# Software Requirement Specification & Implementation

for

Infrastructure Monitoring Dashboard
(Capacity Measurement)

Prepared By:

Akshay Patil

Kohinoor Chatterjee

Madhav Baveja

Shubham Mote

Pankaj Kumar

## Table of contents

# 1. Introduction

This section gives a scope description and overview of everything included in this document.

## 1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the “Infrastructure Monitoring Dashboard” system. It will illustrate the purpose and complete declaration for the development of the system. It will also explain the system constraints, interfaces, and interactions with other external applications and devices. This document is primarily intended to be proposed to authorized guides for its approval and a reference for developing the first version of the system for the development team.

## 1.2 Scope

Understanding the state of infrastructure and systems is essential for ensuring the reliability and stability of services. Infrastructure monitoring system consists of a dashboard that is capable of sensing and interpreting infrastructure metrics and making sensible decisions, with minimal human input. The system will need a data source server to provide metrics for monitoring and time series database to store metrics and to allow querying data whenever needed by dashboard. Implementing effective Infrastructure monitoring offers the benefits such as Increased security, Increased awareness of network infrastructure problems, Increased server services and application availability, Fast detection of network outages, protocol failures and failed processes, services, etc.

## 1.3 Problem Statement

To design a system that provides real time monitoring of system metrics, log files, event logs, service logs, and system logs on Windows servers and Linux servers. This system should be capable of displaying real time system metrics data graphically on dashboard, providing historical metrics visualization and alerting when a metrics log pattern is detected.

## 1.4 References

- Grafana Documentation (https://grafana.com/docs/)
- Prometheus Documentation (https://prometheus.io/docs/introduction/overview/)
- Available examples

## 2. Specific Requirements

### 2.1 Functional Requirements

The functional requirements for the project include:

- Infrastructures- A windows and a linux server would be the two infrastructure platforms that we would be monitoring.
- Applications- The following would be the web application servers that we would be monitoring: a node server, a spring server, a golang server.
- Data Collection Server- A prometheus server which would collect data from all the different infrastructures and applications that we need to monitor.
- Dashboard Server- A grafana server which would read the Prometheus time series database and would help us create graphical dashboards and alerts of the same.

### 2.2 Non-Functional Requirements

- Data collection: Data is collected from all the different infrastructures and applications that we would be monitoring.
- Dashboard: The collected data is be used to create graphical dashboards to monitor the systems in place
- Alert system: The grafana dashboard is configured to alert the maintainers (via email or slack) whenever any critical parameters have abnormal values.

