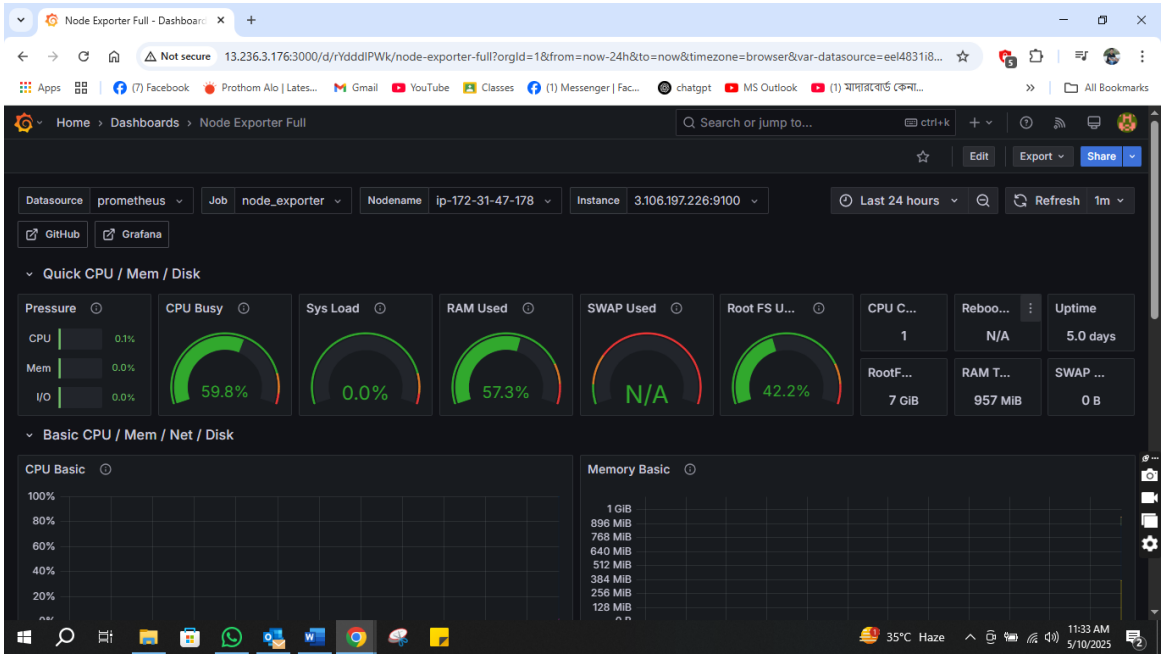
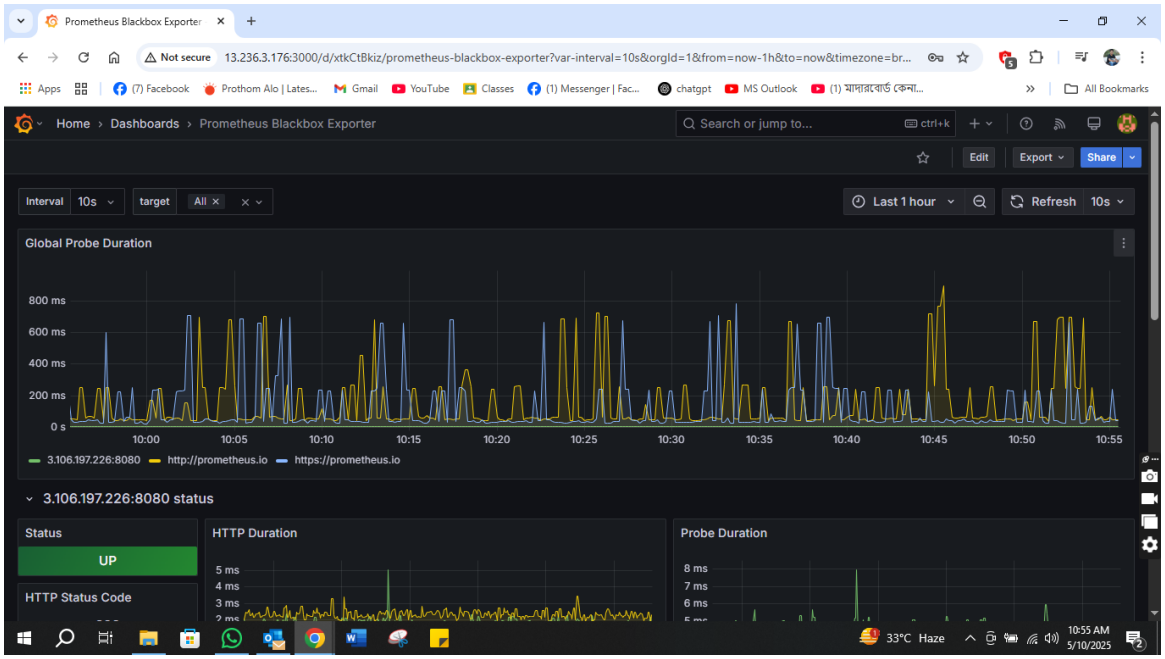


