📘Unified Observability & Monitoring Stack Documentation

A Complete Guide to Grafana, Prometheus, Loki, cAdvisor, Promtail & Alertmanager

🔍 Overview

This document provides a comprehensive explanation and hands-on demonstration of a modern observability and monitoring stack built using open-source tools. It covers every essential aspect of system monitoring — from metrics and logs to traces and alerts — offering a unified approach to understanding application and infrastructure health.

📑 Contents

6. Loki & Promtail

- Centralized log aggregation and querying
- Log collection, labeling, and visualization
- Using LogQL for log queries
- Architecture: *Promtail → Loki → Grafana*

7. Alerting with Grafana & Alertmanager

- Creating alert rules and thresholds
- Contact points and notification policies
- Integrating Alertmanager for alert routing and deduplication
- Real-world use case: CPU usage alert via Slack

8. Visualization Types

- Time series, stat panels, gauges, bar charts, tables, pie charts, state timelines, and more
- Choosing the right visualization for different data types

9. Demo Setup using Docker

Step-by-step setup of all monitoring tools:

- Prometheus
- Grafana
- Loki
- cAdvisor
- Promtail
- Node Exporter
- Alertmanager

Each section includes commands, configurations, and workflows to help you deploy a complete monitoring stack in a containerized environment.

📈 Outcome

By the end of this guide, you'll be able to:
✅ Monitor system and container metrics in real time
✅ Aggregate and analyze application logs
✅ Configure alerts for proactive incident response
✅ Build interactive Grafana dashboards
✅ Deploy a fully functional observability stack using Docker

🔗 Tools Covered

Grafana | Prometheus | Loki | cAdvisor | Promtail | Alertmanager

🔍 What is Observability?

Observability usually has three core components:

1. **Metrics** → Numbers measured over time, like CPU usage, request count, or latency
2. **Logs** → Detailed text output such as errors, database warnings, or stack traces.
3. **Traces** → The path a single request takes as it travels through your system.

Together, these help you **detect, understand, and resolve issues faster**.

🧩 What Are Traces?

Think of a **trace** as a map or timeline of a single request's journey across your system.

Example using a **Prisma backend**:
• A client sends a request to your API → this is the **trace root**.
• The API makes a Prisma database query → that becomes a **span** within the trace.
• The Prisma query hits your PostgreSQL database → another span.

Each span records details like duration, metadata, and errors. When combined, these spans form a **complete picture of where time is spent** and **where potential bottlenecks or failures occur**.

Why Are Traces Useful?

• Identify performance bottlenecks (e.g., API is fast, but the Prisma query takes 600ms).
• Debug distributed systems (when requests move through multiple microservices).
• Correlate logs and metrics back to a specific request.

Traces are typically sent to backends like **Grafana Tempo**, **AWS X-Ray**, or **Jaeger**.

🖥️ What is Monitoring?

Monitoring is the continuous process of **collecting, analyzing, and visualizing** system and application data to understand their **performance, availability, and overall health**.

It's the foundation of **observability** — where we gather **metrics, logs, and traces** to get a complete picture of what's happening inside our systems.

Together, these allow monitoring tools to not just detect when something breaks, but also help you understand **why** it happened and **where** it started.

In short, monitoring ensures you always know the **current state of your system** and can act quickly when something goes wrong.

📈 What is Grafana?

Grafana is a free and open-source tool used to **visualize and analyze monitoring data** from multiple sources like **Prometheus, Loki, or Tempo**.

It transforms raw numbers and logs into **clear dashboards, graphs, charts, and alerts**, making complex data easy to understand at a glance.

Think of Grafana as the **dashboard of a car** — it doesn't just show you numbers, it gives you **insight**. You can instantly see if your system is healthy, which part is under stress, and when something needs attention — all in one place.

🧩 What is a Data Source?

In Grafana, a **data source** is the place where your data actually comes from. Grafana itself doesn't store any data — it only **visualizes data fetched from other systems** like Prometheus, Loki, or Tempo.

A data source could be a **database, metrics system, log aggregator, or API endpoint** that continuously provides the data Grafana needs to display on dashboards.

Each data source type tells Grafana **how to query the data** and what kind of information to expect (metrics, logs, traces, etc.).

⚙️ How to Add a Data Source

To start visualizing data in Grafana, you first need to connect a data source. Here's how you can add one:

1. Open Grafana and log in as an admin.
2. From the left sidebar, go to **Connections** → **Data sources**.
3. Click **Add data source**.
4. Select the data source type (for example, Prometheus, Loki, or PostgreSQL).
5. Enter the connection details — like the URL or access credentials.
6. Click **Save & Test** to verify the connection.

If everything is configured correctly, Grafana will confirm that the data source is connected successfully.

🧠 Available Data Sources in Grafana

Grafana supports a wide range of data sources depending on your observability stack and needs. Some of the most common ones include:

- **Prometheus** → For metrics data
- **Loki** → For log data
- **Tempo** → For traces
- **Elasticsearch** → For full-text log searches
- **MySQL / PostgreSQL** → For relational database queries
- **CloudWatch / Azure Monitor / Google Cloud Monitoring** → For cloud monitoring data
- **InfluxDB, Graphite, OpenTSDB** → For time-series metrics
  Each of these data sources integrates differently based on the data type and query language it uses.

🧩 What if a Data Source is Not Available?

If you don't see a specific data source listed in Grafana by default, you can **install it as a plugin**.
To do this:

1. Go to **Connections** → **Add data source** → **Find more data sources**.
2. Browse or search for the plugin you need (for example, MongoDB or Jira).
3. Click **Install Plugin**.
4. After installation, you'll find it available in the data source list to configure and connect.

   Grafana's plugin ecosystem makes it possible to connect almost any external system to your dashboards.

🔥 Prometheus Data Source

Prometheus is one of the most popular data sources in Grafana and is primarily used for **metrics monitoring**. It collects numerical data (time series) such as CPU usage, memory consumption, and request rates.
When you connect Prometheus to Grafana, you can query data using **PromQL (Prometheus Query Language)** to create visualizations like:

- CPU utilization over time
- API request latency
- Error rate and throughput
- Custom metrics exported from your application

Prometheus works best for **real-time system monitoring** and pairs perfectly with Grafana's time-series panels.

Why Use Prometheus + Grafana

When you use Prometheus and Grafana together, you get a powerful and simple way to monitor your system.

- **Prometheus** collects metrics such as CPU usage, memory, and disk space.
- **Grafana** visualizes that data with clear and beautiful dashboards.

In short, Prometheus watches your system, while Grafana shows what's happening in real time making monitoring smart and efficient.

How Prometheus Works

1. Prometheus requests system data from tools called **exporters** (e.g., "What's the CPU usage now?").
2. Exporters respond with metrics like CPU: 30%, Memory: 40%.
3. Prometheus stores this time-series data (data with timestamps).
4. Grafana connects to Prometheus and turns this data into interactive dashboards and charts.

Node Exporter

Node Exporter is used with Prometheus to collect **hardware and system-level metrics** from your Linux server.

- It runs on each server (node) and provides metrics like CPU usage, memory, disk space, network traffic, and system uptime.
- Prometheus scrapes this data and stores it for visualization or alerting in Grafana.

Example Flow:
EC2 Instance → Node Exporter → Prometheus → Grafana

cAdvisor (Container Advisor)

cAdvisor monitors Docker container-level metrics such as CPU, memory, network throughput, and filesystem usage for each container.

- It helps you understand how each container performs and uses resources.
- Prometheus scrapes these metrics and exposes them to Grafana dashboards for real-time visualization.

Example Flow:
Containers → cAdvisor → Prometheus → Grafana

🧾 Loki Data Source

Loki is Grafana's log aggregation system — designed to collect, store, and query log data efficiently. Unlike traditional log systems, Loki indexes only labels (not full log content), making it faster and cheaper to operate.
When integrated with Grafana, Loki helps you visualize **logs side by side with metrics**. This allows you to quickly move from a spike in a graph (from Prometheus) to the exact log lines that caused it.
Example use cases:

- View application errors and warnings in real time
- Filter logs by labels such as `job`, `statusCode`, or `requestId`
- Correlate logs with metrics using **LogQL** (Loki Query Language)

Why Use Loki + Grafana

When you use **Loki** and **Grafana** together, you get a complete and efficient log monitoring solution.

- **Loki** collects and stores logs from multiple sources in a lightweight and cost-effective way.
- **Grafana** visualizes and filters these logs with powerful queries and interactive dashboards.
  In short, Loki keeps track of what your system is saying, while Grafana helps you read and understand it — all in one place alongside your metrics.

How Loki Works

1. **Promtail** runs on servers or containers and collects log files (for example, `/var/log` or Docker container logs).
2. Promtail adds useful **labels** (like job, instance, or container name) to make logs easier to filter.
3. Logs are **pushed to Loki**, where they are indexed by labels and stored efficiently.
4. **Grafana** connects to Loki and uses **LogQL** queries to visualize and explore logs in real time.

