PerformanceFlow: Hospital assets monitoring and notification system

Submitted By

Zealous Macwan

18mcei04

DEPARTMENT OF COMPUTER & SCIENCE ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

PerformanceFlow: Hospital assets monitoring and notification system

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Computer Science and Engineering

(Information and Network Security)

Submitted By

Zealous Macwan

(18mcei04)

Guided By

Prof. Parita Oza

DEPARTMENT OF COMPUTER & SCIENCE ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2020

Certificate

This is to certify that the major project entitled as **"PerformanceFlow: Hospital assets monitoring and notification system"** submitted by **Zealous Macwan (18mcei04)**, towards the fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering (Information and Network Security) of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Parita Oza,
Guide and Assistant Professor,
CSE Department,
Institute of Technology,
Nirma University,Ahmedabad

Dr. Sharada Valiveti,
Associate Professor,
Coordinator M.Tech CSE(INS),
Institute of Technology,
Nirma University,Ahmedabad

Dr. Madhuri Bhavsar,
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University,Ahmedabad

Dr. R N Patel,
I/C Director,
Nirma University,Ahmedabad

Statement of Originality

I, **Zealous Macwan**, **18mcei04**, give undertaking that the Major Project entitled **"PerformanceFlow: Hospital assets monitoring and notification system"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering (Information and Network Security)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. **PerformanceFlow: Hospital assets monitoring and notification system** is a solution under development at **Philips Innovation Campus, Bangalore** and I have carried out the work under the guidance of Philips Architect. I give assurance that no attempt of plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Parita Oza

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Parita Oza**, Assistant Professor, Computer & Science Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Madhuri Bhavsar**, Hon'ble Head of Computer & Science Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. R N Patel**, I/C Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Science & Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Zealous Macwan

18mcei04

Abstract

Performance Flow project is about developing solution to manage, monitor, maintain and analyse assets of the Hospital. By developing this solution it will help medical stack-holders to improve the overall management of assets with useful insights from the assets and better healthcare satisfaction to patients. Medical staff can easily find the required medical devices without wasting much time on searching for the assets, which will also improve their satisfaction on work. Support and Maintenance staff can monitor and diagnose the IT infrastructure used for asset tracking and management. It will have alerting and notification to report any abnormal performance of the resources and related teams will be notified. As these assets sends valuable information as per their capabilities, these information will be analysed so that assets can be optimally utilized. Patients overall flow in the hospital for required treatment can be monitored and can be improved by finding out where the timing can be improvised.

Contents

Certificate **iii**

Statement of Originality **iv**

Acknowledgements **v**

Abstract **vi**

List of Figures **xi**

List of Tables **xii**

1 Introduction **1**

1.1 Problem Statement 1

1.2 Objective of Study 2

1.3 Scope of Work 3

2 Literature Survey **4**

2.1 Docker 4

2.2 Prometheus 5

2.3 Grafana 6

2.4 CAdvisor 7

2.5 WMI Exporter 7

2.6 BlackBox Exporter 8
2.7 Proprietary System 9

3 Implementation **10**
3.1 High Level Architecture 10
3.2 Data flow between tools 12
3.3 Spinning up Docker Containers 13
3.4 Setup Grafana source and Visualization Panel 15
3.5 Notification 17
3.5.1 Setup Alerts 18
3.5.2 Setup Notification channel 19
3.5.3 Setup Multiple Notification channel 20
3.5.4 Setup notification channel for rule 21
3.6 System Hardening 22
3.7 Docker Swarm 23
3.8 Configuration of Docker images 24
3.8.1 Configuration by passing $Env to container 24
3.8.2 Grafana configuration 25
3.8.3 Prometheus configuration 26
3.8.4 cAdvisor configuration 27
3.9 Build and Deploy Flow 28
3.9.1 Normal Deployment flow 28
3.9.2 Build Script to auto build customized images 29
3.9.3 Push Images to private registry 30
3.9.4 Deployment of the images 31
3.9.5 Sample build script 32
3.9.6 Sample deploy script 33
3.10 Notification integration to proprietary system 34

4 Screenshots of PerformanceFlow **36**

4.1 Sample dashboard of PerformanceFlow System resources . . . 36

4.2 Sample dashboard of PerformanceFlow System APIs 37

4.3 Container Monitoring 38

4.4 Database Monitoring 39

4.5 Windows Process Monitoring 39

4.6 API Monitoring 40

4.7 Tags Monitoring 41

4.8 Notifications 42

4.8.1 Notification on Slack channel 42

4.8.2 Notification on MS Team 42

4.8.3 Notification by Email 43

5 Future Work and Conclusion **44**

5.1 Future Work 44

5.2 Conclusion 44

References **45**

List of Figures

2.1 Docker Vs Virtual Machine Architecture 5
2.2 Grafana Visualization Panels 7

3.1 High Level Architecture of Monitoring and Alerting System . . 10
3.2 Grafana running as a container 14
3.3 Grafana Query Editor for Prometheus 15
3.4 Grafana Visualization Editor 16
3.5 System's working flow with notifications 17
3.6 Grafana Alert creator 18
3.7 Notification Channel 19
3.8 List of Notification Channel 20
3.9 Rule Notification Channel 21
3.10 Container Deployment 28
3.11 Build Script to build all images 29
3.12 Push Images to private registry 30
3.13 Pull and Deploy Images 31
3.14 Container Deployment 34

4.1 Dashboard of PerformanceFlow System 36
4.2 Dashboard of API monitoring 37
4.3 Containers Monitoring Dashboard 38

4.4 Containers CPU Usage Graph Panel 38
4.5 Containers Memory Usage Graph Panel 38
4.6 Database Status 39
4.7 Windows Process Status 39
4.8 API endpoint Monitoring 40
4.9 API Response time 40
4.10 Tags Activity 41
4.11 Tags per day 41
4.12 Notification on Slack channel 42
4.13 Notification on MS Team 42
4.14 Notification by Email 43

List of Tables

2.1 Comparison between Docker and Virtual Machine 5

Chapter 1

Introduction

1.1 Problem Statement

In Healthcare facilities, patient volume is increasing day by day with expectation of higher patient experience and satisfaction. To fulfill these demands, hospital facilities needs to streamlined workflow to reduce number of steps to complete the medical events. One of the key components for efficient workflow is tracking and management of assets. When medical staff are under higher workload, they might be wasting valuable time to search for medical devices. So to reduce this time, asset tracking and management is required. For these Hospitals are using different kind of RF-ID tags which can be attached to assets and real time location of assets can be fetched and staff can reduce the timing of searching for the assets. Maintenance and inventory department can also track assets to get insights and act accordingly. Similar way, staff and patient can have these kind of tags which will send real time location of the respected person. These tags will provide useful information for patient monitoring. As these tags are tagged with particular patients, all the upcoming and historical medical events of the patient can be tracked

and it can be useful to improve the workflow like where the patient currently is, where he required to go next for medical event like scanning or to meet the doctor etc. These will also give insights like how many patients are waiting for the same kind of medical treatment as currently other patient is undergoing the treatment. In situation like medical triage when the number of incoming patients are more than capacity of the medical center or ER, efficient workflow is required to meet the demands.