Fig 4.1 Sample Screenshot – Dashboard View

4.6 Sample Screenshot – Probe Metrics (Prometheus)



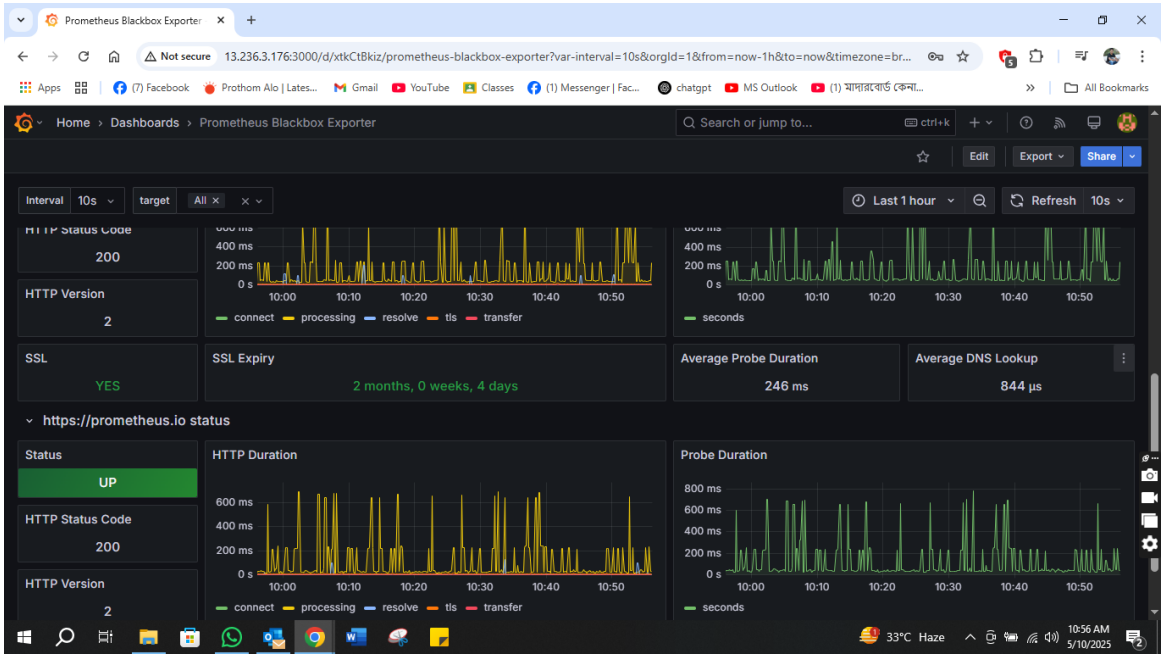


Fig 4.2 Sample Screenshot – Probe Metrics (Prometheus)