Promtail

Promtail is the agent responsible for collecting and shipping logs to Loki.
It reads log files, attaches metadata (labels), and sends them securely to Loki for central storage and querying.

Example Flow:
EC2/Containers → Promtail → Loki → Grafana

Workflow Summary

Metrics:

- EC2 → Node Exporter → Prometheus → Grafana
- Containers → cAdvisor → Prometheus → Grafana
- EC2/Applications → CloudWatch Agent → CloudWatch → Grafana

Logs:

- EC2/Containers → Promtail → Loki → Grafana
- EC2/Applications → CloudWatch Agent → CloudWatch → Grafana

📊 Common Visualizations in Grafana

Grafana supports multiple visualization types, each designed to tell a different story about your data. Let's go through the most common ones 👇

🕑 Time Series

Used to visualize data collected over time.
Examples: monitoring weather data throughout the day or tracking a YouTuber's subscriber count.
The Y-axis shows the value being measured, and the X-axis represents time. Data can be displayed as lines, bars, or points.

📊 Bar Chart

Useful for comparing data categories side by side — for example, request counts across different endpoints or environments.

📉 Stat Visualization

Displays a single real-time value like current CPU usage or memory percentage. Great for showing important key metrics at a glance.

⛽ Gauge and Bar Gauge

These look like a fuel gauge or volume meter. They visualize numeric values within a defined range.
For example, to monitor CPU utilization:
🟢 Normal → 0–80%
🔴 Critical → Above 90%
Perfect for showing if something crosses its safe limit.

🧾 Table

Displays data in rows and columns for easy comparison. You can also include bar gauges inside table cells and highlight specific rows or columns based on conditions.

🥧 Pie Chart

Used to show proportions of a whole — such as the percentage of traffic from different services or the distribution of status codes (200s vs 500s).

⏱️ State Timeline and Status History

These help visualize how a system's state changes over time.

State Timeline Example:
Consider CPU utilization for EC2 instances:
• Below 20% → Low
• 20–50% → Normal
• 50–80% → High
• Above 80% → Critical
A state timeline lets you see exactly when the CPU was in each range.

Status History:

Used for boolean states - like whether an instance is running or stopped.

If running, it displays "true"; if stopped, it shows "false".

💬 Text Visualization

Used to add text, HTML, or external images directly into a dashboard. Great for adding titles, descriptions, or custom images.

🧩 Panel Plugins

Grafana also offers a wide range of panel plugins that extend visualization capabilities. You can explore them on the **Panel Plugins** page to discover new ways of representing data.

Creating a Dashboard in Grafana

Dashboards in Grafana are used to visualize and monitor data in real time. They display multiple panels, each showing different metrics or logs from your data sources.

Steps to Create a Dashboard

1. **Login to Grafana:**
   Open your Grafana instance in a browser and log in with your credentials.

2. **Create a New Dashboard:**
   From the left menu, click **"+"** → **"Dashboard"** → **"Add new panel."**

3. **Select a Data Source:**
   Choose the data source you want to use — for example, **Prometheus** for metrics or **Loki** for logs.

4. **Write a Query:**
   Use PromQL (for Prometheus) or LogQL (for Loki) to fetch data you want to visualize.

5. **Choose a Visualization Type:**
   Select from charts like **Time Series**, **Stat**, **Gauge**, **Bar Chart**, or **Pie Chart** based on your data.

6. **Customize Panel Settings:**
   Add titles, adjust axes, set thresholds, and format colors for better clarity.

7. **Save the Dashboard:**
   Click **"Apply"** and then **"Save dashboard."** Give it a name and optionally add it to a folder.

8. **Add More Panels:**
   Repeat the process to build a complete dashboard showing system metrics, logs, and performance data together.

Grafana Alerts

Grafana Alerts help you stay informed when something goes wrong in your system. Instead of constantly watching dashboards, you can set up alerts that automatically notify you through email, Slack, or other tools whenever a condition crosses a threshold.

How Alerts Work

1. **Grafana continuously evaluates alert rules** you define (for example, CPU usage > 90%).
2. When the condition is met, it triggers an **alert**.
3. The alert is routed to a **contact point** using **notification policies**.
4. You receive a message about the issue so you can act quickly.

Components of Grafana Alerting

1. Alert Rules

Alert rules define the **conditions** that trigger an alert.

- You can create alert rules directly from any **panel** in your dashboard.
- Use queries (PromQL for Prometheus, LogQL for Loki, etc.) to set conditions.

Example:

    - **Query:** `avg(node_cpu_seconds_total{mode="idle"})`
    - **Condition:** When CPU idle time is less than 10% → trigger alert.

Each rule has:

- **Query:** The data source query to evaluate.
- **Condition:** The logic that decides when to alert.
- **Evaluation interval:** How often Grafana checks the rule.
- **Labels:** Metadata (like severity, service) to group or filter alerts.

2. Contact Points

Contact points are where alerts are sent when triggered.
Examples include:

- **Email**
- **Slack**
- **Microsoft Teams**
- **PagerDuty**
- **Webhook**

You can define multiple contact points and choose which one to use for each type of alert.

3. Notification Policies

Notification policies decide **how alerts are routed** to different contact points.
They define rules such as:

- Which **alerts go to which contact points** (based on labels like `severity=critical`).
- **Grouping** alerts (for example, sending one message for multiple alerts in the same service).
- **Timing** and **escalation** — how and when to repeat alerts if not resolved.

For example:

- Alerts with `severity=critical` → Send to **Slack**
- Alerts with `severity=warning` → Send to **Email**

4. Alertmanager Data Source

Grafana can also integrate with **Alertmanager**, which is a part of the **Prometheus ecosystem**.
Alertmanager manages alerts sent by Prometheus and can:

- **Group**, **deduplicate**, and **silence** alerts.
- **Route** them to notification channels (Slack, Email, etc.).

You can add Alertmanager as a **data source** in Grafana to:

- View active alerts directly from the Alertmanager.
- Visualize or manage alert status in dashboards

Steps to Add Alertmanager Data Source:

1. Go to **Connections** → **Data Sources** → **Add data source.**
2. Search for **Alertmanager**.
3. Enter the Alertmanager URL (e.g., `http://<alertmanager-ip>:9093`).
4. Click **Save & Test**.

This allows Grafana to read alerting data and display alert states directly in your dashboards.

Creating an Alert Rule in Grafana

1. Open a panel in your dashboard and click **“Alert”** → **“Create alert rule.”**
2. Select the **data source** (Prometheus, Loki, etc.).
3. Write the **query** and define the **condition** for triggering.
4. Set the **evaluation interval** (e.g., every 1m).
5. Add **labels** (like `severity=critical` or `team=backend`).

6. Save the rule and link it to a **notification policy**.
7. Test the alert to confirm it triggers correctly.

Example Use Case

- **Condition:** CPU usage > 90% for 5 minutes.
- **Action:** Send a Slack alert to the “DevOps” channel.
- **Policy:** Group all alerts by instance and repeat every 30 minutes until resolved.

Installing Monitoring Stack with Docker

1. Prometheus

Prometheus is a monitoring system and time series database.

Docker Installation: Run the official Prometheus Docker image:

```
docker run -d -p 9090:9090 prom/prometheus
```

- **Official Installation Guide**: Prometheus

2. Grafana

Grafana is an open-source platform for monitoring and observability.

Docker Installation: Run the official Grafana Docker image:

```
docker run -d -p 3000:3000 grafana/grafana
```

- **Official Installation Guide**: Grafana Labs

3. Loki

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus.

Docker Installation: Run the official Loki Docker image:

```
docker run -d -p 3100:3100 grafana/loki
```

- **Official Installation Guide**: Grafana Labs

4. cAdvisor

cAdvisor (Container Advisor) provides container users an understanding of the resource usage and performance characteristics of their running containers.

Docker Installation: Run the official cAdvisor Docker image:

```
docker run -d -p 8080:8080 --volume=/:/rootfs:ro
--volume=/var/run:/var/run:ro --volume=/sys:/sys:ro
--volume=/var/lib/docker/:/var/lib/docker:ro --privileged
--name=cadvisor google/cadvisor:latest
```

- **Official Installation Guide**: GitHub

5. Promtail

Promtail is an agent which ships the contents of local logs to a private Grafana Loki instance or Grafana Cloud.

Docker Installation: Run the official Promtail Docker image:

```
docker run -d -v /var/log:/var/log -v
/etc/promtail/promtail.yaml:/etc/promtail/promtail.yaml
grafana/promtail
```

- **Official Installation Guide**: Grafana Labs

6. Node Exporter

Node Exporter is an exporter for hardware and OS metrics exposed by *nix kernels.