1.2 Objective of Study

Objective of this project is to develop the system which will be used as monitoring and notification of the assets, IT infrastructure,process and services supporting the working of assets, tags. Also to develop notification and alerting system to raise alerts and notification when any threshold rules for values are violated. These system will have dashboards to monitor and having features to view real-time as well as historical data based on capabilities. Second is to analyse the data and find useful insights out of the collected data to improve efficiency of workflow, utilization of resources, managements of assets, monitoring of patients, staff management etc.

- Develop monitoring system having visualization of real time and historical data.
- Develop Notification and Alerting system with capabilities to integrate with existing platforms.
- Deployment package of the system for easy installation

1.3 Scope of Work

As this solution is for monitoring,alerting,notification,analysing the resources only, this solution is not intended to be used for clinical decision making. This solution is not used for diagnostic purpose and has no connection to operation room or surgical use.

This solution is not for automatic resolving of the issues for which alerts are raised, respected entity to whom the notification is sent or having responsibility of the work need to act to resolve issues.

Chapter 2

Literature Survey

To implement required system using available tools and technology, it's working and understanding is required which has been carried out. Feasibility of the tools and technology to implement the system also considered. Below are the tools and technology which has been surveyed and used for the development.

2.1 Docker

Docker[1] is a tool which allows developer to easily create, deploy and run applications by using containers[2]. Docker uses OS level virtualization to wrap application in container. Containers are packages of application, libraries and its dependencies. By wrapping up all required resources in single package it allows developer to ship it out easily. Containers are isolated from other containers but have their own software, libraries and configuration files and can communicate with each other using well defined channel.

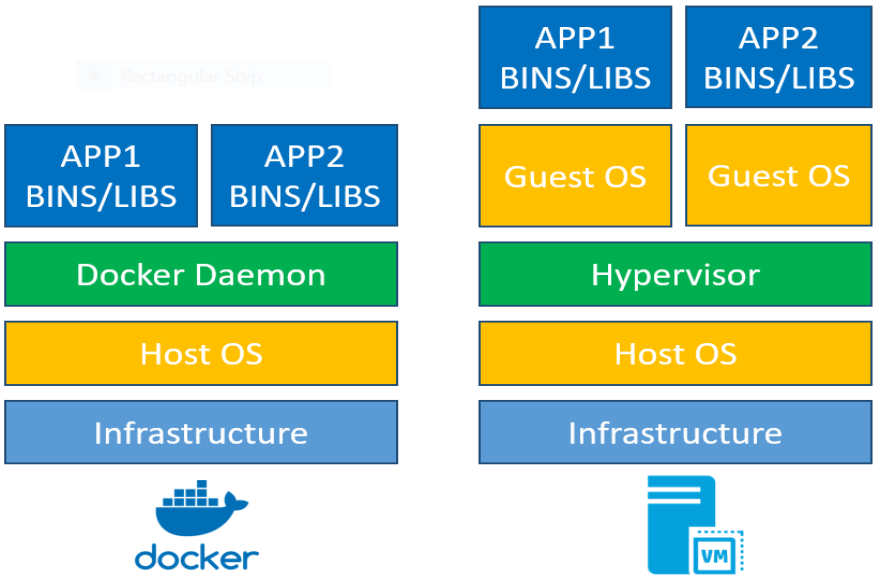

Figure 2.1: Docker Vs Virtual Machine Architecture

	Docker	VM
Process Isolation	OS level	HW level
OS	Shares	Separate per VM
Boot Time	few seconds	few minutes
Storage Space	in MBs	in GBs
Memory	Allocated as required	Relatively High
PreBuilt Availability	Easily Available	Difficult
Creation Time	few seconds	Relatively longer
Resource Usage	Low	High

Table 2.1: Comparison between Docker and Virtual Machine

2.2 Prometheus

Prometheus[4] is a software used for event monitoring. It stores event information in time series database providing higher dimension for events. It

is built on HTTP pull model means it will itself pull metrics over http. For adding source from where it will pull values is added as job in configuration file where it requires target, configuration and time interval at which it need to be scrapped.
One of the alternative of Prometheus is Graphite but it's not full enriched as Prometheus. Prometheus having client libraries, own alert manager,many exporters and flexible query language [5].

Sample yml file :

```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
  - job_name: 'cadvisor'
    static_configs:
    - targets: ['localhost:3000']
```

2.3 Grafana

Grafana[6] is powerful visualization tool, which is used for creating dashboards for visualizing data. It provides many type of visualization like Graph, Singlestat, Table, Text etc. It also provide many popular data-source from where the data can be read.

One of the other available tool for visualization is Kibana[7], which is primarily used for analyzing log messages generated by sources. Kibana does not support alerting feature which is one of the requirement for current system.

Figure 2.2: Grafana Visualization Panels

2.4 CAdvisor

Container Advisor[8] is used to monitor the resource usage and performance characteristics of running containers. CAdvisor daemon collects, aggregates, processes, and exports information about running containers. It keeps resource parameter, usage, network statics of each containers isolated from others. It also exports the metrics over HTTP to be scrapped.

2.5 WMI Exporter

WMI exporter[10] uses Windows OS wmi services to get insights of the system like memory, storage usage,process and services status. WMI exporter collects these details and exposes as metrics to http port which can

be scrapped by prometheus. In the current requirement this will be used to monitor windows systems and it's resources supporting our development. Sample wmi command with parameter :

```
.\wmi_exporter.exe --collectors.enabled "process"
--collector.process.processes-where "Name LIKE 'skype%'"
```

2.6 BlackBox Exporter

Blackbox exporter[9] used to do blackbox probing of endpoints over HTTP, HTTPS, ICMP etc. It will also export metrics over HTTP to be scraped. We can configure the different prober in configuration files. It can also be configured for different parameters like certificate verification, timeout, IPv4 and IPv6 address.

Sample yml file :

```
modules:
  http_2xx:
    prober: http
#   timeout: 60s
    http:
      method: GET
      preferred_ip_protocol: ip4
      tls_config:
        insecure_skip_verify: true
```

2.7 Proprietary System

One of the requirement of the project is that, notification from the monitoring and notification system should be able to integrate with existing proprietary system, which is already in use for internal secure communication and notification, because it does not depend on external tools it is more reliable for internal communication. To achieve this integration,the working, flow and requirements of the system has been explored and integration with the monitoring and notification system has been carried out. Which includes understanding of the properties passing to the portal and rule management and email notification functionality.