### 2.3 Hardware and Software Requirements

**Hardware:**

- Linux and Windows t2.micro instances
  - 1 vCPU
  - 1GB RAM
  - 8GB SSD

- Heroku App instances (for other hosting):
  - 500MB storage
  - 600Hours/month

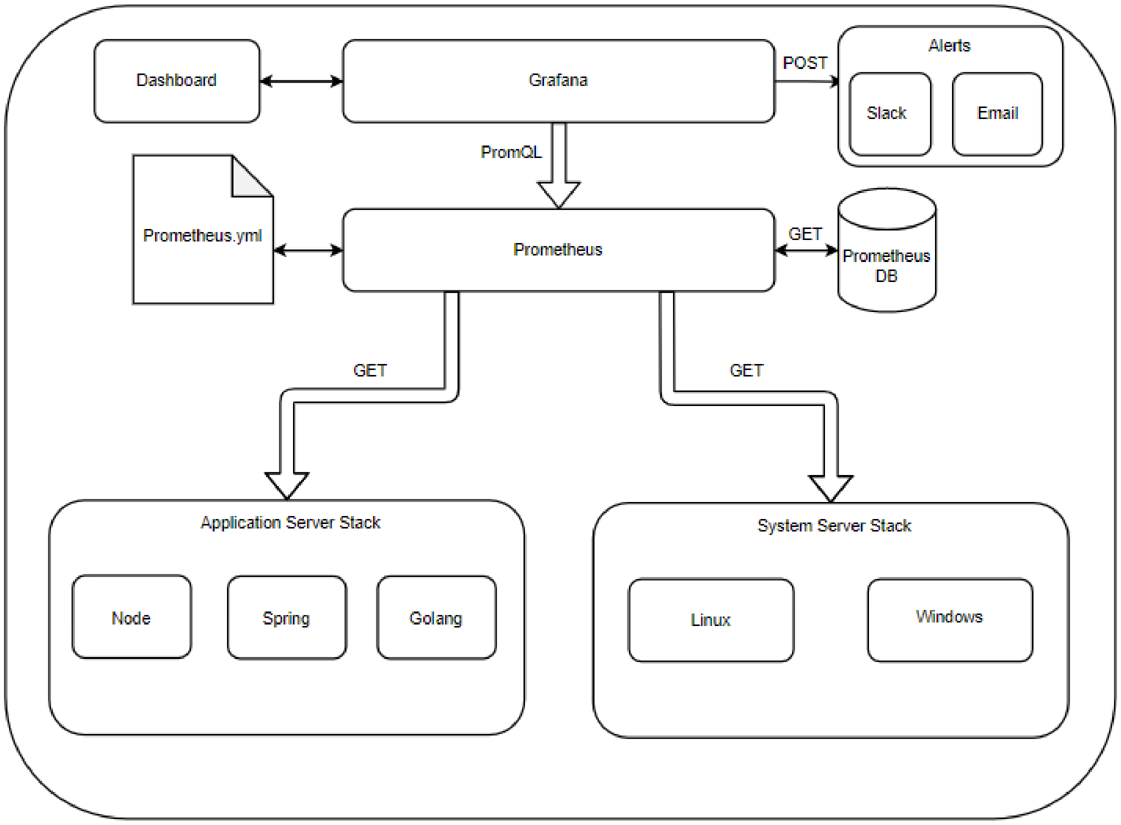
**Software:**

- Prometheus
- Grafana
- Node.js with Express
- Golang
- Spring Boot
- Git
- Github

## 3. Software Design Specification

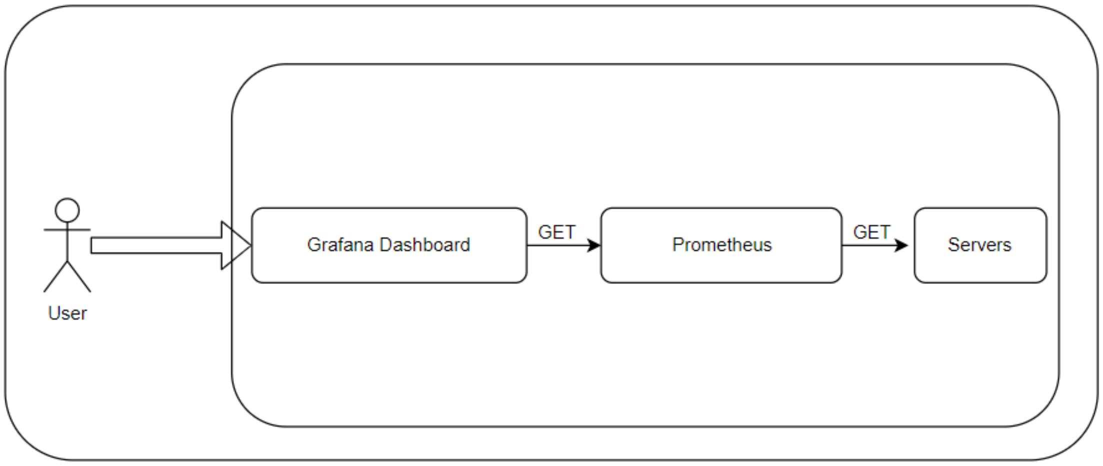
### 3.1 Architecture Diagram

An architecture diagram is a graphical representation of a set of concepts, that are part of an architecture, including their principles, elements and components.