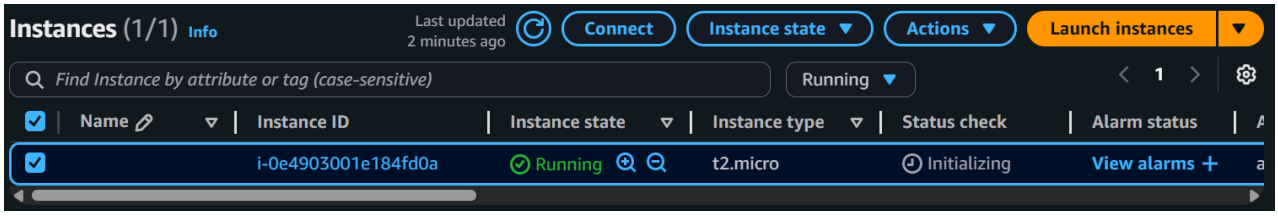
Docker Installation: Run the official Node Exporter Docker image:

```
docker run -d -p 9100:9100 --name=node_exporter prom/node-exporter
```

- **Official Installation Guide**: GitHub

Demo: creating dashboards in grafana

1. Launch an ubuntu Ec2 instance.



2. To install docker compose use below commands

sudo yum update
sudo yum install docker-compose-plugin

3. Execute the below file to install prometheus, grafana, loki, promtail and Cadvisor as docker containers.

docker-compose file

```
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
      - prometheus-storage:/prometheus
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"
      - "--storage.tsdb.path=/prometheus"

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana
    depends_on:
      - prometheus
  node_exporter:
```

```
    image: prom/node-exporter:latest
    container_name: node_exporter
    ports:
      - "9100:9100"

  loki:
    image: grafana/loki:latest
    container_name: loki
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/loki-config.yml
    volumes:
      - ./loki-config.yml:/etc/loki/loki-config.yml

  promtail:
    image: grafana/promtail:latest
    container_name: promtail
    volumes:
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /var/log:/var/log
      - ./promtail-positions:/tmp
      - ./promtail-config.yml:/etc/promtail/config.yml
    command: -config.file=/etc/promtail/config.yml
    depends_on:
      - loki

  cadvisor:
    image: gcr.io/cadvisor/cadvisor:latest
    container_name: cadvisor
    ports:
      - "8080:8080"
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys:ro
      - /var/lib/docker/:/var/lib/docker:ro
    restart: unless-stopped

volumes:
  grafana-storage:
  prometheus-storage:
```

4. Configuring prometheus to identify the targets node exporter and cadvisor

prometheus.yml

```
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: "node_exporter"
    static_configs:
      - targets: ["node_exporter:9100"]

  - job_name: cadvisor
    static_configs:
      - targets: ["cadvisor:8080"]
```

5. Configuring promtail to add the path to our docker and nginx logs

promtail-config.yml

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push
    tenant_id: "local"

scrape_configs:
  # Docker container logs
  - job_name: docker
    static_configs:
      - targets:
          - localhost
        labels:
          job: docker-logs
          __path__: /var/lib/docker/containers/*/*-json.log

  # Access logs (Nginx)
  - job_name: access-logs
```

```
  static_configs:
    - targets:
        - localhost
      labels:
        job: access-logs
        __path__: /var/log/nginx/access.log
```

6. Configuring loki to store the logs

```
server:
  http_listen_port: 3100   # port Loki listens on

common:
  path_prefix: /loki                # base folder for chunks & index
  ring:
    instance_addr: 127.0.0.1
    Kvstore:
      store: inmemory               # in-memory ring for single-node
  replication_factor: 1             # single-node replication

schema_config:
  configs:
    - from: 2020-10-24
      store: tsdb
      object_store: filesystem        # store logs on local filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h                 # rotate index every 24h

storage_config:
  filesystem:
    directory: /loki/chunks         # store actual log chunks