Chapter 3

Implementation

3.1 High Level Architecture

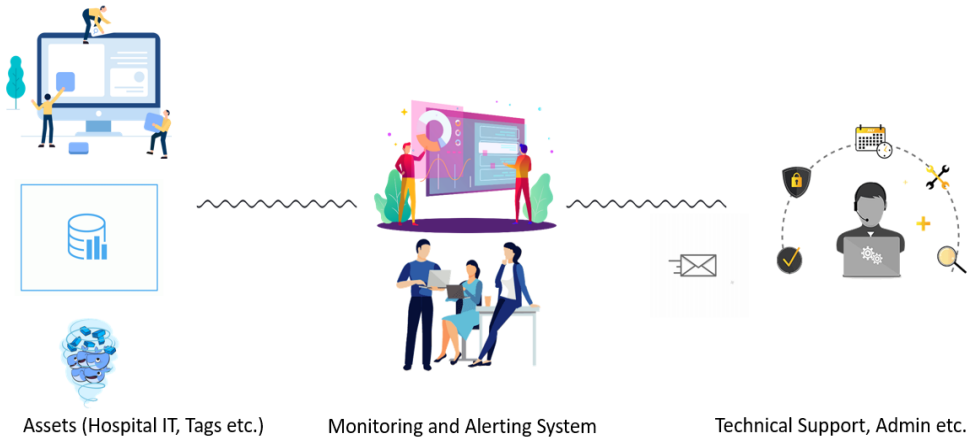

Figure 3.1: High Level Architecture of Monitoring and Alerting System

As shown Figure 4.1 is high level architecture of the system. Here assets are hospital IT infrastructure, supporting applications and services, tags and other medical devices. As these assets needs to be monitored, their status and value needs to be exported. Monitoring and Notification system then

will scrape event data from the sources and will store,process and display to the users by mean of dashboards. Which can be accessible over HTTP on browser. Dashboards are collection of different visualization panels like graphs, gauges and status-map. System will also evaluate collected values of the assets against threshold limit as per rule and will send notification if rules are violated. For any notification to be sent it will send it as per configuration to the respected channel like email, slack or ms team. As shown in figure sources, which generates metrics and the monitoring system are isolated and they are communicating using HTTP channel. Notification part of the system sends notification with required values to the support team, which will take action to analyse and resolve the issues for which alerts are generated.

3.2 Data flow between tools

Exporters: It collects data from the assets and generates metrics which can be scrape by Prometheus. In current system, Blackbox exporter, cAdvisor are the exporters.

Monitor : Prometheus is a monitoring system, having time series database to store collected data. It also provides query language to slice and dice collected data for better analyse and visualization.

Visualizer : Grafana provides a powerful visualization of the data using graphs, gauges, text, status-map etc. Grafana can get data from multiple popular data sources like MySQL, MS SQL Server, Prometheus etc. For the requirement of the having dashboards to visualize our assets values, Grafana also provides feature to separate collection panels into dashboards.

Notifier : Grafana also provides a feature to create alerts and notifications, which is used to fulfill our requirement of notification.

Data flow starts from the exporters, Exporters are the one which collects values from assets and makes it available on http endpoint in form of metrics. Next, These endpoints needs to be configured in Prometheus which is called targets. On frequent time interval Prometheus then pulls metrics from these targets and stores in time-series database. Now we have data available in time-series format in Prometheus which will be our data source for Grafana. Other than Prometheus, SQL database is also used as data source to visualize database. Grafana needs to be configured to get data from these data sources. These values are then passed to Grafana panels, which is accessible through browser. For alert and notification rules needs to be set on Graph panel, for that individual rules on the panel also need to be created along with conditions. When these conditions fails it will generate alert and then notification will be sent to specified channel.

3.3 Spinning up Docker Containers

To easily build and run our system at different installation sites, Docker is used to create separate containers for each component of the system. Docker hub registry also provides some prebuilt application images which can pulled and run as a container. For our required tools, prebuilt images are available on docker registry.

Docker containers can only be created using docker images. Here docker image is the file, which contains code of the application to be executed in a container.

This images can be stored in docker registry or private registry so that it can be accessed whenever required.

To understand the working of docker, sample container creation is discussed below:

Install Docker software, which is available for all major platforms

Pull the docker image of the application for example, Grafana

```
docker pull grafana/grafana:1.0
(pulls grafana image having tag 1.0)

docker images
(lists all images available locally)
```

Spin up container of grafana image

```
docker run -name grafana -p 81:3000 grafana/grafana:1.0
( -name grafana sets container name to grafana)
( -p 81:3000 maps local port to container's port)
( -grafana/grafana:1.0 image to be run on container)
```

On successful creation of the container, container status can be seen using

```
docker ps
(lists all containers running)
```

Once the container is up and running, its nothing but your application is running and can be accessed as it is configured. For Grafana, it can be accessed on specified port 81 here using browser.

Same way for prometheus, cAdvisor and Blackbox containers can be created by pulling their images.

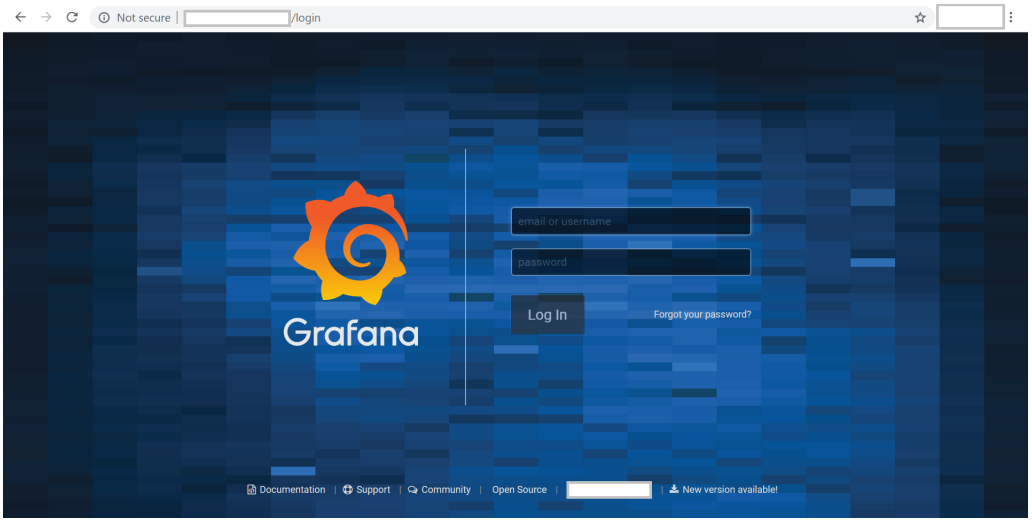

Figure 3.2: Grafana running as a container

3.4 Setup Grafana source and Visualization Panel

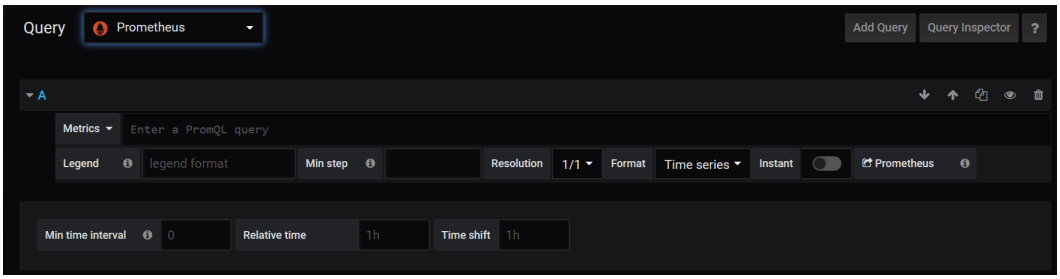

Figure 3.3: Grafana Query Editor for Prometheus

As shown in figure 3.3, First step to setup visualization panel is to setup data source and query for the panel. List of configured data source is available in drop down menu to choose from. Next is to write query to display data on panels. For Prometheus data source, PromQL query is required. For SQL database, SQL query is required.