The above diagram shows the flow of the project that we have worked on. The architecture can be broken down into three main components, that is, Grafana, Prometheus and Server. The Server contains the data that needs to be scraped by Prometheus. This server could be a system server or an application server. In our case we have two system servers, namely, Linux and Windows; and three application servers, namely, Node, Spring and Golang. The server that needs to be monitored hosts a page that contains the metrics of the server of application. The only requirement is that the data should be in a format that is required by Prometheus.
Further, we specify the address and a few more details of the target in the Prometheys.yml file. This file is used as a guide by Prometheus. Prometheus will scrape data from the given target in intervals specified in the yml file. Prometheus also hosts it's on server's metrics.
Additionally, using PromQL this data is accessed and used by Grafana. In Grafana we can access this data and create multiple Dashboard configurations. Grafana also allows us to create alerts and send their status on various platforms such as Slack, Email, etc.

### 3.2 Use Case Diagram



A use case diagram is a dynamic or behaviour diagram. The diagram models the functionality of a system from a user's point of view.

Referring the above diagram, we can see that the user accesses the dashboard which is made in Grafana.

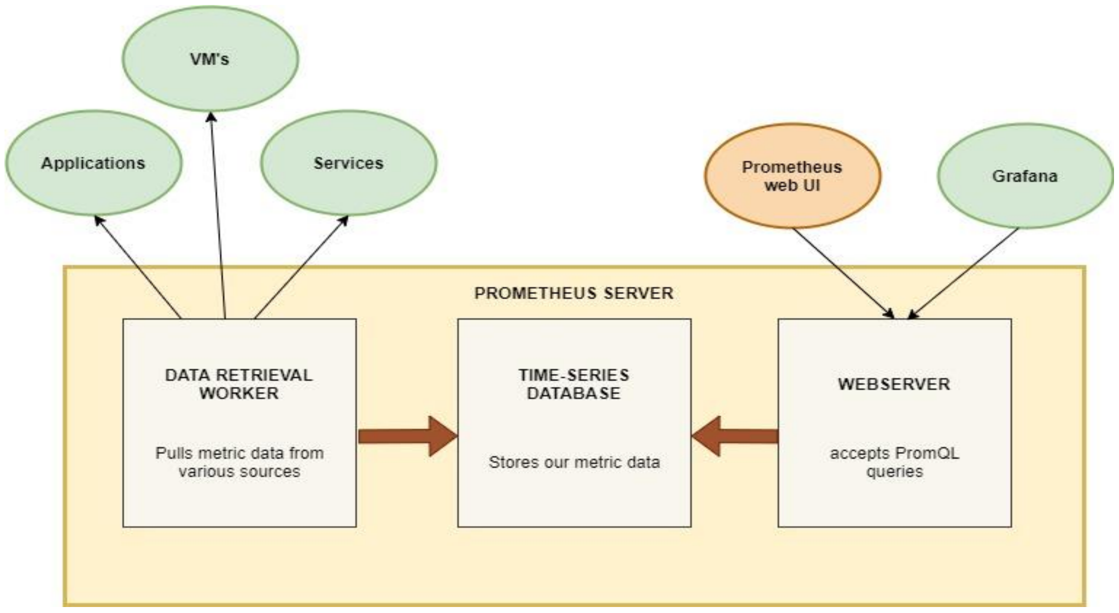
## 4. Implementation:

### 4.1 Introduction:

The combination of Grafana and Prometheus is a common monitoring stack used by DevOps teams to build up a monitoring system i.e for storing and visualizing time series data. Prometheus acts as the storage backend and Conveniently, exposes a wide variety of metrics that can be easily monitored. Open source grafana acts as the interface for analysis and visualization. Like any server running processes on a host machine, there are specific metrics that need to be monitored such as used memory and storage as well as general ones reporting on the status of the service. Prometheus, a system and service monitoring system, collects metrics from predefined targets via a pull model by scraping metrics from HTTP endpoints on these targets. Targets can be found by Service Discovery or manually configured to pull data from endpoints of your application like an API server. Grafana supports lots of data sources such as CloudWatch, Stackdriver, Elasticsearch, and sure, Prometheus. After connecting Grafana with the data sources, we can create a dashboard and panel manually, or import various dashboards shared on the official website.