limits_config:
  enforce_metric_name: false
  max_entries_limit_per_query: 10000
```

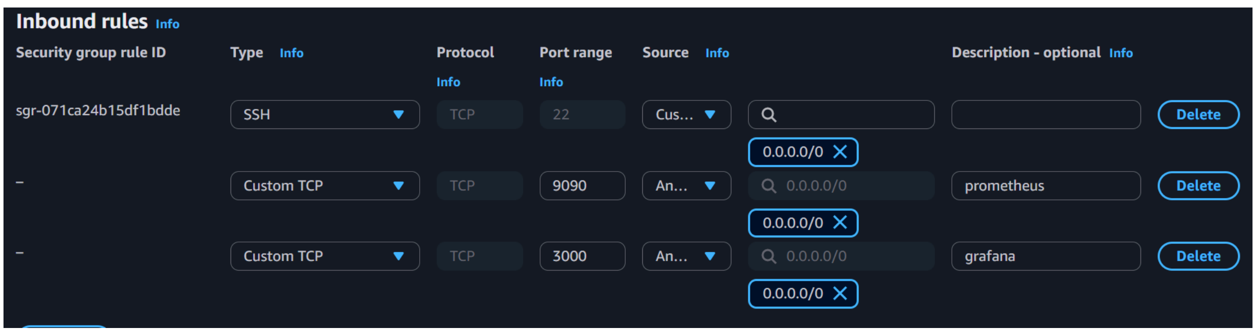
7. Now execute the docker file using command

docker compose up -d

8. Use command docker ps to check the status of containers

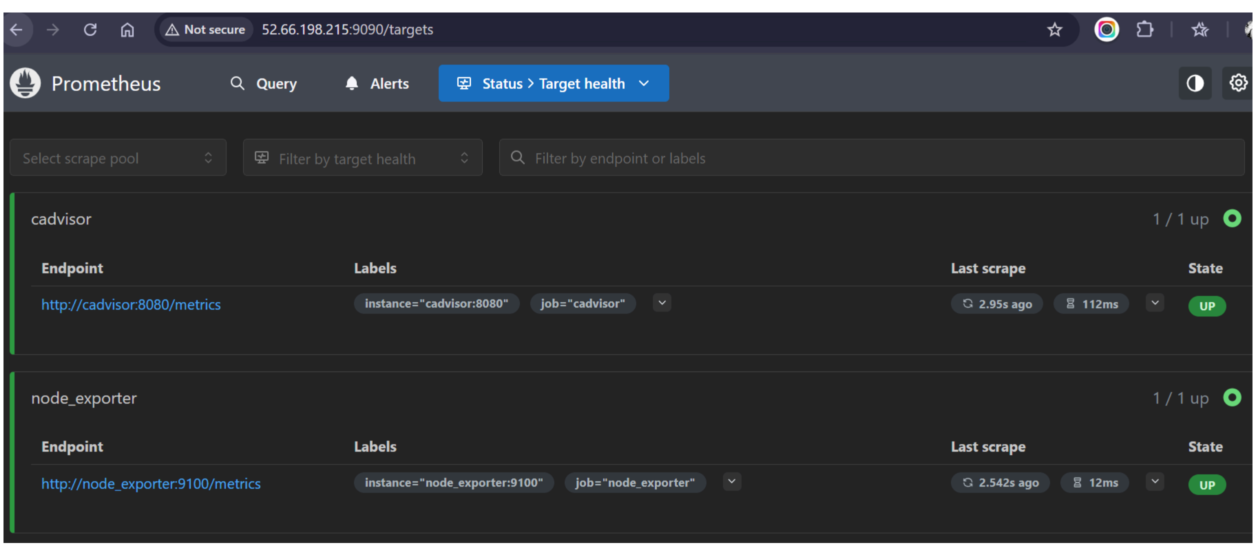
```
[ec2-user@ip-172-31-40-146 grafana]$ docker ps -a
CONTAINER ID   IMAGE                              COMMAND                  CREATED         STATUS                    PORTS
                    NAMES
51fa1eb89470   grafana/promtail:2.9.1             "/usr/bin/promtail -…"   2 minutes ago   Up 4 seconds
                    promtail
69b735445c21   grafana/loki:2.9.1                 "/usr/bin/loki -conf…"   2 minutes ago   Up 2 minutes              0.0.0.0:3100->3100/tcp, :
::3100->3100/tcp    loki
9adde84d10ce   grafana/grafana:latest             "/run.sh"                6 minutes ago   Up 6 minutes              0.0.0.0:3000->3000/tcp, :
::3000->3000/tcp    grafana
a65a150da5b7   gcr.io/cadvisor/cadvisor:latest    "/usr/bin/cadvisor -…"   6 minutes ago   Up 6 minutes (healthy)    0.0.0.0:8080->8080/tcp, :