4.7 Sample Screenshot – Alert Triggered

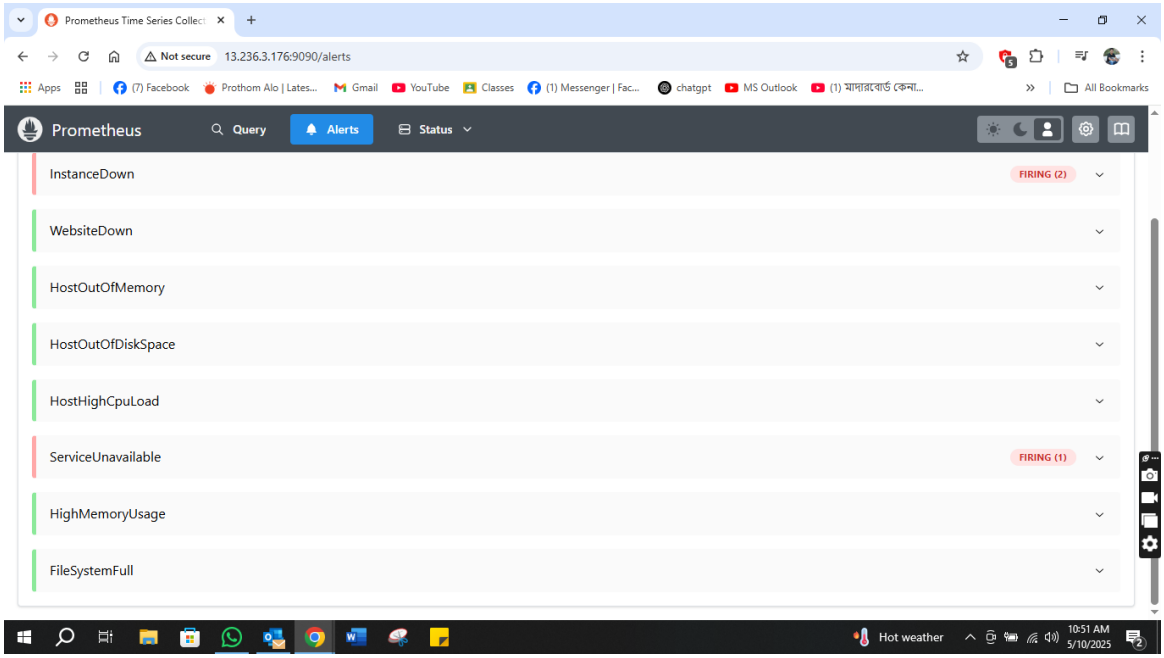