To create a full example of a monitoring dashboard with Prometheus and Grafana, we need three components:

1. Metrics Exporter: usually a server which handles requests.
2. Prometheus Server: periodically pull metrics from exporters/jobs
3. Grafana: query prometheus server to generate visual dashboards

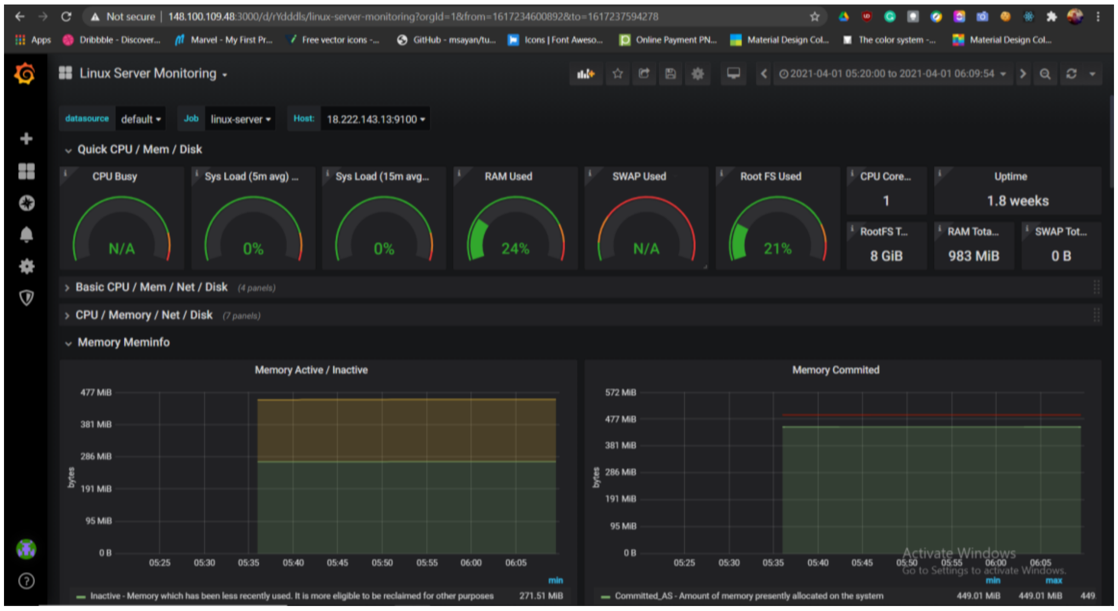


With respect to this project, we monitored some target servers using Grafana and Prometheus. The screenshots of some of the monitoring dashboards are attached below:

### 4.2 Linux Server Monitoring:

The **node_exporter** is an official package that should be installed on Linux servers to be monitored. It exposes multiple hardware and OS metrics, which will be pulled by Prometheus and eventually visualized on Grafana.
Steps to follow:

- Configure Node Exporter as a service on Linux server.
- Start Node Exporter
- Add a new job in the prometheus.yml file.
- Add Prometheus data source and set the URL as Prometheus server's IP with port 9090
- Create a new dashboard or use the existing node exporter dashboard, which is present on Grafana's official website to monitor Linux server metrics.select the Prometheus data source and Import.