::8080->8080/tcp    cadvisor
2f07dc74e945   prom/node-exporter:latest          "/bin/node_exporter"     6 minutes ago   Up 6 minutes              0.0.0.0:9100->9100/tcp, :
::9100->9100/tcp    node_exporter
8b0100200ffd   prom/prometheus:latest             "/bin/prometheus --c…"   6 minutes ago   Up 6 minutes              0.0.0.0:9090->9090/tcp, :
::9090->9090/tcp    prometheus
```

9. Add ports 9090 and 3000 on Ec2 instance security groups which are default ports of grafana and prometheus

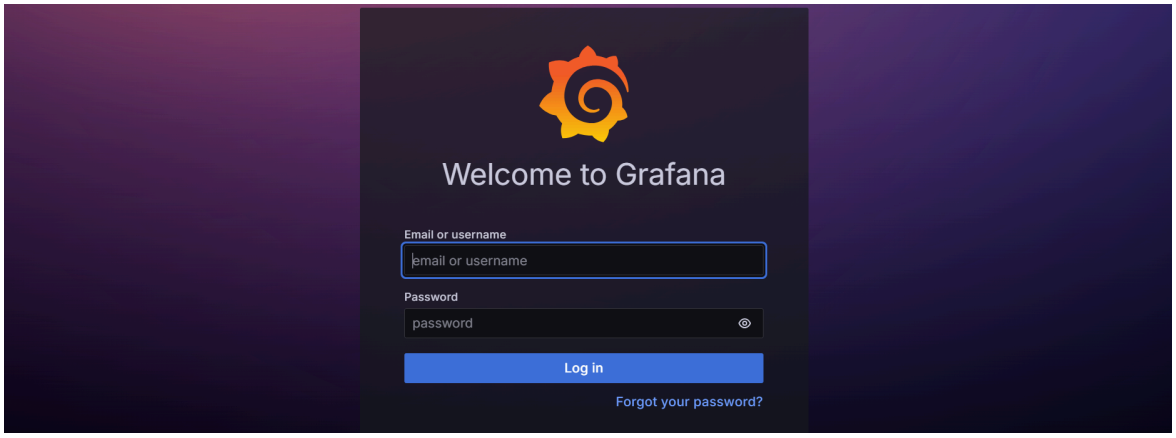


10. Copy the Ec2 instance public ip in your browser with port 9090 now you should see prometheus dashboard

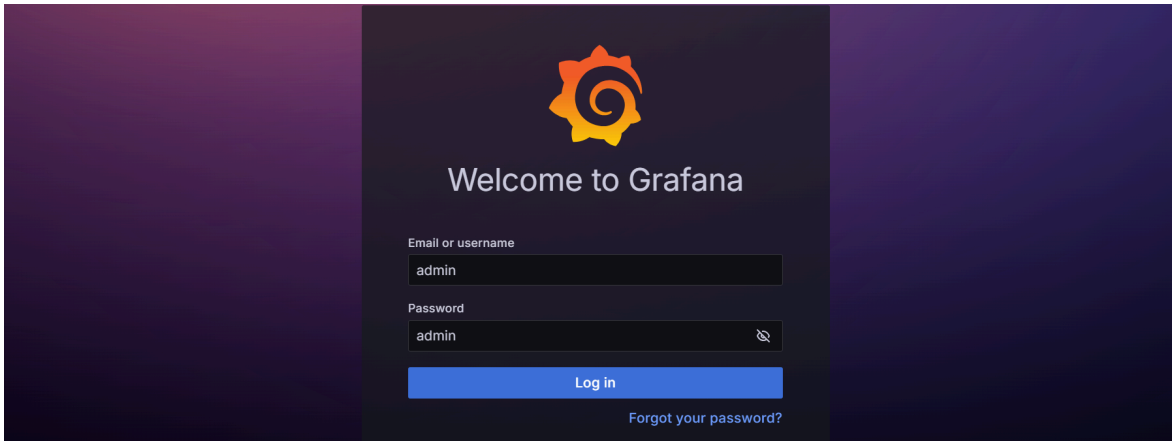
Go to status > target health to verify the exporters are up and running.



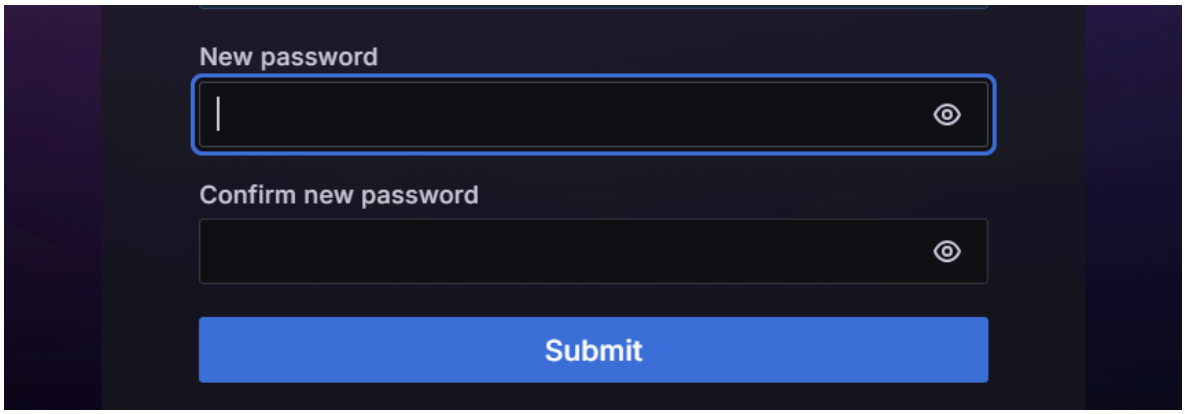
11. Copy the Ec2 instance public ip in your browser with port 3000 now you should see grafana dashboard



11. Login with default username and password which is admin, admin.

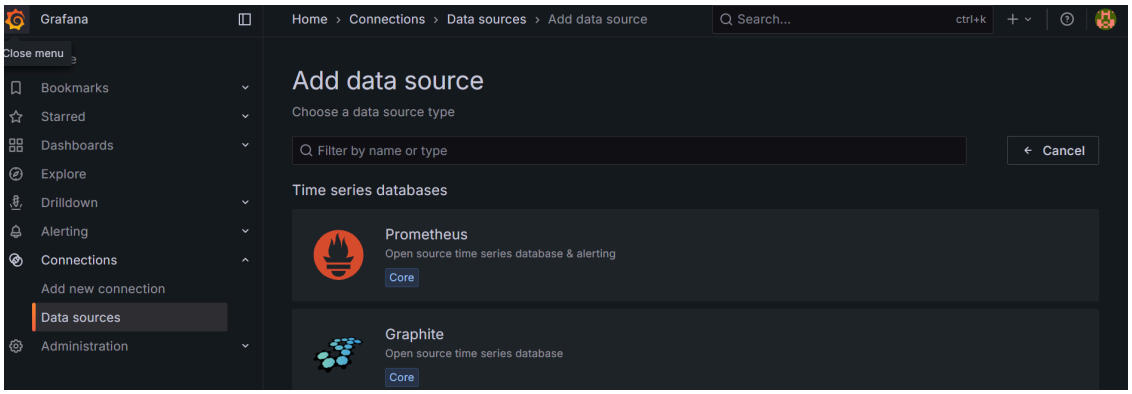


Now you will be promoted to set a new password



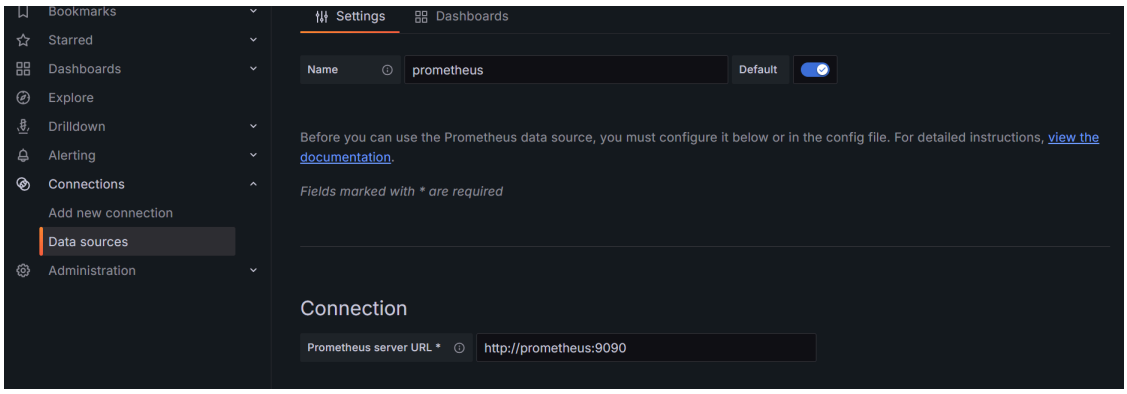
