Detailed Documentation: Monitoring Asterisk Performance with Grafana, Prometheus, and Process Exporter

Overview

This document provides a comprehensive overview of how Grafana, Prometheus, and Process Exporter can be utilized to monitor the performance of an Asterisk SIP server. The solution is designed to analyze system performance, track key metrics, and monitor the server’s resilience against potential attacks.

Objectives

1. Provide real-time monitoring of the Asterisk server's performance.
2. Evaluate SIP traffic statistics to detect anomalies and performance bottlenecks.
3. Enhance the ability to identify and mitigate SIP-based attacks effectively.
4. Monitor critical KPIs such as Call Setup Time, Call Completion Rate, Throughput, Resource Utilization, Latency, and Jitter.

5. Enable proactive measures to maintain service continuity and optimize server performance.

SIP Server Performance Monitoring

Key Metrics for Monitoring

To effectively monitor SIP server performance, the following metrics are analyzed:

- **Call Setup Time**: The duration between initiating a SIP request (e.g., `INVITE`) and receiving the final acknowledgment (`200 OK`). This metric helps identify potential delays in call establishment.

- **Call Completion Rate**: The ratio of successfully completed calls to the total calls initiated. It provides insights into call success and failure trends.

- **Throughput**: The volume of SIP requests processed over time, expressed as calls per second or requests per second. This metric

indicates the server's ability to handle concurrent SIP traffic.

- **Resource Utilization**: Key system resources like CPU load, memory, disk, and network bandwidth utilization during server operation. Monitoring these parameters helps in optimizing resource allocation.

- **Latency**: The time taken for data to travel between origin and destination. High latency can degrade call quality and user experience.

- **Jitter**: Variability in latency over time, often caused by network congestion or hardware issues. Jitter is critical in maintaining consistent call quality.

SIP Performance Analysis Dashboard for Asterisk

A SIP Performance Analysis Dashboard offers a centralized interface to monitor and analyze the

performance of an Asterisk server in real time. Using Grafana as the visualization platform, this dashboard integrates with Prometheus for data collection and Process Exporter for monitoring processes. The dashboard includes panels for:

1. System Performance Metrics

Collected via **Node Exporter**:

- CPU utilization
- Memory usage
- Disk I/O performance
- Network throughput

2. Asterisk Process Metrics

Collected via **Process Exporter**:

- Resource consumption of the Asterisk process (CPU and memory).
- Number of active threads and processes.

3. SIP Traffic Metrics

Collected via a **Custom Process Exporter** and analyzed from SIP log files:

- Total endpoints registered on the server.
- Total active and completed calls.

- Call setup time and completion rate.
- Jitter and latency statistics for active calls.

Attack Monitoring and Detection

SIP-Based Attack Challenges

Detection of SIP-based attacks is challenging due to:

1. High effectiveness in causing service disruptions.
2. Subtle indicators of malicious activity within regular traffic.

Countermeasures

To defend against attacks, the monitoring setup incorporates:

- **Rate Limiting**: Prevents excessive SIP requests from overwhelming the server.
- **Firewalls**: Blocks unauthorized or malicious traffic.
- **Intrusion Detection Systems (IDS)**: Identifies and flags suspicious activity.

- **Traffic Analysis**: Identifies patterns indicative of attacks, such as spikes in failed call attempts or abnormal jitter values.

Key Parameters for Attack Detection

1. **Call Setup Time**: Spikes may indicate a flood of incomplete calls.
2. **Call Completion Rate**: A sudden drop can signify denial-of-service attacks.
3. **Throughput**: Abnormally high request rates might indicate brute-force attacks.
4. **Resource Utilization**: Elevated CPU or memory usage could signal attack attempts.
5. **Latency and Jitter**: Increased variability often correlates with congestion caused by attacks.

Workflow for Monitoring

1. **Data Collection**:
    - **Node Exporter** collects system-level metrics.
    - **Process Exporter** tracks Asterisk's resource utilization.