Sample PromQL query

```
probe_http_status_code{instance="https://www.google.com/"}
(Returns http response status code)
```

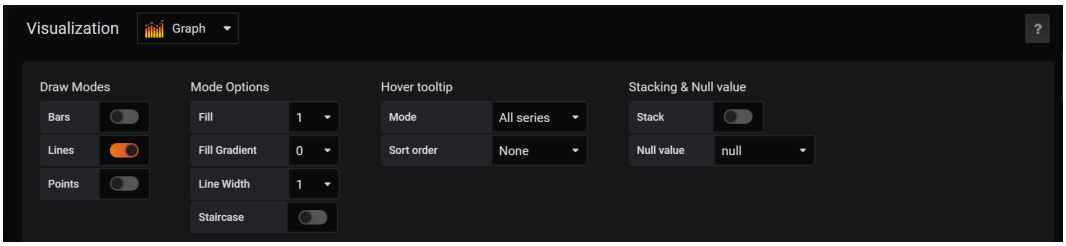

Figure 3.4: Grafana Visualization Editor

Figure 3.4 shows the visualization options for Graph panel. Options and visualization features are different for different panels.

3.5 Notification

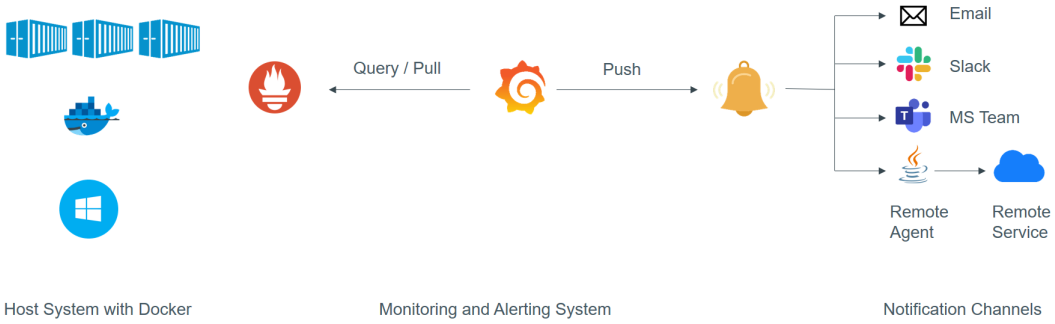

Figure 3.5: System's working flow with notifications

Grafana provides many popular service integration to be used as notification channel. As shown in figure Email, Slack and Microsoft team are popular communication channels. This notification channel requires Web-hook URL and some configuration like intended recipients or group, which can be configured in Grafana notification channel options.

One of the requirement of the system was, it should be able to integrate to existing proprietary system for notification. As Grafana directly can not be integrated with such systems, intermediate agent was required. For which separate agent or say remote notifier is built which fetches alerts from notification systems and converts alerts into specific format. It will also saves converted alerts into log file. Now proprietary system can read this log file and gets the alerts, which is being used to generate notification to the portal and sends mail notification to intended recipients.

3.5.1 Setup Alerts

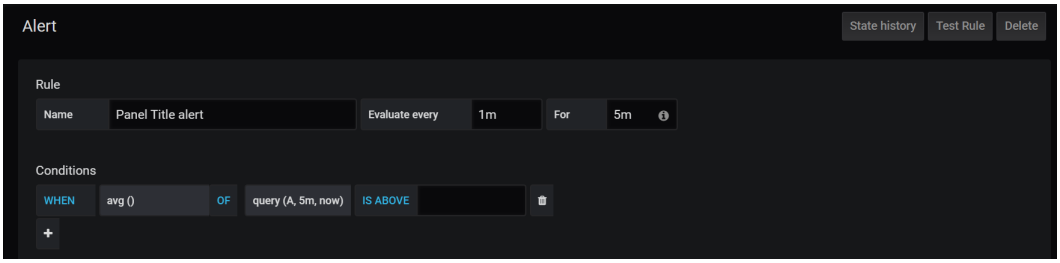

Figure 3.6: Grafana Alert creator

Figure 3.5 shows the alert function for graph panel. In conditions section, multiple conditions can be applied on the data available with panel. When conditions are not satisfied rule will be marked as violated.

3.5.2 Setup Notification channel

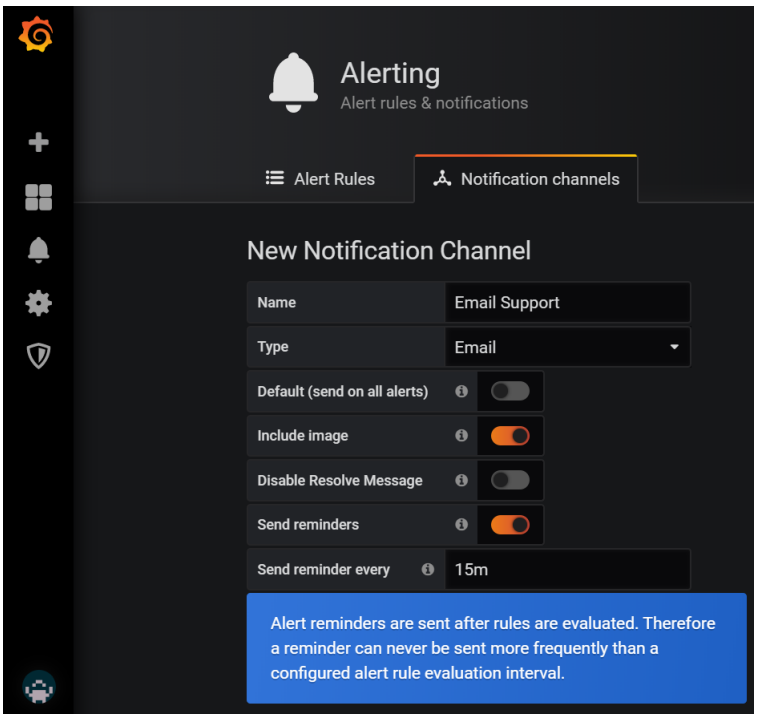

Figure 3.7: Notification Channel

Notification channels can be added by accessing Alerting- Notification Channels tab. As shown in figure, email channel is selected. For which recipients email ids needs to be added and sender email id and SMTP server configuration needs to be setup in Grafana configuration file.

Similar way for different channels different fields needs to be filled. For slack and MS Team Web-hook URL is required. Other common options are reminder timing and image inclusion. Image include option allows to include current snapshot of the panel to be sent to notification along with values. Reminder timing sends notification at specified time interval.