12. Adding data source prometheus

1. Navigate to connections>Data Sources click “Add data source”

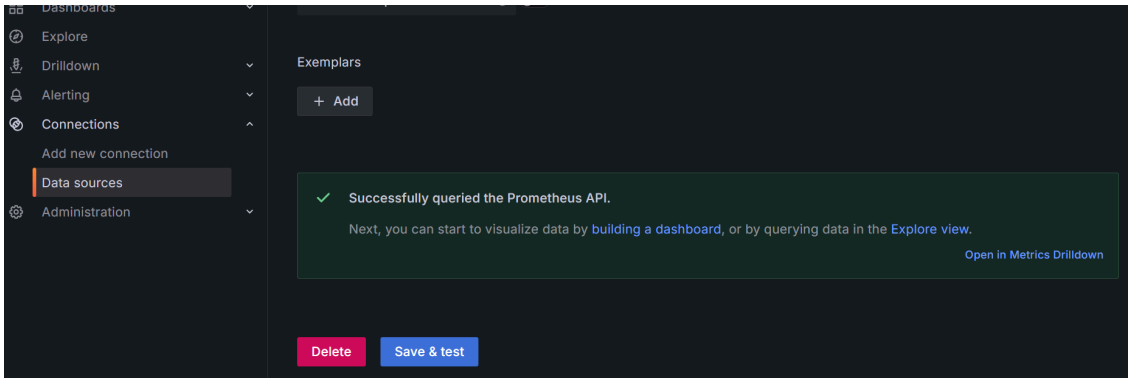


2. Select Prometheus from the list
3. Configure Prometheus Settings

In the URL field, enter: http://prometheus:9090

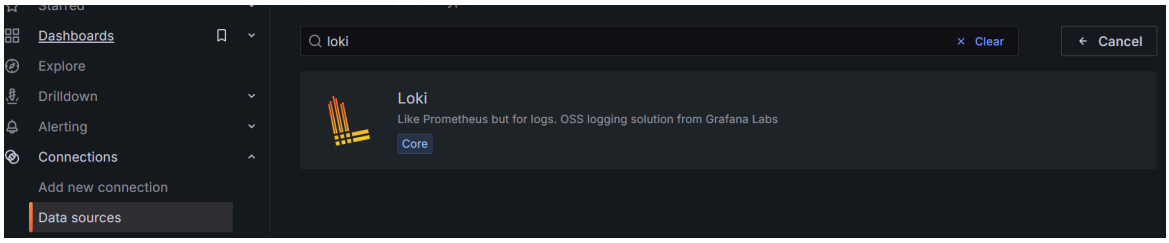


4. Leave the other settings as default click “Save & Test”



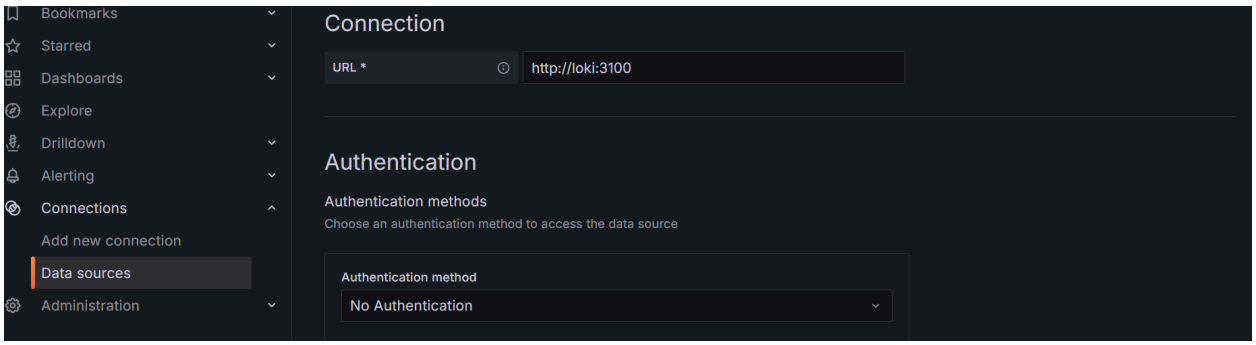
13. Adding data source Loki

1. Navigate to data sources and select add new data source and search for loki

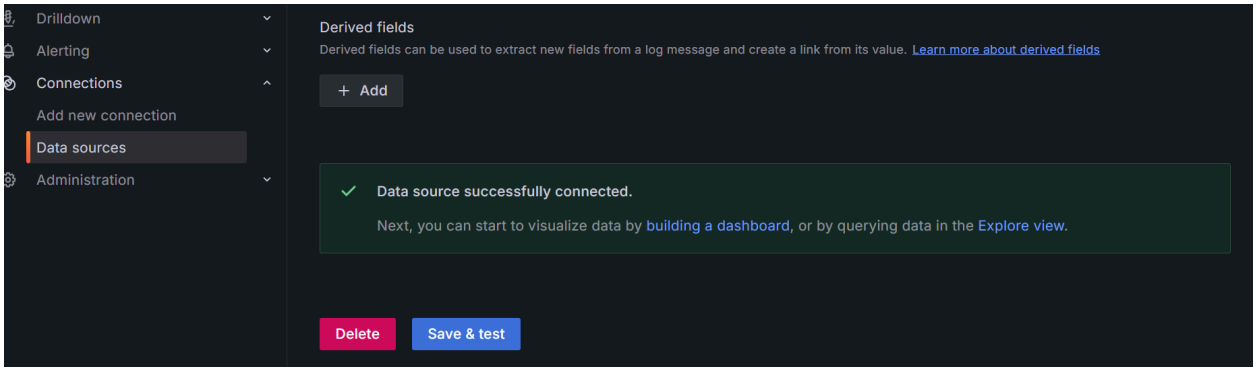


2. Select loki data source
3. configure loki settings

In the URL field, enter: http://loki:9090



4. Leave the other settings as default click “Save & Test”

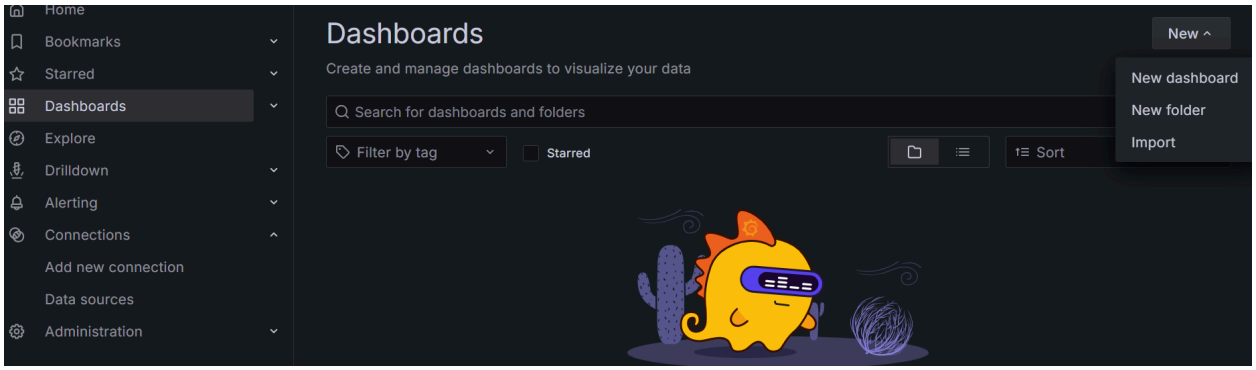


Creating dashboards

We can create dashboards in two ways one is create custom dashboards and another way is to import existing dashboards from grafana labs

14. Importing dashboard to visualise node exporter metrics

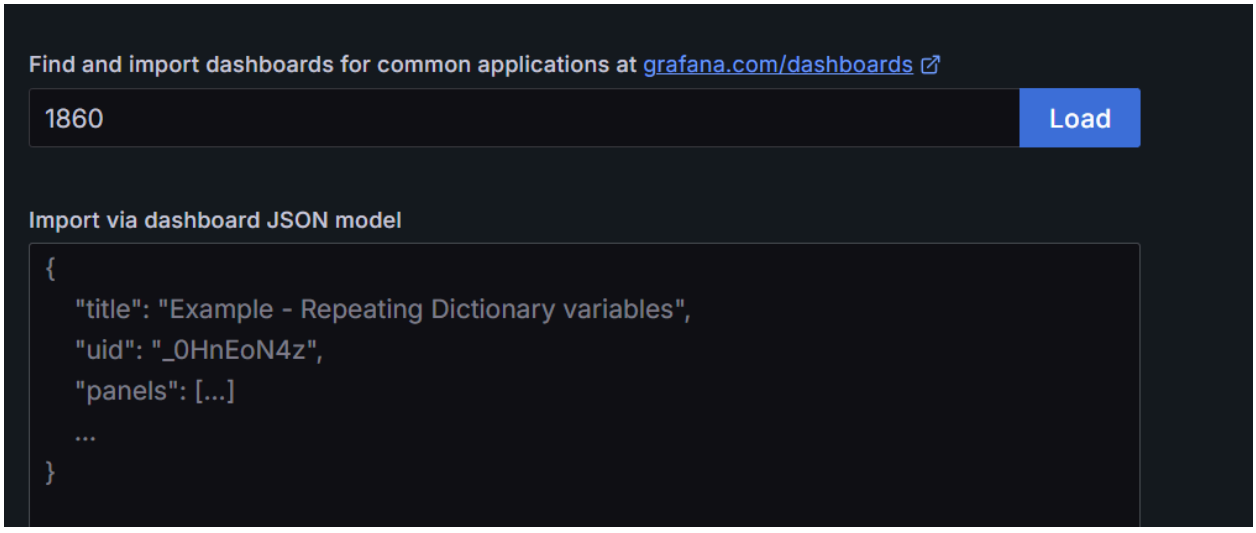
1. Navigate to dashboards > select new > import



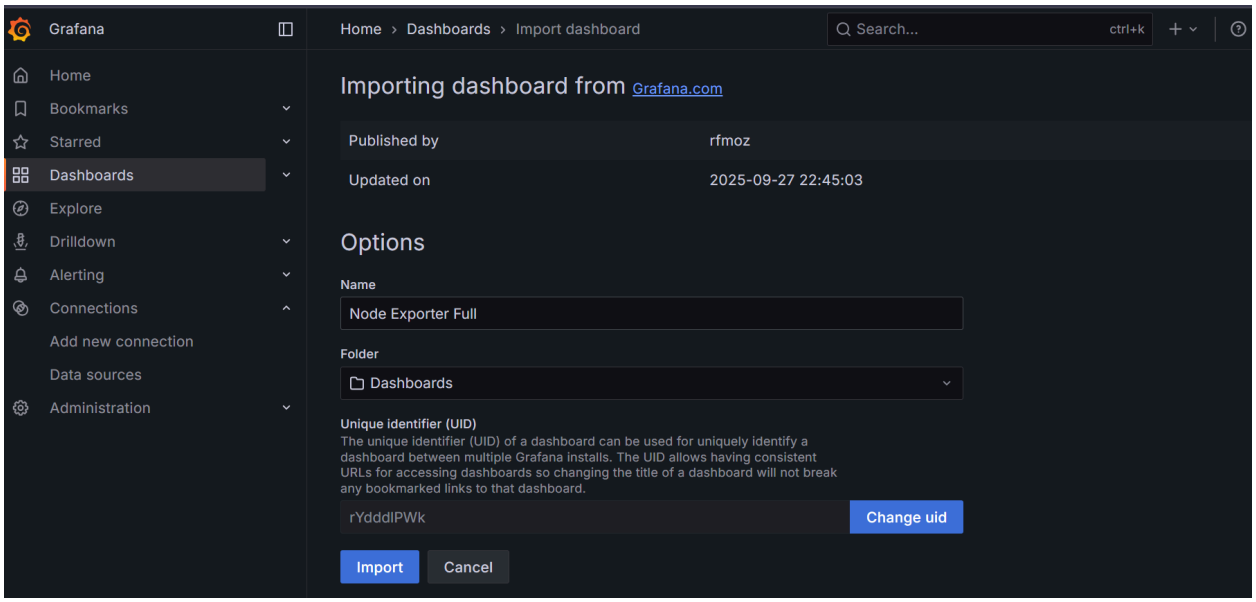