    - **Custom Process Exporter** analyzes SIP log files for call and traffic metrics.
2. **Data Storage and Processing**:
    - Prometheus scrapes data from exporters and stores it for query-based analysis.
3. **Visualization**:
    - Grafana queries Prometheus to create real-time visualizations and alerts.
4. **Alerting**:
    - Threshold-based alerts notify administrators of potential issues, such as resource overutilization or abnormal traffic patterns.

Conclusion

By leveraging Grafana, Prometheus, and Process Exporter, this monitoring solution ensures robust and proactive performance analysis for Asterisk SIP servers. The integration of system and SIP-specific metrics enables administrators to maintain optimal

server performance, detect attacks promptly, and implement effective mitigations.

Prerequisites

1. A Debian-based system with root access.
2. A running Asterisk instance.
3. Basic knowledge of Linux commands.
4. Internet access for downloading necessary packages.

Step 1: Update the System

Ensure the system is up-to-date.

sudo apt update && sudo apt upgrade -y

Step 2: Install Prometheus

2.1 Download Prometheus

Visit the https://prometheus.io/download/ o get the latest version or use the following commands:

cd /opt
sudo wget
https://github.com/prometheus/prometheus/releases/download/v3.1..0/prometheus-3.1.0.linux-amd64.tar.gz

2.2 Extract and Set Up Prometheus

sudo tar -xvf prometheus-3.1.0.linux-amd64.tar.gz
sudo mv prometheus-3.1.0.linux-amd64 prometheus

cd prometheus

2.3 Create a Service File

Create a systemd service file for Prometheus:

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

Add the following:

```
[Unit]
Description=Prometheus Service
After=network.target

[Service]
User=root
ExecStart=/opt/prometheus/prometheus \
   --config.file=/opt/prometheus/prometheus.yml \
   --storage.tsdb.path=/opt/prometheus/data

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

Reload systemd and start Prometheus:

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

Step 3: Install Node Exporter (optional : if you want to cheak host server performance then only)

3.1 Download and Install Node Exporter

cd /opt
sudo wget
https://github.com/prometheus/node_exporter/releases/download/v1.8.0/node_exporter-1.8.2.linux-amd64.tar.gz
sudo tar -xvf node_exporter-1.8.2.linux-amd64.tar.gz

sudo mv node_exporter-1.8.2.linux-amd64 node_exporter

3.2 Create a Service File

sudo nano /etc/systemd/system/node_exporter.service

Add the following:

```
[Unit]
Description=Node Exporter Service
After=network.target

[Service]
User=root
ExecStart=/opt/node_exporter/node_exporter

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

Reload systemd and start Node Exporter:

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

Step 4: Install Process Exporter

For kali linux : sudo apt-get -y install prometheus-process-exporter

4.1 Download and Install Process Exporter

```
cd /opt
sudo wget
https://github.com/ncabatoff/process-exporter/releases/download/v0.7.5/process-exporter-0.8.3.linux-amd64.tar.gz
sudo tar -xvf process-exporter-0.8.3.linux-amd64.tar.gz
sudo mv process-exporter-0.8.3.linux-amd64 process_exporter
```

4.2 Configure Process Exporter

Create a configuration file to monitor Asterisk:

```
sudo nano /opt/process_exporter/process.yml
```

Add the following:

```
process_names:
  - name: "asterisk"
    cmdline:
      - "asterisk"
```

4.3 Create a Service File

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

Add the following:

```
[Unit]
Description=Process Exporter Service
After=network.target

[Service]
User=root
ExecStart=/opt/process_exporter/process-exporter \
    --config.path /opt/process_exporter/process.yml

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

Reload systemd and start Process Exporter:

```
sudo systemctl daemon-reload
sudo systemctl enable process_exporter
sudo systemctl start process_exporter
```

Step 5: Configure Prometheus to Scrape Metrics

Edit the Prometheus configuration file:

sudo nano /opt/prometheus/prometheus.yml

Add the following scrape configurations:

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

  - job_name: 'process_exporter'
    static_configs:
      - targets: ['localhost:9256']

  - job_name: 'asterisk_metrics_exporter'
    static_configs:
      - targets: ['localhost:8000']
```