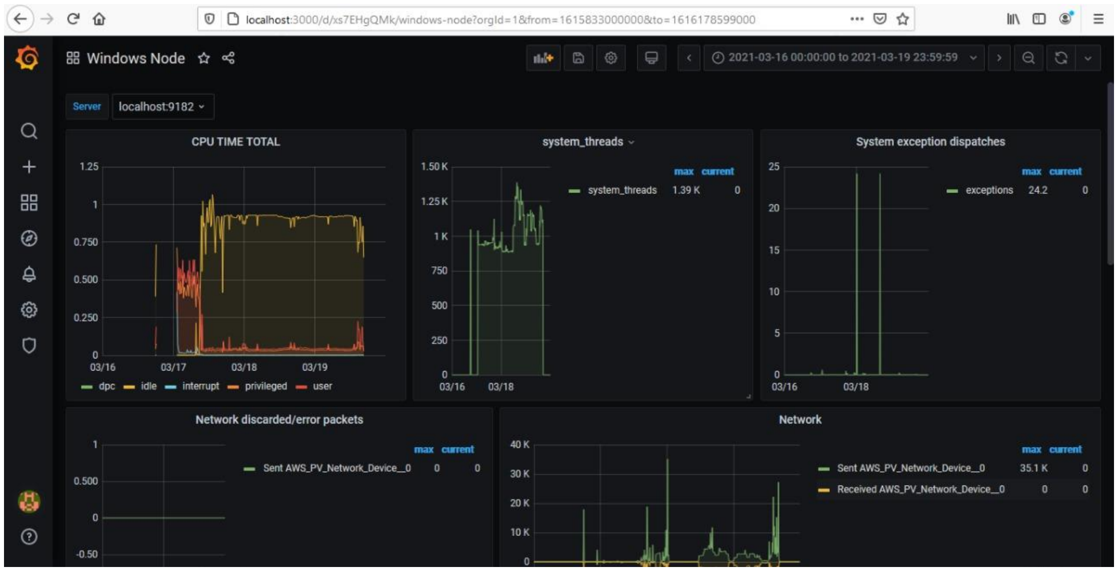


## 4.3 Windows Server Monitoring

The **WMI exporter** is an official package that should be installed on Windows servers to be monitored. The WMI exporter will run as a Windows service and it will be responsible for gathering metrics about the Windows system, which will be pulled by Prometheus and eventually visualized on Grafana.

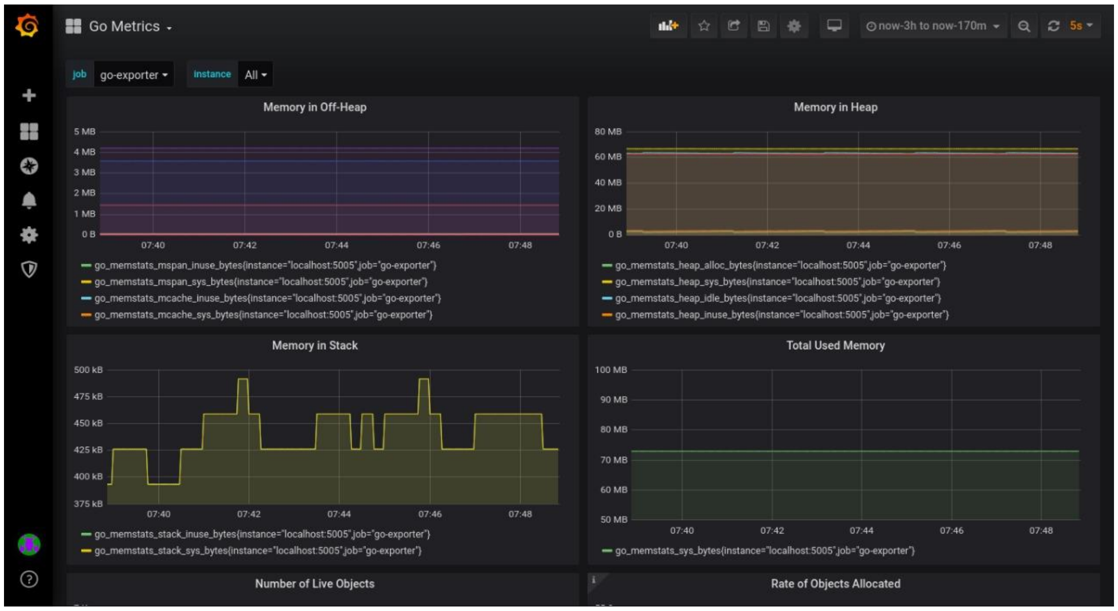
Steps to follow:

- Configure WMI Exporter as a service on Windows server.
- Start Node Exporter
- Add a new job in the prometheus.yml file.
- Add Prometheus data source and set the URL as Prometheus server's IP with port.
- Create a new dashboard or use the existing Windows Node dashboard, accessible via the 2129 ID, which is present on Grafana's official website to monitor Windows server metrics. select the Prometheus data source and Import.



## 4.4 Golang Application Monitoring

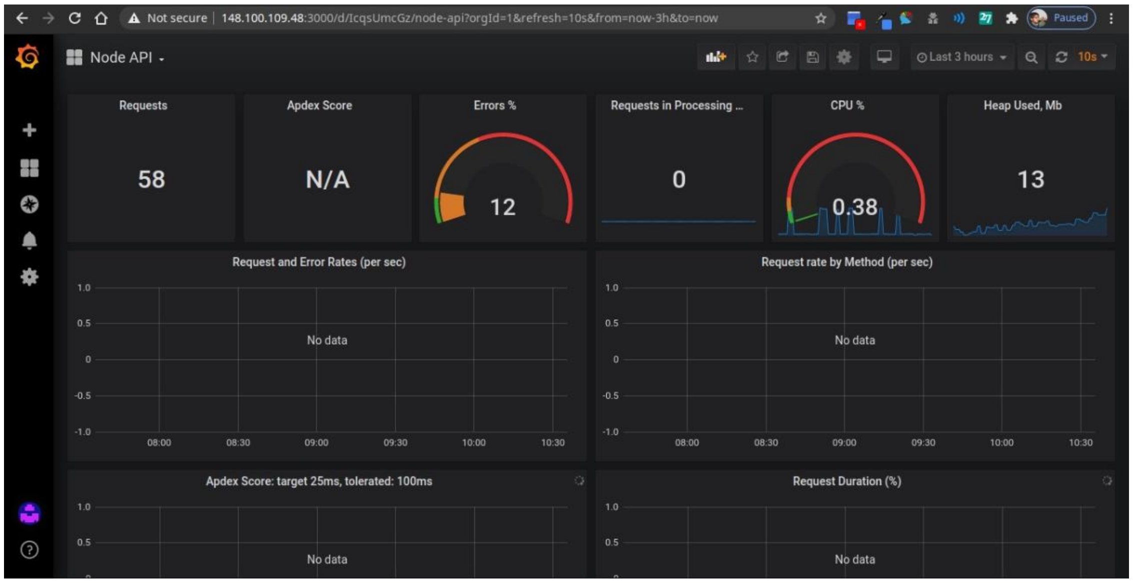
Prometheus has provided a client library for Go, here https://github.com/prometheus/client_golang. There are many insights we can get by exposing app's metrics like requests per second, error rate, average latency etc. We can also set an alert to notify when some metrics go wrong. To provide an HTTP endpoint for prometheus to scrape the metrics. Prometheus has provided the handler. We need to include it in our HTTP server.



Go Metrics
job go-exporter
instance All
now-3h to now-170m
5s
Memory in Off-Heap
go_memstats_mspan_inuse_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_mspan_sys_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_mcache_inuse_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_mcache_sys_bytes{instance="localhost:5005",job="go-exporter"}
Memory in Heap
go_memstats_heap_alloc_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_heap_sys_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_heap_idle_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_heap_inuse_bytes{instance="localhost:5005",job="go-exporter"}
Memory in Stack
go_memstats_stack_inuse_bytes{instance="localhost:5005",job="go-exporter"}
go_memstats_stack_sys_bytes{instance="localhost:5005",job="go-exporter"}
Total Used Memory
go_memstats_sys_bytes{instance="localhost:5005",job="go-exporter"}
Number of Live Objects
Rate of Objects Allocated