2. Importing node_exporter_full dashboard

Go here: node_exporter_full > Copy Dashboard ID

Paste the ID and click on Load

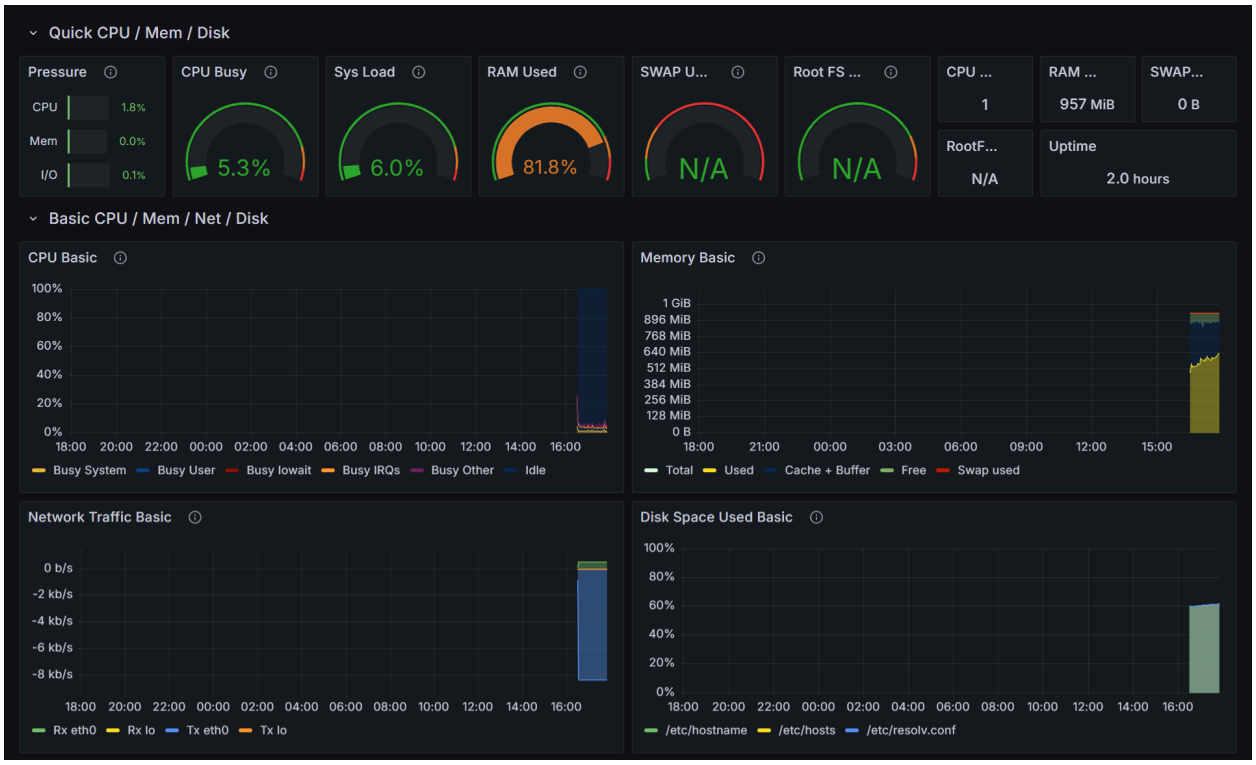


4. Click on import



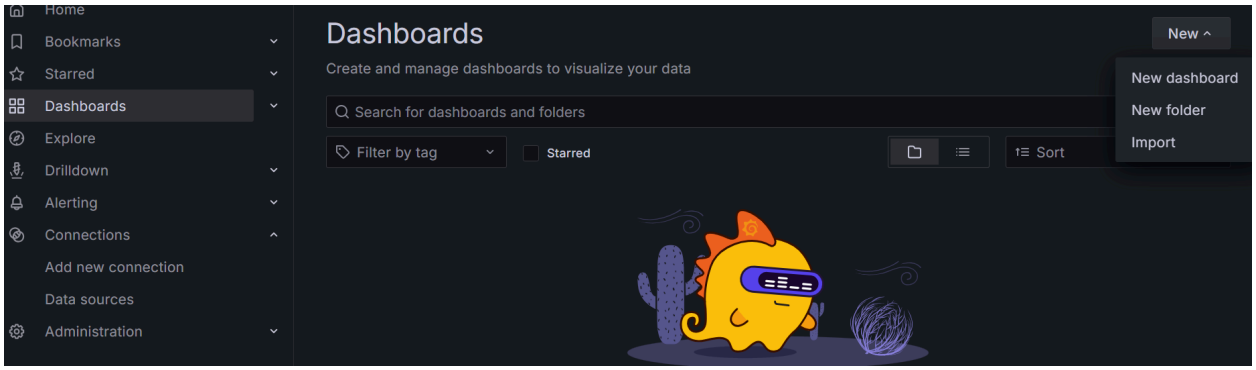
5. Go to dashboards > select Node Exporter Full

This dashboards contains all the server metrics



15. Importing dashboard to visualise Cadvisor metrics

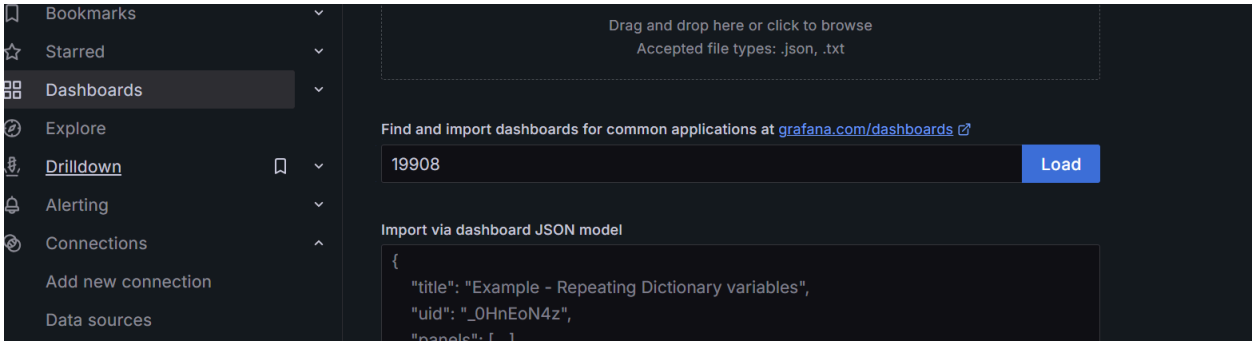
1. Navigate to dashboards > select new > import



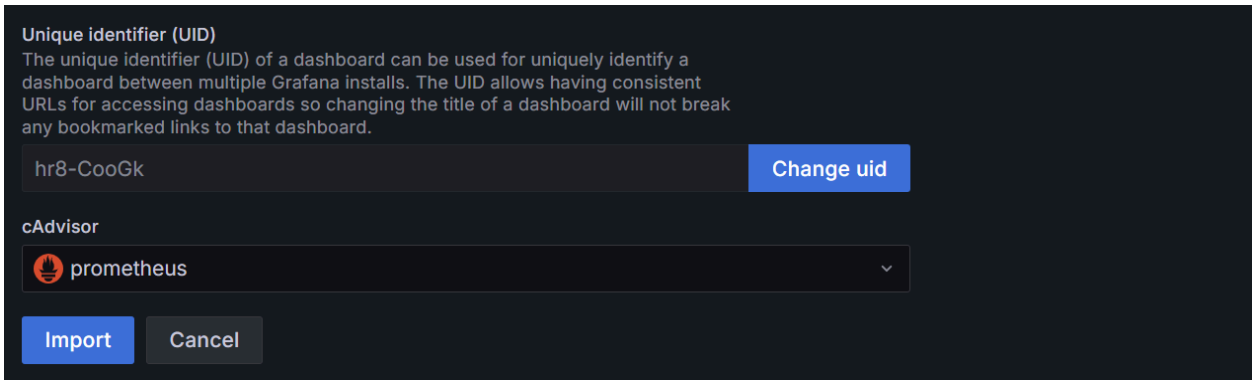
2. Importing Cadvisor dashboard

Go here: Cadvisor > copy Dashboard ID

Paste the ID and click on Load



3. Select the prometheus data source and click on import

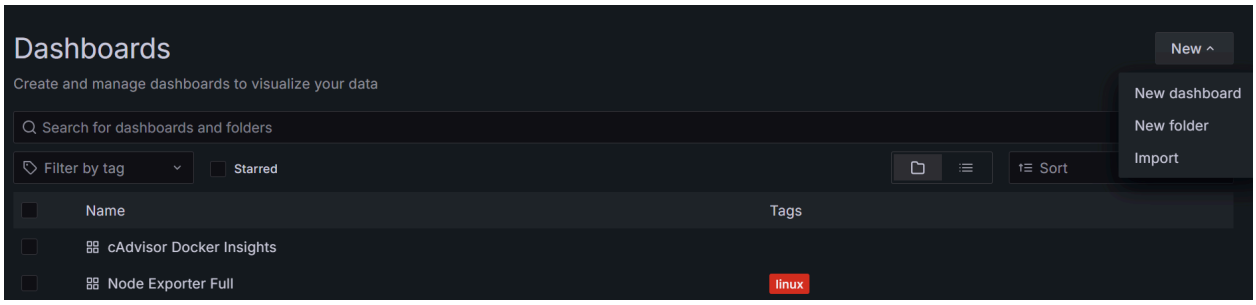


4. Go to dashboards > select cAdvisor Docker Insights

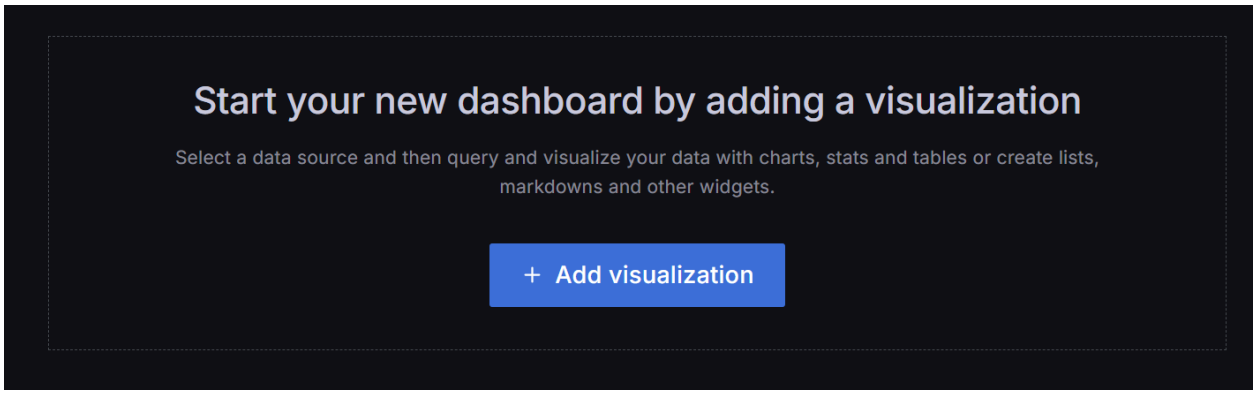
This dashboards contains all the server metrics

15. Creating new dashboard to visualise container logs using loki

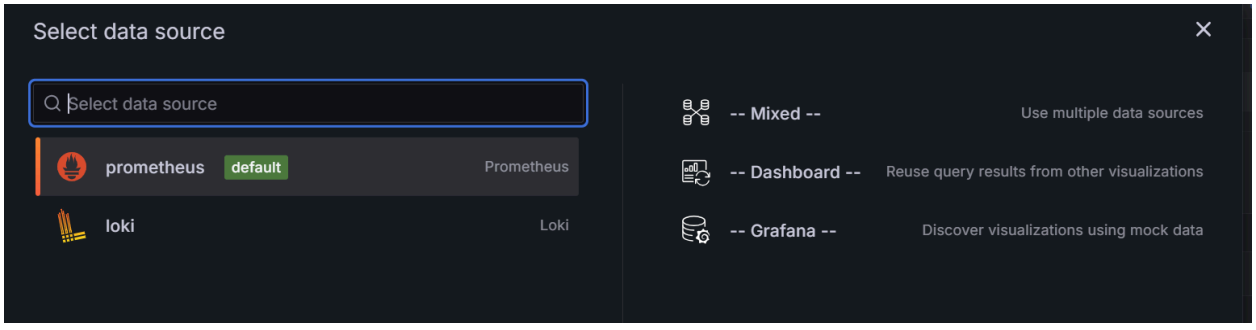
1. Select new dashboard.



2. Click on add visualization

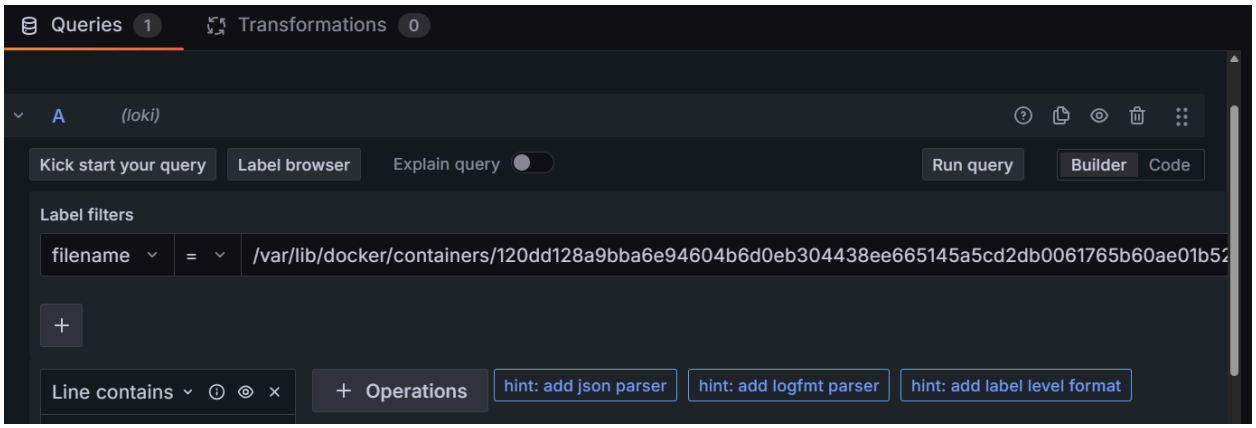


3. Select loki data source.