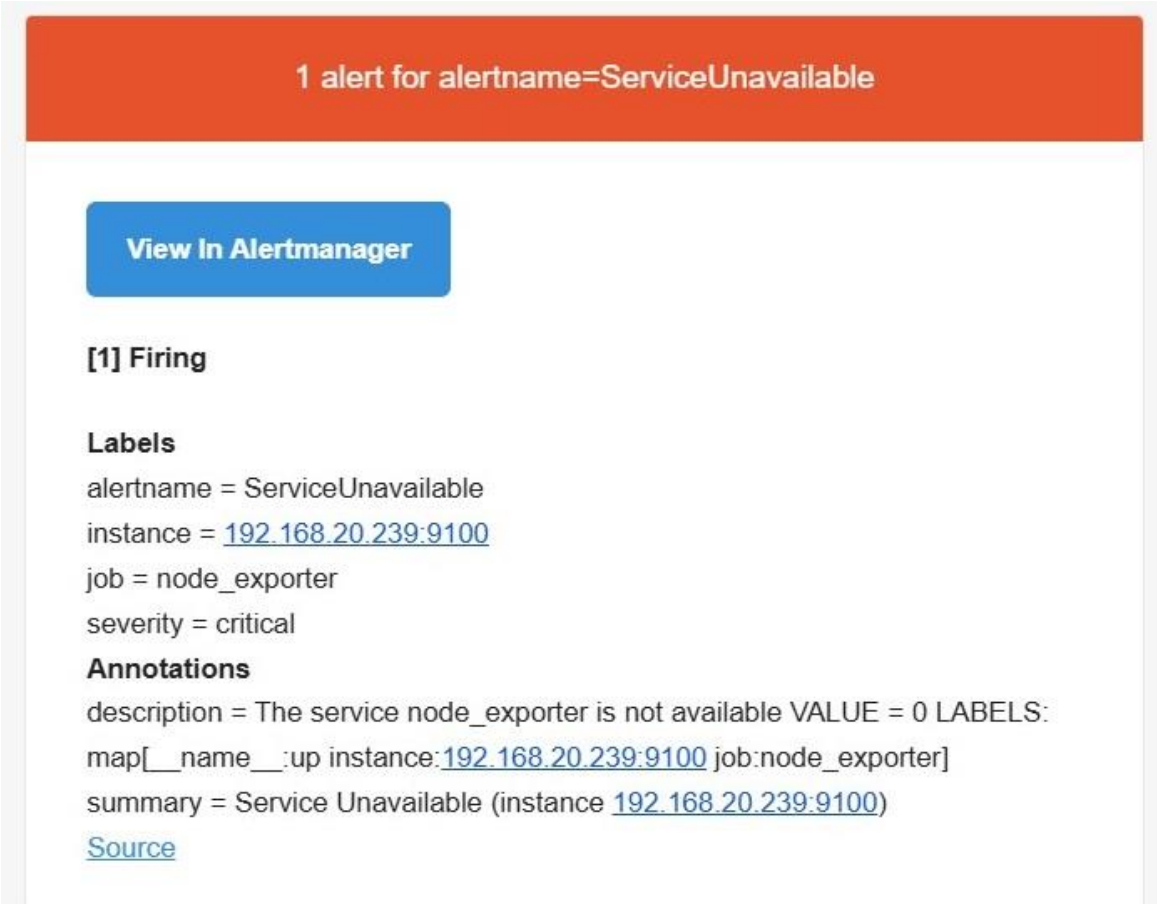