Restart Prometheus:

sudo systemctl restart prometheus

Step 7: Install and Configure Grafana

7.1 Install Grafana

sudo apt install -y software-properties-common
sudo add-apt-repository "deb [arch=amd64] https://packages.grafana.com/oss/deb stable main"
sudo apt update
sudo apt install grafana -y

7.2 Start Grafana

sudo systemctl enable grafana-server
sudo systemctl start grafana-server

7.3 Access Grafana

Open a web browser and navigate to `http://localhost:3000`. Log in with:

- **Username**: admin
- **Password**: admin

7.4 Add Prometheus as a Data Source

1. Go to **Configuration > Data Sources > Add Data Source**.
2. Select **Prometheus**.
3. Set the URL to `http://localhost:9090` and save.

7.5 Import Dashboards

1. Use Grafana's built-in dashboards or import JSON templates to visualize Asterisk and system metrics.

Step 8: Validate and Monitor

Verify the exporters are running:
sudo systemctl status node_exporter process_exporter asterisk_metrics_exporter

1.
2. Ensure Prometheus scrapes metrics without errors.
3. Customize Grafana dashboards as needed.

Conclusion

You have successfully set up a monitoring solution for Asterisk performance using Prometheus and Grafana. Adjust and scale the setup to fit your environment.

Others conf file content:-

1:nano /etc/prometheus/prometheus.yml

My global config

global:

```
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default i>
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is ev>
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
        - static_configs:
        - targets:
        # Uncomment and set the Alertmanager endpoint if you have one
        # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evalu>
rule_files:
  # Uncomment and specify your rule files if you have any
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself and Asterisk metrics.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries
scraped>
```

```
  - job_name: "prometheus"
        static_configs:
        - targets: ["localhost:9090"]


  - job_name: "asterisk"
        static_configs:
        - targets: ["localhost:8087"]
        fallback_scrape_protocol: "PrometheusText0.0.4"
  - job_name: 'node_exporter'
        static_configs:
        - targets: ["localhost:9100"]
  - job_name: 'asterisk_process'
        static_configs:
        - targets: ['localhost:9256']
```

2:nano /etc/asterisk/prometheus.conf

```
[general]
enabled=yes
uri = metrics
Core_metrics_enabled =yes
```

3:- nano /etc/asterisk/http.conf

[general]

servername=Asterisk

enabled=yes

bindaddr=0.0.0.0

bindport=8087

4: nano /etc/asterisk/manager.conf

[general]

enabled = yes

webenabled = yes

bindaddr = 127.0.0.1 ; Use 0.0.0.0 to allow external access (ensure firewall i>

[Asterisk]

secret = 123456

read = all

write = all

5: nano /etc/asterisk/pjsip.conf

global]

type=global

user_agent=Asterisk PJSIP

```
endpoint_stats = yes

[general]

allow_reload = yes


[transport-udp]

type=transport

protocol=udp

bind=0.0.0.0:5060


[7001]

type=endpoint

context=internal

disallow=all

allow=ulaw

auth=auth7001

aors=7001


[auth7001]

type=auth

auth_type=userpass

username=7001

password=7001
```

```
[7001]
type=aor
max_contacts=5

[7002]
type=endpoint
context=internal
disallow=all
allow=ulaw
auth=auth7002
aors=7002

[auth7002]
type=auth
auth_type=userpass
username=7002
password=7002

[7002]
type=aor
max_contacts=5
```

```
[7003]
type=endpoint
context=internal
disallow=all
allow=ulaw
auth=auth7003
aors=7003

[auth7003]
type=auth
auth_type=userpass
username=7003
password=7003

[7003]
type=aor
max_contacts=5

[7004]
type=endpoint
context=internal
disallow=all
```

allow=ulaw

auth=auth7004

aors=7004

[auth7004]

type=auth

auth_type=userpass

username=7004

password=7004

[7004]

type=aor

max_contacts=5

[sipp]

type=endpoint

context=default

disallow=all

allow=ulaw

auth=sipp_auth

aors=sipp_aor

[sipp_auth]

type=auth

auth_type=userpass

username=sipp

password=secret123

[sipp_aor]

type=aor

max_contact=5