3.5.3 Setup Multiple Notification channel

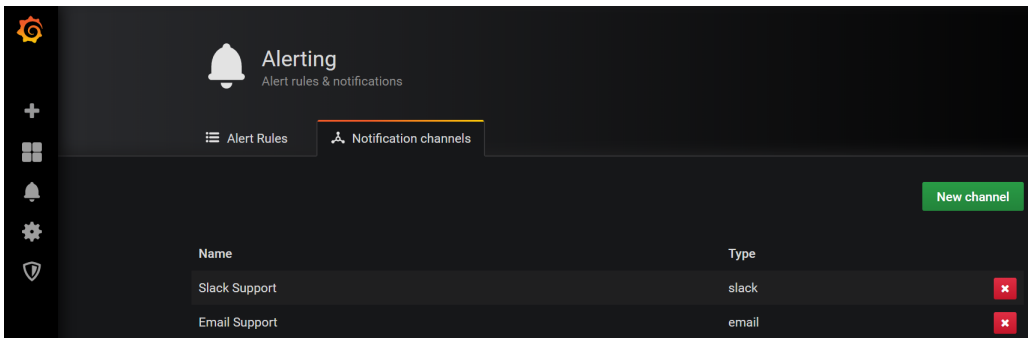

Figure 3.8: List of Notification Channel

Grafana also allows to add multiple notification channels, Which can be of same type or different type. Most of the notification channels are of internet based. To use them, Grafana instance must be able to communicate with them via internet. Grafana also having option for custom notification channel for which custom service needs to available to take notification requests.

3.5.4 Setup notification channel for rule

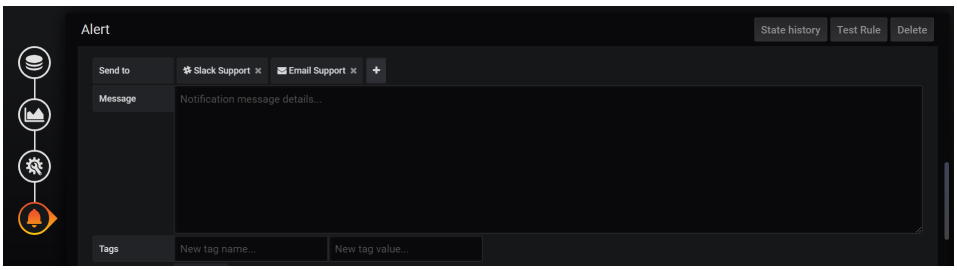

Figure 3.9: Rule Notification Channel

Once notification channels are added as explained in previous section, They available to use for alerts in alerts tab of the graph panel. As shown in figure slack and email channels are added to current alert. So whenever rule is violated, Grafana will try to send notification to these channels.

3.6 System Hardening

System hardening is about securing the system from known vulnerabilities and possible attacks. For securing our notification and monitoring notification, enough measures has been taken and required restriction been applied on the system. Few of them are discussed here.

As we are using docker containers for our system, we have applied restriction on access of these containers. There is restriction applied on root access to container, only from docker console root access can be gained. This will prevent non root user from accidentally modifying the files and also from attackers.

- As we are using docker containers for our system, we have applied restriction on access of these containers. There is restriction applied on root access to container, only from docker console root access can be gained. This will prevent non root user from accidentally modifying the files and also from attackers.

- CPU and Memory constraints has been applied at docker engine level to control over utilization of resources in case of any attacks.

- At application level, Grafana has been configured to have different accounts for admin and non admin users. Non admin user only can view dashboards and can not modify or change admin settings.

3.7 Docker Swarm

It is a container orchestration tool, which allows containers to be deployed across multiple hosts. It provides high availability of applications as number of containers can be scaled. For this project docker swarm is used to deploy stack of our services. Docker swarm also uses the docker-compose.yml file. This file defines services, network and volumes, which allows multiple containers to be spin up in single go.

```
Docker stack deployment command

        docker stack deploy -c docker-compose.yml mystack
        (passing docker-compose.yml file)
        (mystack Providing name to stack)

        docker stack ls
        (lists available stacks)
```

3.8 Configuration of Docker images

As discussed in previous section about setting up Prometheus jobs and targets, creation of dashboards in Grafana, setting up rules notification channels, Setting up targets for Blackbox etc will become tedious task for multiple installation of the system at different sites.

So to have easy and customized installation of the system, some of the configuration which can be pre-set are discussed here. Also for all images some of the customization like port number on which service can accessible, is also discussed.

3.8.1 Configuration by passing $Env to container

Sample Docker Compose file

```
docker-compose.yml


Version: '3.4'
services:
     monitoring:
          image: grafana/grafana:v1.0
          environment:
               - SERVER_PORT=${PORT_NUMBER}
    .....
```

3.8.2 Grafana configuration

Sample Docker Compose file

`docker-compose.yml`

```
Version: '3.4'
services:
     monitoring:
          image: grafana/grafana:v1.0
          environment:
           -SERVER_PORT=${PORT_NUMBER}
           -PATH = ../provisioning_sample/
    .....
```

As discussed previously, creating visualization panels for assets includes steps like selecting data source, writing queries, customizing visualization, writing alert rules and setting up notification channels.

Same steps needs to be carried out for all assets to be monitor. Also these dashboard panels are created manually after the installation of Grafana.

As system will be installed at multiple installation sites, creating dashboards manually is not feasible. Grafana provides option to export these dashboards in JSON file format. Which can be imported after installation. This will eliminate importing of dashboards after installation.

Still we wanted our system installation to be as easy as possible, we have provisioned these dashboards inside image itself. This provisioning setting is shown in above sample compose file. Where we are using these provisioned dashboards and passing to Grafana at the time of deployment of the containers.

3.8.3 Prometheus configuration

Sample Prometheus.yml file

```
prometheus.yml


global:
     scrape_interval:     5m
     evaluation_interval: 5m
     scrape_timeout: 2m


scrape-configs:
       - job_name: monitor
         static_configs:
         - targets: [‘localhost:8080’]


    .....
```

As prometheus needs prometheus.yml file with job details and targets to be scraped. Manual updation of file was required before. To eliminate this, we have prepared sample yml file with all jobs and their targets. This sample file is included in image itself which can be replaced with original yml after installation to have new jobs and targets.

3.8.4 cAdvisor configuration

Sample docker-compose.yml file

docker-compose.yml

```
Version: '3.4'
services:
     cAdvisor:
          image: cAdvisor:v1.0
          deploy:
               mode: global
          command:
            - -v=4
            - -docker_only
            - -port=${CONTAINER_MONITOR_PORT}
    .....
```

In cAdvisor configuration, we are passing custom port number, on which cAdvisor will put metrics about performance of docker containers. As we are using swarm mode, we want only one service of cAdvisor to be running on node, for which deploy mode is set to global. Other mode is "replicated", which creates multiple replica of the service.

3.9 Build and Deploy Flow

3.9.1 Normal Deployment flow

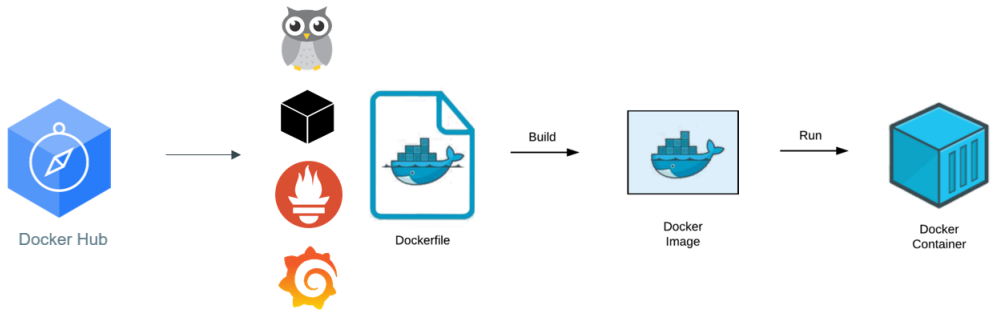

Figure 3.10: Container Deployment

Pre built application images are available on docker hub. For our system required images are cAdvisor, Blackbox Exporter, Prometheus and Grafana. Docker images are built using dockerfile. Dockerfile is a file which contains all commands to be called to assemble an image.