## 4.5 Node.js Application Monitoring

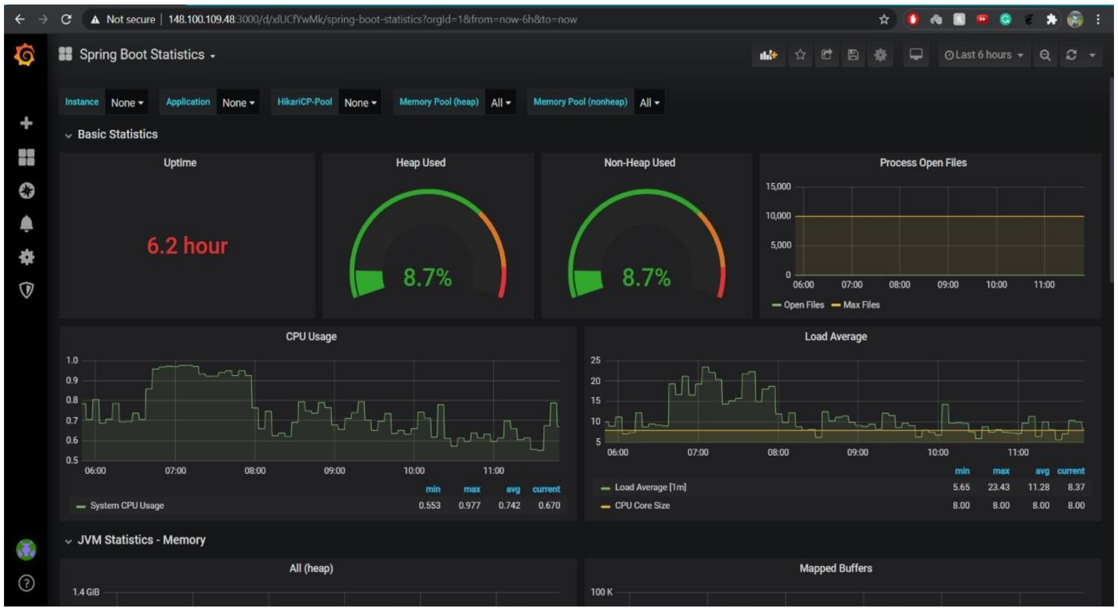
**prom-client** is the most popular Prometheus client library for Node.js. It provides the building blocks to export metrics to Prometheus via the pull and push methods and supports all Prometheus metric types such as histogram, summaries, gauges and counters. To provide an HTTP endpoint for prometheus to scrape the metrics. Prometheus has provided the handler. We need to include it in our HTTP server.



Not secure | 148.100.109.48:3000/d/IcqsUmcGz/node-api?orgId=1&refresh=10s&from=now-3h&to=now
Paused
Node API
Last 3 hours
10s
Requests
58
Apdex Score
N/A
Errors %
12
Requests in Processing ...
0
CPU %
0.38
Heap Used, Mb
13
Request and Error Rates (per sec)
No data
Request rate by Method (per sec)
No data
Apdex Score: target 25ms, tolerated: 100ms
No data
Request Duration (%)
No data

## 4.6 Spring Boot Application Monitoring

Set up a regular Spring Boot application by using Spring Initializr. Add dependency for Actuator. Add dependency for Micrometer. Expose our needed Prometheus endpoint in the application. properties file. After this we can run the application and monitor this application using the Grafana dashboard.