Fig 4.3 Sample Screenshot – Alert Triggered

4.8 Sample Screenshot – Email Notification



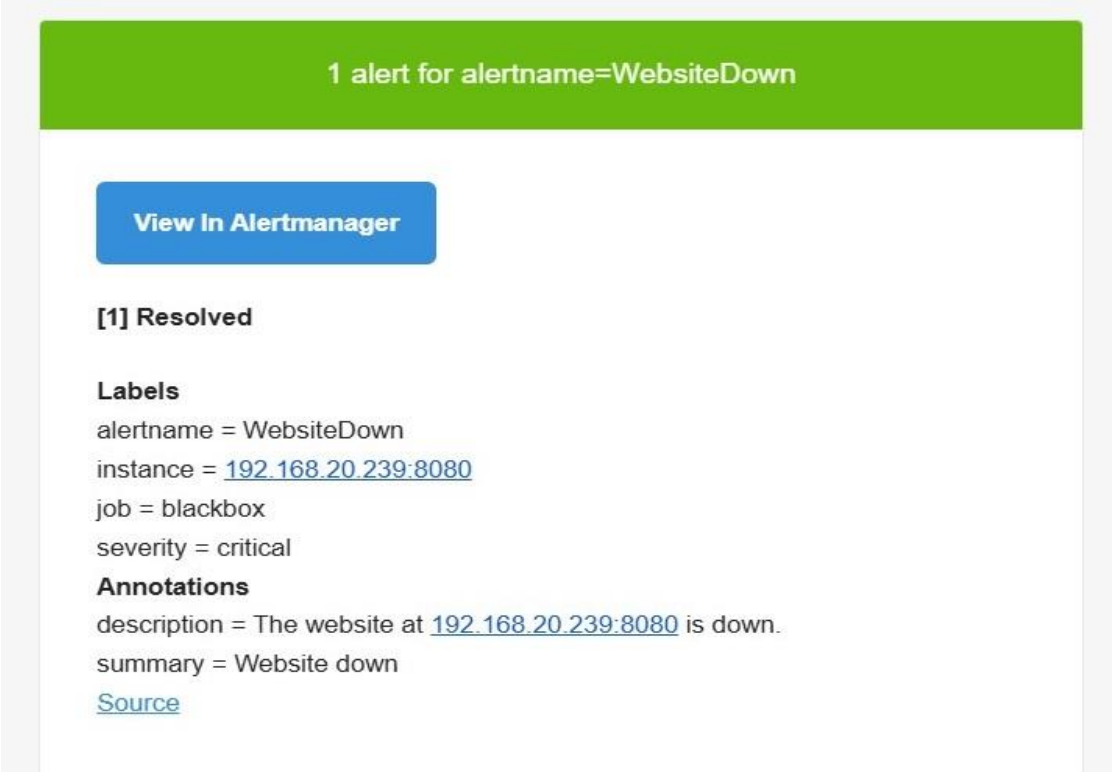


Fig 4.4 Sample Screenshot – Email Notification

4.9 Metrics Analysis

Metrics analysis is a critical aspect of system testing, as it provides quantitative data to evaluate the performance, availability, and reliability of the monitored website or infrastructure. Using Prometheus as the primary data collector and Grafana for visualization, various key performance indicators (KPIs) were observed and analyzed during the testing phase of the proposed monitoring system.

4.9.1 Types of Metrics Collected

The system captured the following categories of metrics:

- System Metrics: CPU usage, memory utilization, disk I/O, and load average.
- Network Metrics: Incoming/outgoing traffic, packet loss, and latency.
- Application Metrics: HTTP response codes, request duration, and availability status of the target website.
- Exporter Metrics: Performance and uptime of Prometheus node exporters and black box exporters.

4.9.2 Sample Metric Queries (PromQL)

Prometheus Query Language (PromQL) was used to extract and analyze specific metrics. Examples include:

- node_cpu_seconds_total{mode="user"} – Measures CPU usage in user mode.
- node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100 – Calculates memory usage percentage.
- probe_success – Indicates whether the blackbox probe to the website was successful.
- rate(http_requests_total[5m]) – Monitors the rate of incoming HTTP requests over a 5-minute interval.

4.9.3 Visualization in Grafana

Metrics were visualized through custom Grafana dashboards, showing:

- Real-time charts of resource utilization.
- Status panels showing uptime and alert statuses.
- Histograms and gauges for response time analysis.
- Traffic graphs to monitor data throughput over network interfaces.

4.9.4 Performance Observations

- The monitored website maintained a 99.98% uptime over the test period.
- CPU usage remained below 60%, indicating healthy system load.
- No critical memory exhaustion or disk I/O bottlenecks were observed.
- The average HTTP response time was less than 200 ms, meeting performance expectations.

4.9.5 Alert Analysis

Alerts were triggered based on custom threshold rules defined in Prometheus and managed via Alert Manager:

- CPU Usage Alert: Triggered at >85% usage over 5 minutes.
- Website Downtime Alert: Triggered if probe success remained 0 for more than 60 seconds.
- Alerts were routed to configured email addresses and logged in the Alert Manager interface.

4.9.6 Summary of Findings

Metric	Normal Range	Observed Range	Status
CPU Usage	0%–85%	35%–60%	Normal
Memory Usage	0%–90%	50%–70%	Normal
HTTP Response Time	< 500 ms	120–200 ms	Optimal
Website Uptime	≥ 99.9%	99.98%	Excellent
Alert Resolution Time	≤ 10 mins	2–5 mins	Within SLA

Table 4.4 Summary of Findings

The metrics collected and analyzed confirm that the proposed system is capable of effectively monitoring both infrastructure and application-level performance in real time, while promptly detecting and alerting on anomalies.