Step 1. Pull images from Docker Hub

Step 2. Create and Run containers from downloaded images

3.9.2 Build Script to auto build customized images

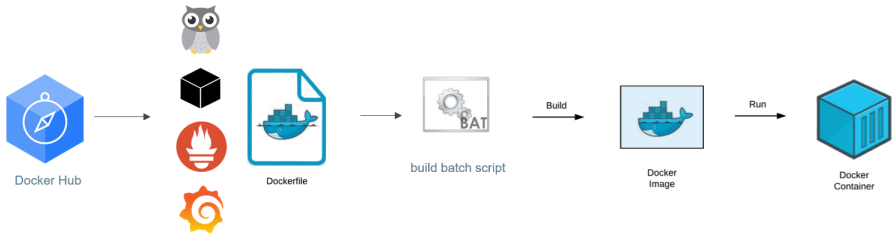

Figure 3.11: Build Script to build all images

Pre built images are configured with default options. But we were required to change these default configuration, so we have created dockerfiles for all four images. These dockerfile will pull images from dockerhub and then adds custom configuration as per our requirements and then assembles customized image locally.

To automate this image build, we have created a built script which will use dockerfiles and creates customized images. Other logic like tagging image with build version number, pushing it to private registry is also been included in the script. Sample build script is shown in next section.

3.9.3 Push Images to private registry

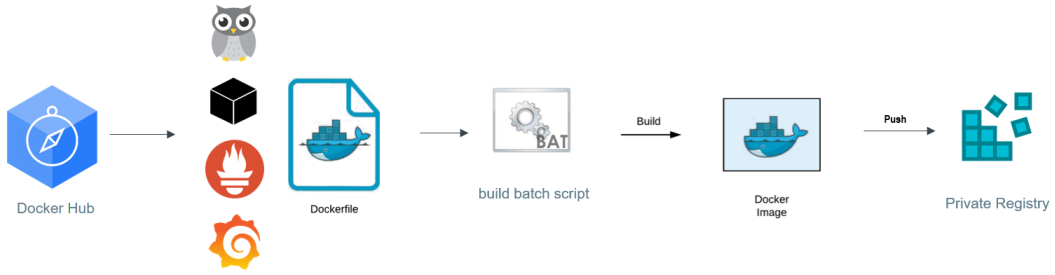

Figure 3.12: Push Images to private registry

To avoid creation of customized images every time, build script will push images to our private registry. Private registry is more secure and reliable for such customized images because it can only be accessed by organization and we do not have to depend on external docker hub. Private registry also stores image with different version numbers or tag id.

To push images to private registry, its authentication is also handled by this build script along with image path and specific format of image name.

3.9.4 Deployment of the images

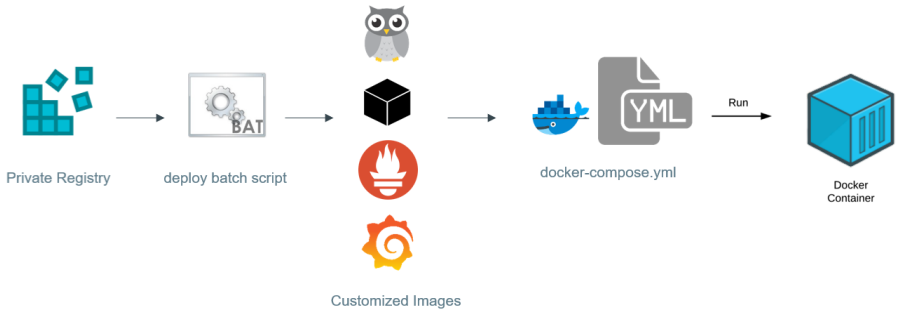

Figure 3.13: Pull and Deploy Images

Once we have images in our private images, it works same as docker hub, we can pull and run images from there. As discussed earlier we have to carry out steps like

- Environment variables to be passed to containers like port number for the services,tag id of images, registry URL etc
- Pull images of specific tag from private registry
- Create Stack of services and run containers.

3.9.5 Sample build script

Sample build.bat file

```
build.bat


set TEST=true
set PUSH=false
set VERSION=1.0.0
set URL= ...
set BUILDNUMBER=
set LOGIN_SUCCESS=false
set BUILD_SUCCESS=false
set PUSH_SUCESS=false

Generate build number according to timezone

Build images with tags

Login to Registry

Push images to registry

.....
```

3.9.6 Sample deploy script

Sample deploy.bat file

```
deploy.bat


set PORT=1000
set VERSION=1.0.0
...

Pull images from registry

Pass docker-compose file

Deploy docker stack of services



.....
```

3.10 Notification integration to proprietary system

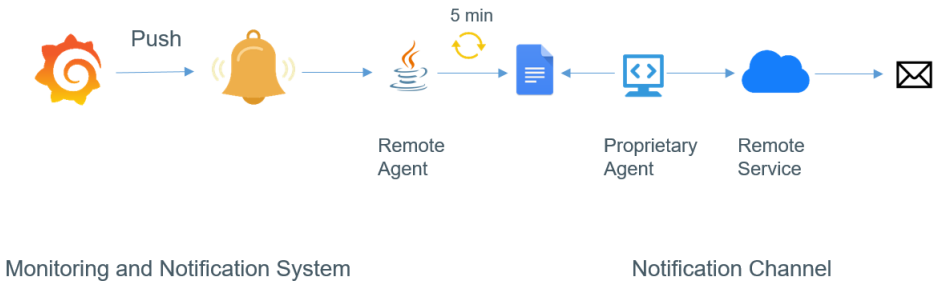

Figure 3.14: Container Deployment

Proprietary system is built and widely used by organization as internal communication channel. It does not depend on any other external system. It was one of the requirement that our system can communicate with it. Grafana can not directly communicate with such system as working of such system be always different.

To provide this integration, we have built one application which we are referring as remote agent. Remote agent is placed between PerformanceFlow system and proprietary system. This agent work is to get alerts generated by Grafana and to convert them into form that can be picked up by proprietary system.

Remote agent is configurable to time interval at which it fetches the alerts and writes into log file.

Below are some of the features developed in remote agent.