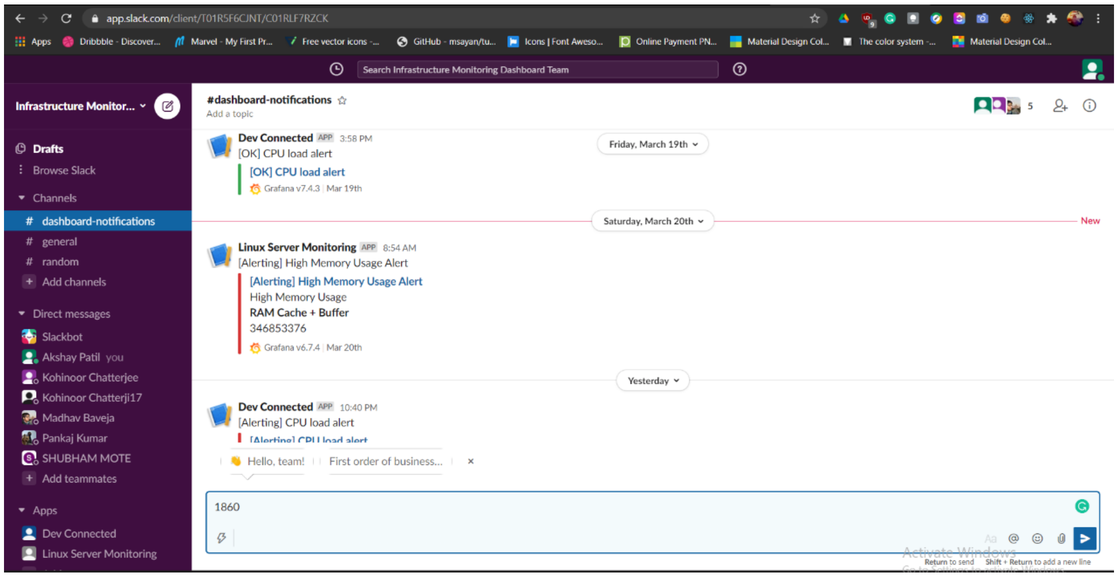


## 4.7 Grafana Alerting with Slack:

Alerting in Grafana allows us to attach rules to dashboard panels. After saving the dashboard Grafana will extract the alert rules into a separate alert rule storage and schedule them for evaluation.
Steps to follow to create slack alerting:

- Setup New Notification Channel. Select Slack from “Type” as we want to send notifications over Slack.
- Create a Slack app with Incoming Webhook enabled. Select the Incoming Webhooks feature, and click the Activate Incoming Webhooks toggle to switch it on.
- Pick a channel that the app will post to, a new entry under the Webhook URLs for our workspace section will be added, with a Webhook URL.
- Complete Setup New Notification Channel: Enter the Name of the channel and paste webhook URL to the Url line.
- Select the panel where an alert is to be created. Add Alert Notification and edit message accordingly.

Below screenshot shows the different alerts were created from different dashboards to the slack channel **dashboard-notification**:



#dashboard-notifications

Friday, March 19th

Dev Connected APP 3:58 PM
[OK] CPU load alert
[OK] CPU load alert
Grafana v7.4.3 | Mar 19th

Saturday, March 20th

Linux Server Monitoring APP 8:54 AM
[Alerting] High Memory Usage Alert
[Alerting] High Memory Usage Alert
High Memory Usage
RAM Cache + Buffer
346853376
Grafana v6.7.4 | Mar 20th

Yesterday

Dev Connected APP 10:40 PM
[Alerting] CPU load alert
[Alerting] CPU load alert

1860

## 5. Future Scope

The project made here is just monitoring selected systems like a Linux server, Windows server, Java Spring Boot Web-server, etc. But we can use it for monitoring any application and any system. As Prometheus is already a well-established tool for storing system monitoring data (time-series data), we can find data exporter for almost every system (language/framework/OS).
If it is not pre-developed for some system, we can easily develop it by exposing an API endpoint and providing metrics data in a fixed format i.e. required by the Prometheus server to store it.
Grafana is also a well-known visualization tool. So, there are a lot of predefined dashboards for different systems. We can easily import them and use them in our interest. But making your own dashboard is also simple with Grafana. You can follow the Grafana documentation for creating a custom dashboard.