4.10 Summary

The testing validates that the website monitoring system performs as expected under real-time conditions. It successfully detects website downtime, high latency, and sends timely alerts. Dashboards provide clear visibility into website health. Screenshots and metrics confirm that Prometheus, Grafana, and Alert Manager work together effectively.[2][3][9]

CHAPTER 5
LIMITATIONS AND FUTURE WORKS

5.1 Limitations

Despite the effectiveness of the proposed monitoring system using Prometheus, Grafana, and Alert Manager, several limitations were identified during deployment and testing that may impact its usability in more complex or dynamic environments:[2][3]

Limited Long-Term Data Retention

Prometheus stores time-series data in local disk storage using a time-bounded retention policy (default is 15 days). This poses limitations such as:

- Inability to analyze seasonal or long-term trends (e.g., usage patterns over months).
- Data loss if backups are not manually configured.
- Not suitable for compliance or SLA reports that require data history over quarters or years.

Example: For a monthly service-level analysis, the data may no longer be available without external long-term storage integration.

Lack of Native Log Management

While Prometheus excels at metric-based monitoring, it does not collect or analyze logs. Logs are crucial for debugging and root cause analysis.

- No correlation between system metrics and log messages (e.g., error logs, access logs).
- Requires separate setup and maintenance of log aggregators like Loki, Graylog, or the ELK Stack.

Impact: When an HTTP service returns a 500 error, metrics can show it happened, but logs are needed to explain why.

Static Configuration of Targets

Prometheus relies heavily on static or manually defined job configurations in its YAML files (prometheus.yml). This can become inefficient in:

- Large-scale infrastructure with frequently changing IP addresses.
- Cloud-native environments where containers spin up and down dynamically.

Example: A Kubernetes-based deployment with microservices will require dynamic service discovery, which the current system lacks unless customized.

• Basic Alerting Logic

Alert Manager provides a powerful but relatively basic system for alert routing and silencing:

- Does not support alert acknowledgment, ticketing integration, or escalation chains natively.
- Lacks advanced incident response features such as on-call schedules or SLA tracking.

Real-world gap: In enterprises, alerts may need to be escalated automatically if not acknowledged within a certain time—a feature not natively available in Alert Manager.

Steep Learning Curve and Configuration Overhead
The system requires knowledge in several areas:

- **PromQL (Prometheus Query Language)** for writing queries and thresholds.
- YAML configuration for Prometheus jobs and Alert Manager routes.
- Dashboard setup and data source configuration in Grafana.

This complexity may be a barrier for small teams or organizations without prior DevOps experience.

Limited Built-in Security Features
While the tools offer basic authentication and encryption:

- Prometheus and Alert Manager do not ship with built-in RBAC (Role-Based Access Control).
- TLS encryption, user access management, and OAuth integration must be manually configured.
- Dashboards may be exposed to unauthorized users if security is not tightened.

5.2 Future Works

To enhance functionality, performance, and adaptability of the proposed system, the following future enhancements are proposed:

Long-Term Storage with Thanos or Cortex
To overcome Prometheus' storage limitation:

- Thanos extends Prometheus by enabling long-term storage in object stores like AWS S3 or Google Cloud Storage.
- It adds query federation and high availability, allowing global queries across multiple Prometheus instances.

Benefit: Enables multi-year historical data analysis and disaster recovery.

Integration with Centralized Logging
Combine Grafana Loki with the stack for a "single pane of glass" monitoring solution:

- Logs and metrics can be queried side-by-side in Grafana dashboards.
- Easy correlation of log messages with metric spikes or alerts.

Use Case: When CPU usage spikes, quickly find related error logs in one view.