- Task scheduler to fetch and generate log files at time interval
- Log file backup based on file size, log file names with timestamps

- Reading environment variable from host system
- Customized log file path to be read by proprietary agent
- Arguments passing for target address,port and authorization key

Chapter 4

Screenshots of PerformanceFlow

4.1 Sample dashboard of PerformanceFlow System resources

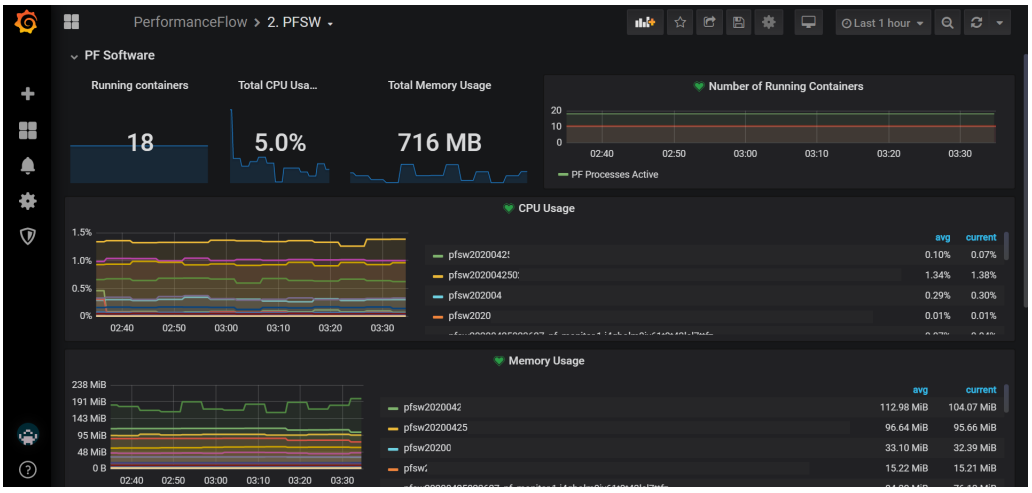

Figure 4.1: Dashboard of PerformanceFlow System

4.2 Sample dashboard of PerformanceFlow System APIs

Figure 4.2: Dashboard of API monitoring

4.3 Container Monitoring

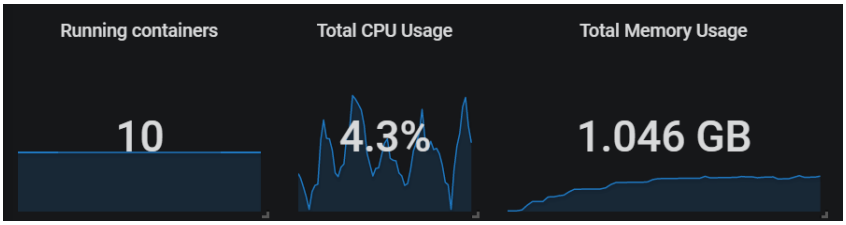

Figure 4.3: Containers Monitoring Dashboard

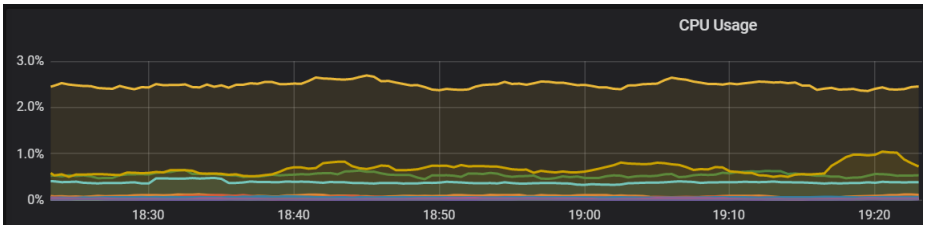

Figure 4.4: Containers CPU Usage Graph Panel

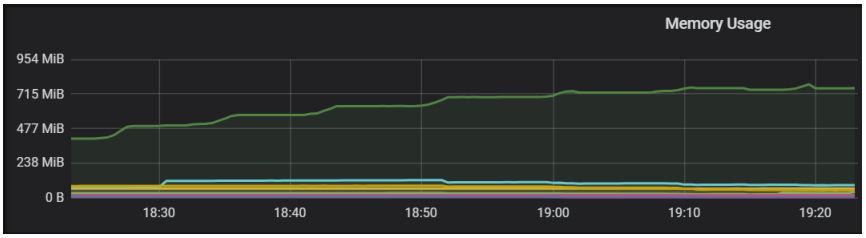

Figure 4.5: Containers Memory Usage Graph Panel

4.4 Database Monitoring

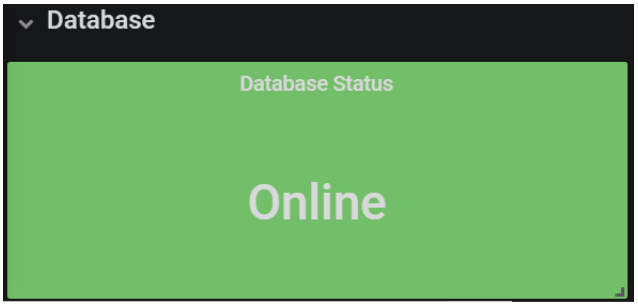

Figure 4.6: Database Status

4.5 Windows Process Monitoring

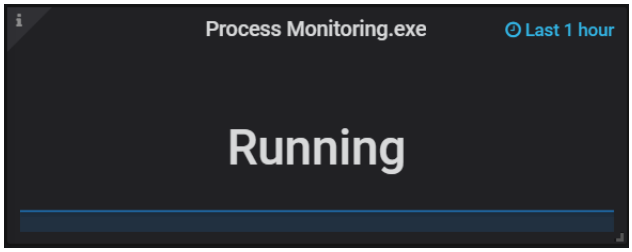

Figure 4.7: Windows Process Status

4.6 API Monitoring

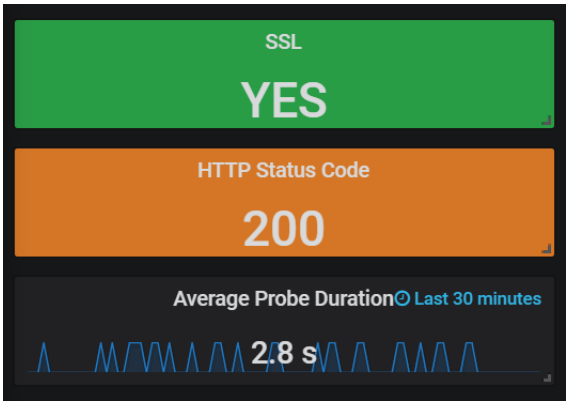

Figure 4.8: API endpoint Monitoring

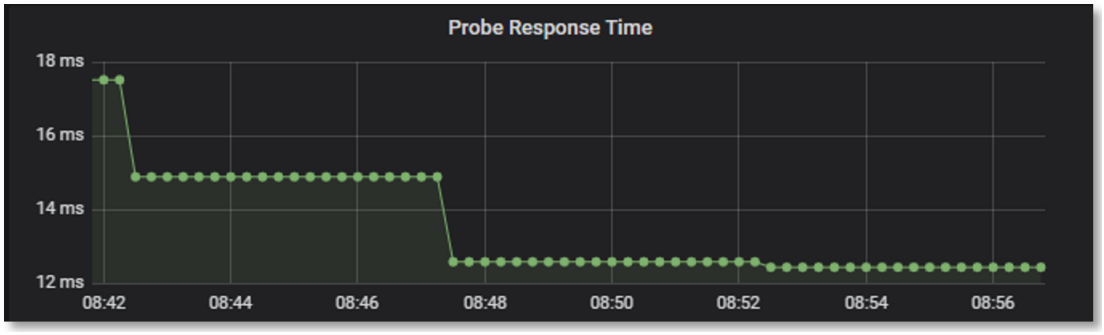

Figure 4.9: API Response time

4.7 Tags Monitoring

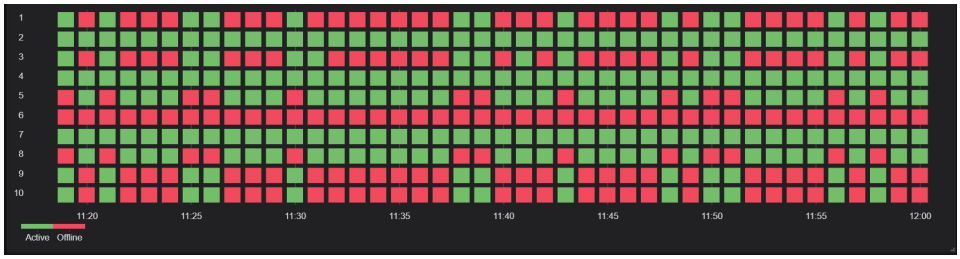

Figure 4.10: Tags Activity

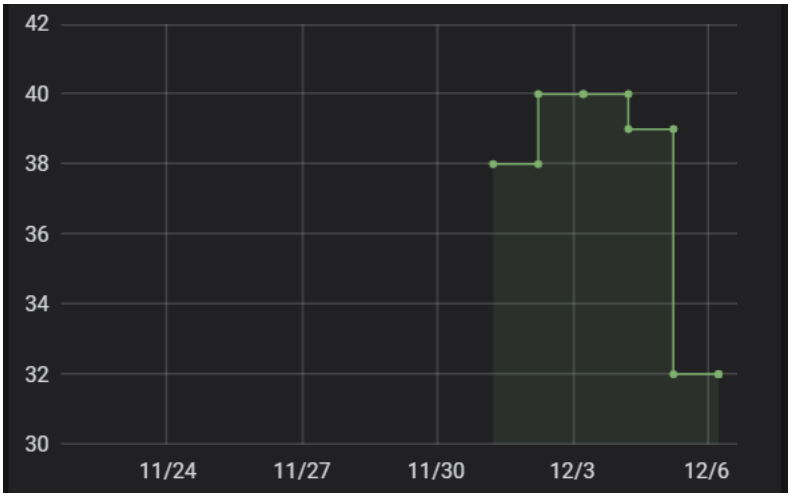

Figure 4.11: Tags per day

4.8 Notifications

4.8.1 Notification on Slack channel

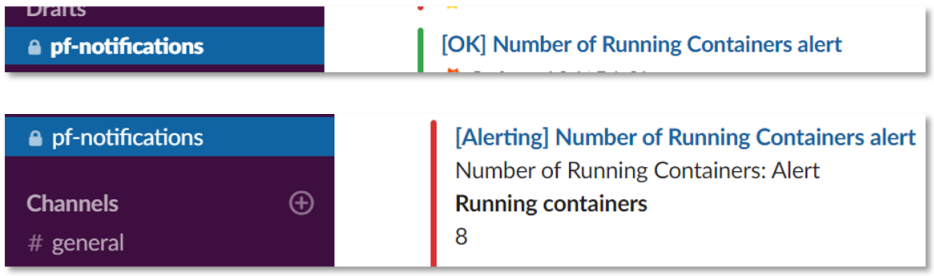

Figure 4.12: Notification on Slack channel

4.8.2 Notification on MS Team

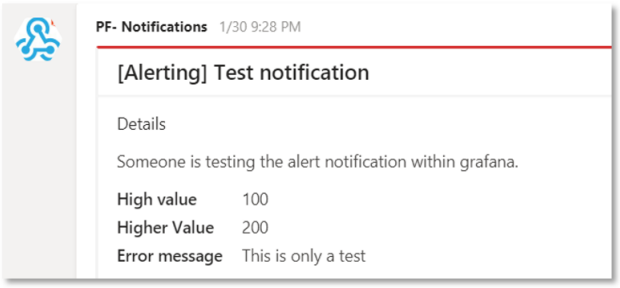