4. Select label filter as filename and choose the container (promtail container) you want to add and click on run query

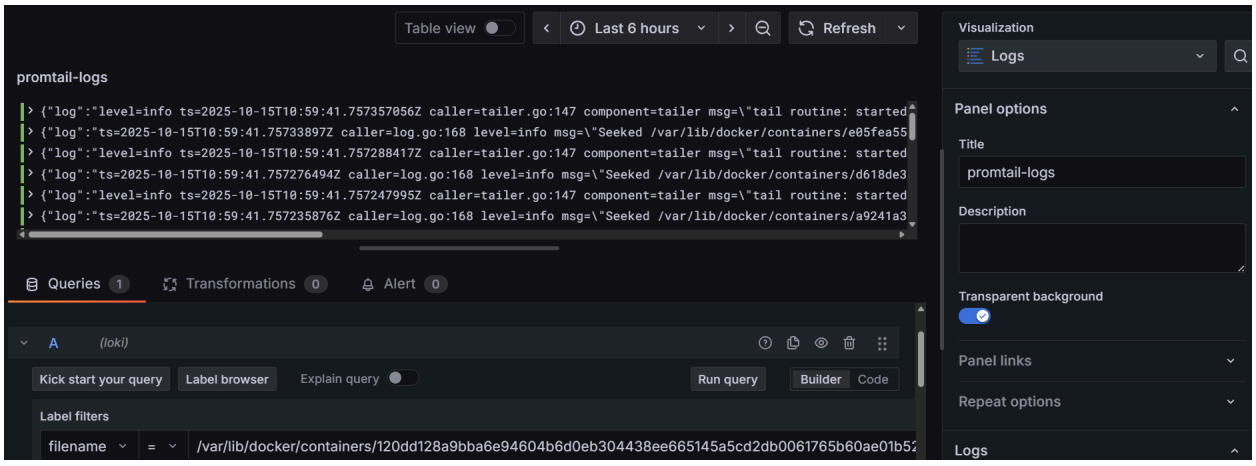
You can also write a LogQL query in the code section.



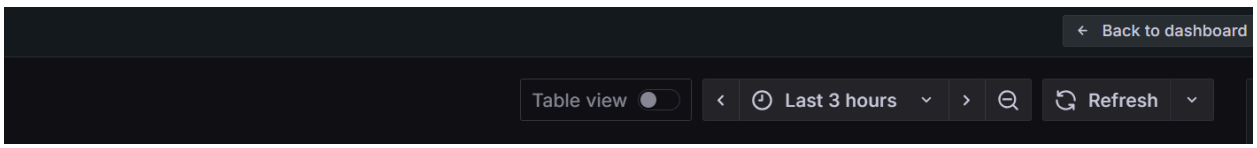
5. Select the visualization type as logs.

Add Title promtail-logs

Add customisations like transparent background, timestamps, pretty json on your choice in the panel options

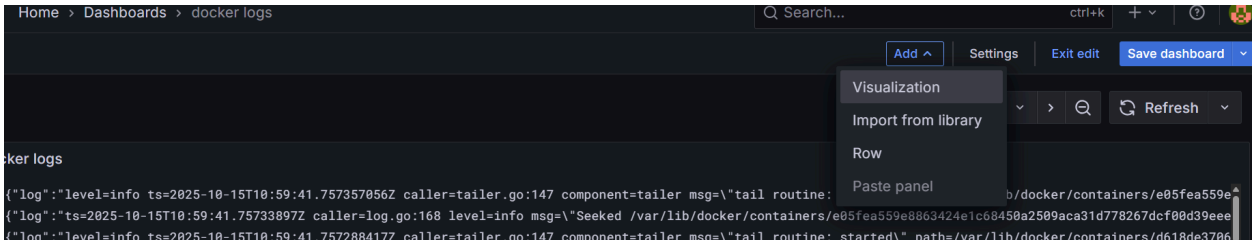


And then click on back to dashboard



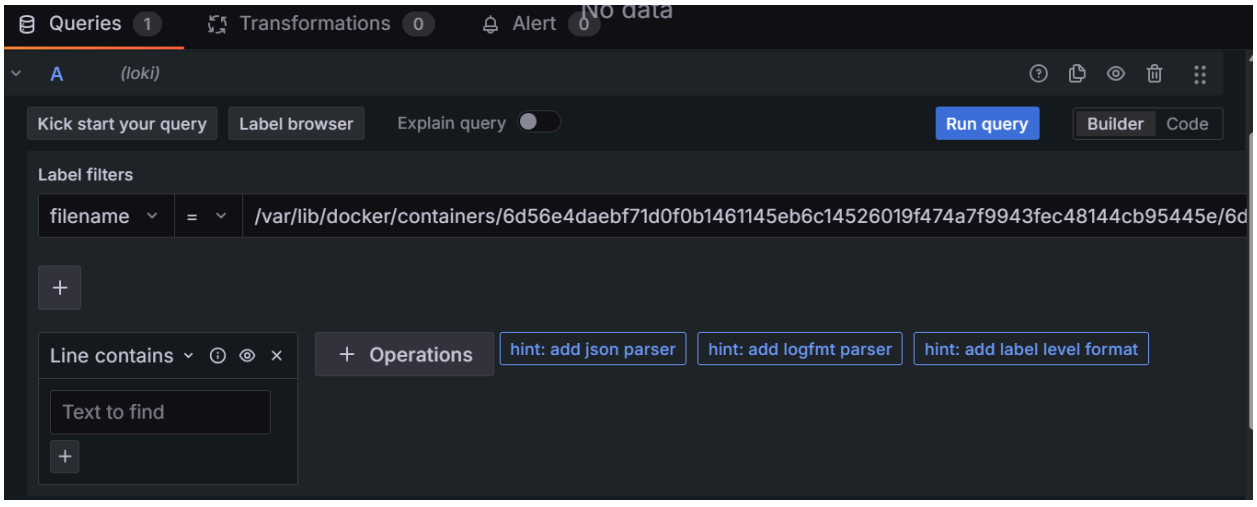
6. Adjust the length and width of the panel on your choice.

7. Click on add > visualization, to create panels for other containers.



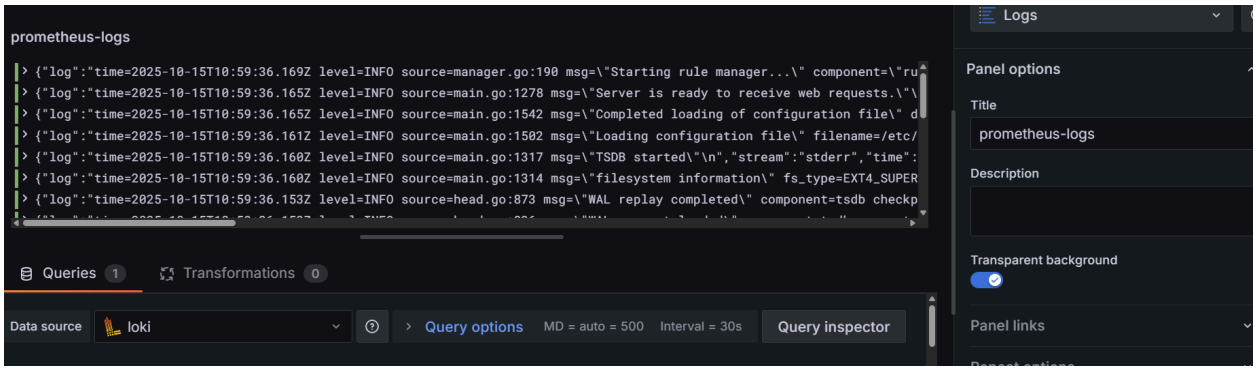
8. Go to query editor and select label as filename and choose container (prometheus) on your choice.

And click on run query

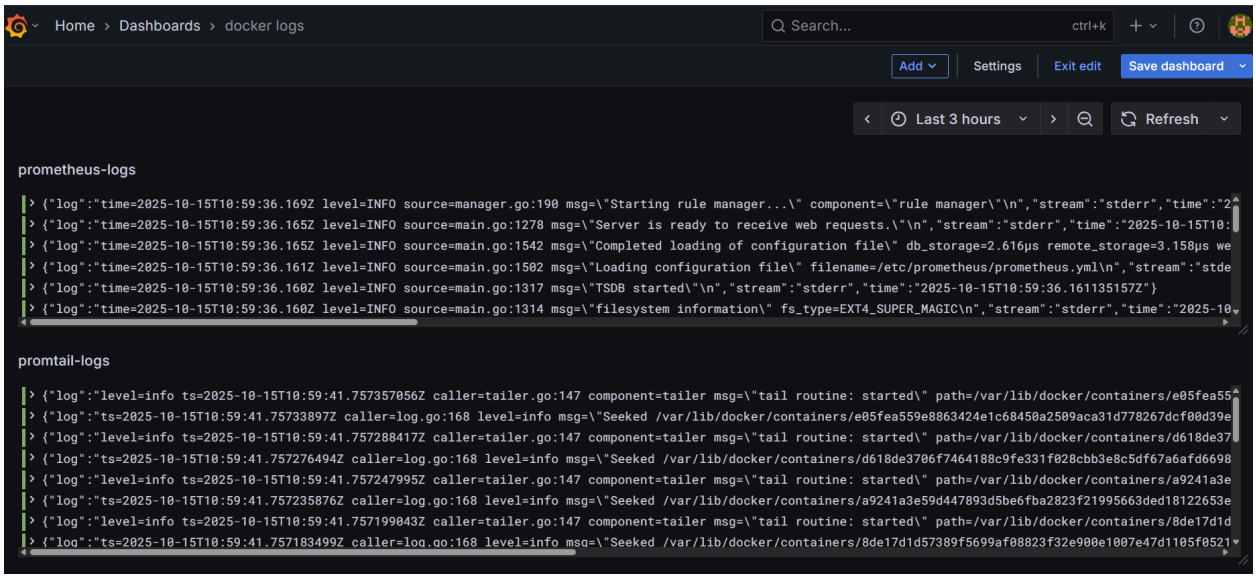


9. Select visualization as Logs.

Add Title prometheus-logs



10. Click on back to dashboard and customize the panel



10. Repeat the same process for all the containers.

All the containers logs got added.

Click on save dashboard.