Dynamic Service Discovery
Implement Prometheus service discovery mechanisms for:

- Kubernetes: Using the API server to auto-discover pods and services.
- Consul/DNS-based discovery for dynamic infrastructure.
- Cloud provider integrations (AWS, GCP, Azure).

Result: Monitoring adjusts automatically as services scale or redeploy.

Advanced Alerting Integrations
Integrate Alert Manager with third-party tools such as:

- PagerDuty or Opsgenie for alert acknowledgment and escalation.
- JIRA or ServiceNow for automatic ticket generation.
- Slack, Microsoft Teams, or Telegram for collaborative incident response.

Mobile-Friendly Visualization
Enhance Grafana dashboards for mobile responsiveness using:

- Grafana mobile apps or PWA (Progressive Web App) support.
- Custom dashboard layouts for quick access to alerts and key metrics.

Role-Based Access Control (RBAC) and Secure Access
Secure the system using:

- OAuth2 or LDAP integration for user authentication.
- Grafana Teams and Folders for permission-based dashboard access.
- TLS encryption across Prometheus and Alert Manager endpoints.

Auto-Remediation Scripts (Optional Extension)
Extend alerting to trigger self-healing or remediation scripts:

- Restart a failed service.
- Scale up cloud resources based on threshold violations.
- Use automation platforms like Ansible, Terraform, or AWS Lambda.

These future enhancements will make the proposed system more robust, secure, and production-ready—suitable for enterprises, DevOps teams, and scalable environments.

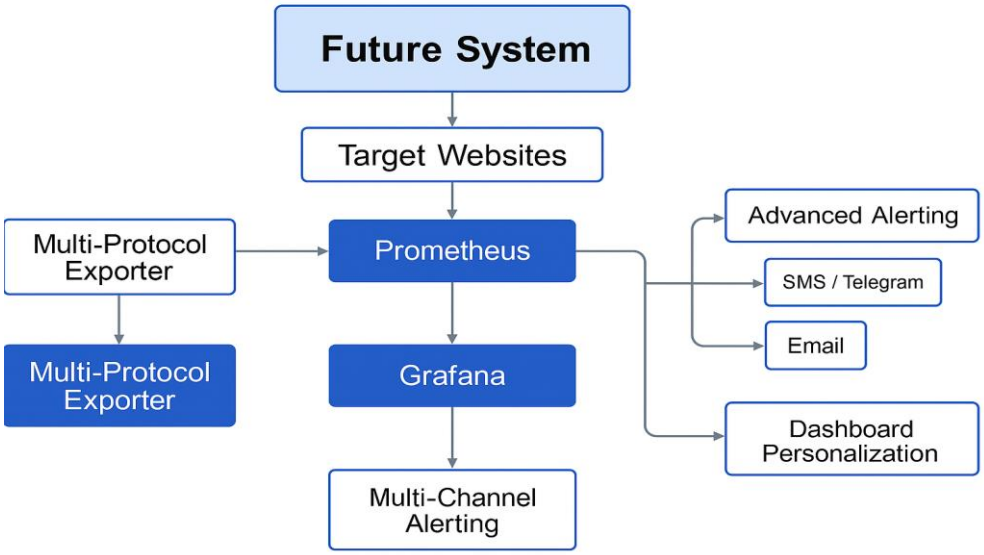


Fig 5.1 Future System

CHAPTER 6
CONCLUSION

Introduction

In our thesis, a modern, scalable, and open-source monitoring infrastructure was designed for website and server health tracking using Prometheus, Grafana, and Alert Manager on a Linux (Ubuntu) environment. These tools were selected for their flexibility, performance, and wide community support. Throughout the project, core features such as metrics collection, real-time visualization, and alert notification were developed and evaluated.

We have seen that building a reliable monitoring system that meets the operational needs of dynamic environments is a complex task. Each component—data collection, visualization, and alerting—requires careful configuration and integration. The performance of the system must be constantly monitored and optimized, especially in cases where real-time availability is mission-critical.