Figure 4.13: Notification on MS Team

4.8.3 Notification by Email

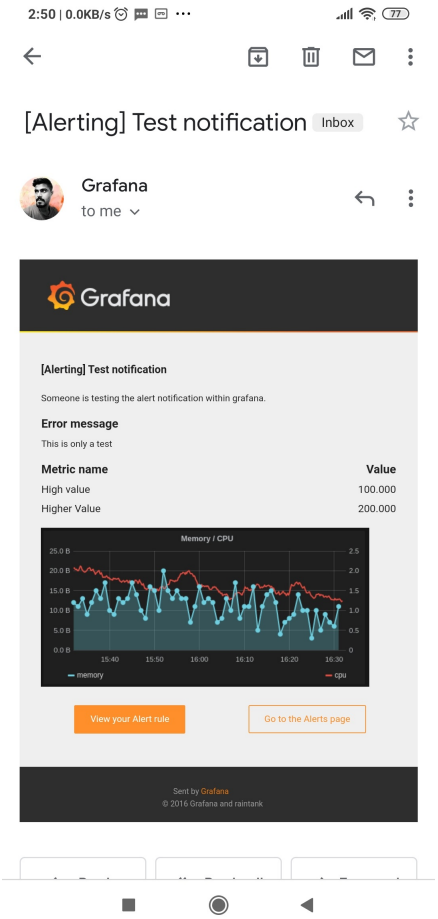

Figure 4.14: Notification by Email

Chapter 5

Future Work and Conclusion

5.1 Future Work

In current PerformanceFlow system, all requirements has been implemented. Future work can be feature enhancement in current system. Some of them are like single sign on, daily report generation and more assets monitoring and analysis panels.

5.2 Conclusion

The current system as per requirements, is able to monitor all required resources and able to display status in dashboard with time-range feature to see historical data. It is also capable of alert and notification to available notification channels. Integration of current system with proprietary system using remote agent has been completed. Also, packaging and installation scripts of the system has been developed, so it is ready to be used.

References

[1] Docker, Overview

https://docs.docker.com/engine/docker-overview/

[2] Docker, Resources : Container

https://www.docker.com/resources/what-container

[3] Docker Hub for Images : Container

https://hub.docker.com/search?image_filter=official&type=image

[4] Prometheus Overview

https://prometheus.io/docs/introduction/overview

[5] Prometheus Comparison

https://prometheus.io/docs/introduction/comparison/

[6] Grafana Docs

https://grafana.com/docs/grafana/latest/

[7] Kibana

https://www.elastic.co/guide/index.html

[8] Google cAdvisor

https://github.com/google/cadvisor

[9] BlackBox exporter

https://github.com/prometheus/blackbox_exporter

[10] WMI Exporter

https://github.com/martinlindhe/wmi_exporter