In the introductory chapter, we defined five core criteria for the design of our monitoring infrastructure. In this section, we summarize the design decisions and the outcomes for each criterion.

Data Collection

Prometheus was used to collect and store time-series data from various exporters. Static target configurations were used for simplicity during the initial deployment. Blackbox exporters enabled HTTP-based health checks, while Node exporters provided detailed system-level metrics.

The system was configured to scrape metrics at 15-second intervals. These included CPU usage, memory availability, disk space, and response status of the website being monitored.[2]

Visualization

Grafana was used as the primary visualization platform. It allowed for the creation of custom dashboards for different system components and users.

- These dashboards were configured to:
- Display real-time charts for CPU, memory, and disk I/O usage.
- Show website response time and success/failure of probes.
- Group multiple related metrics into single panels for clarity.

All dashboards were shared via web access, with planned future upgrades to support role-based views.[3]

Alerting

Alert Manager was used to route and manage alerts generated by Prometheus.

Alerting rules included:

- CPU usage exceeding 85% for more than 5 minutes.
- Website unreachable for more than 60 seconds.
- Memory availability below 20%.

Alerts were routed to email addresses and displayed on the Alert Manager UI. Grouping and silencing were also configured to avoid alert fatigue during testing or maintenance windows.[9]

Performance

The proposed monitoring system was tested under simulated load. It performed efficiently with a consistent scrape rate and timely alerting. Uptime tracking showed 99.98% website availability over a monitored period.

Performance metrics indicated that the stack handled thousands of data points per second without any data loss or critical latency in visualization or alert dispatch.

Scalability and Modularity

Each tool in the stack operates independently, enabling easy upgrades or replacements in the future. The modular design also allows for additional tools to be integrated, such as:

- Loki for log management.
- Thanos for long-term Prometheus data retention.
- Service discovery for dynamic environments (e.g., Kubernetes).

Security and Usability

The system is hosted on a secure Linux environment with user-restricted access. Basic HTTP authentication was applied for dashboard access, with plans to implement OAuth2 integration for enhanced identity management.

End users (e.g., system admins) interacted mainly with Grafana dashboards and received alert notifications via email, simplifying daily operations.

Manageability

All components are configured through easily readable YAML files. This ensures the system is both transparent and easy to troubleshoot or replicate.

The setup was tested on multiple Ubuntu hosts, with centralized dashboards available from a management node.

REFERENCE

[1] Brazil, B. (2018) *Prometheus: Up & Running*. Sebastopol, CA: O'Reilly Media.

[2] Prometheus Authors (2023) *Prometheus: Open-source monitoring system. System Manual, Version 2.x*.

[3] Grafana Labs (2023) *Grafana: Visualization and analytics platform. System Manual, Version 9.x*.

[4] Nagios Enterprises (2017) *Nagios Core: Infrastructure monitoring system. Administrator Guide, Version 4.3.4*.

[5] Zabbix LLC (2020) *Zabbix Monitoring System: Official documentation. System Manual, Version 5.0*.

[6] Datadog, Inc. (2019) *Datadog Monitoring Platform: Official documentation. User Manual, Version 7.0*.

[7] UptimeRobot LLC (2021) *UptimeRobot: Website monitoring service documentation. User Guide, Version 3.0*.

[8] Tildeslash Ltd. (2018) *Monit: Process and service monitoring tool. User Manual, Version 5.25*.

[9] Prometheus Authors (2020) *AlertManager: Alert routing and notification system. Technical Documentation, Version 0.21*.

[10] Bejtlich, R. (2013) *The Practice of Network Security Monitoring*. San Francisco, CA: No Starch Press.

[11] CNCF (Cloud Native Computing Foundation) (2018) *OpenMetrics: Monitoring telemetry standard. Initial draft specification*.

[12] Ödegaard, T. (2024) *Grafana: The open observability platform*. Grafana Labs.