ENPM818N Final Project

EKS and Monitoring with OpenTelemetry

University of Maryland

7950 Baltimore Avenue

College Park, MD 20742

10-29-2024

Phase 4: Customizing Grafana Dashboards for EKS Monitoring

Sreelekshmy Rengith
UID: 119458815

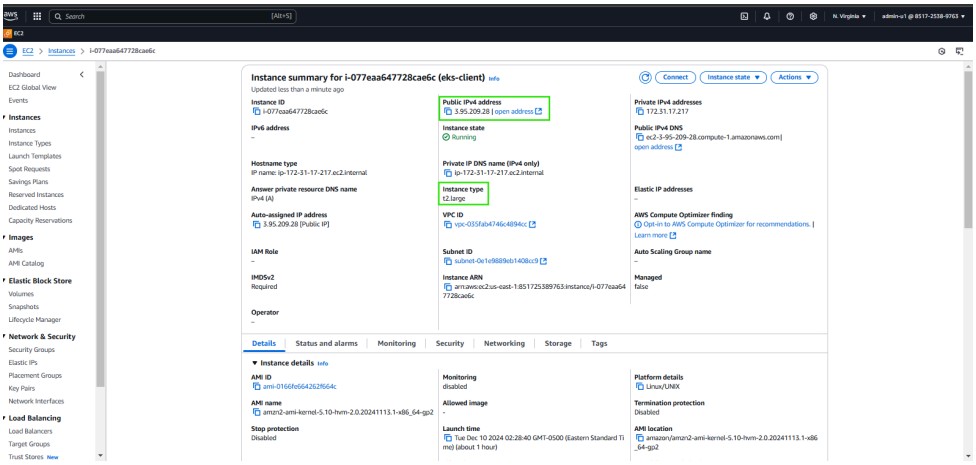
Customizing Grafana Dashboards for EKS Monitoring

Deployment Steps and Monitoring Setup

1. Created an EC2 Instance

We initiated the deployment by launching an EC2 instance with the following specifications:

- Instance Type: 3.medium
- Operating System: Amazon Linux 2
- Security Group: Configured to allow SSH (port 22), HTTP (port 80), and HTTPS (port 443) access.
- IAM Role: Attached an IAM role with permissions for managing EKS and related AWS services.



2. Installed Necessary Tools

After connecting to the instance via SSH, we installed the following tools to manage the Kubernetes cluster and deployments:

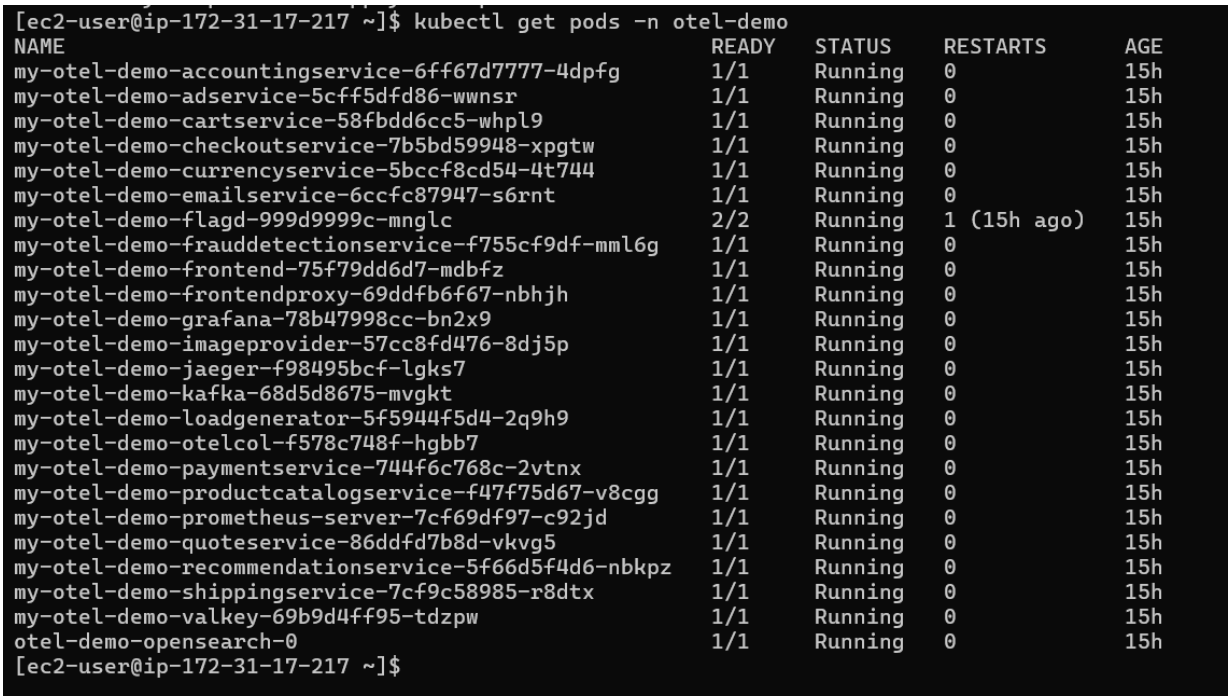
- kubectl: For interacting with Kubernetes resources.
- eksctl: For creating and managing EKS clusters.
- AWS CLI: For interfacing with AWS services.
- Helm: For deploying applications using Helm charts.

3. Created and Configured the EKS Cluster

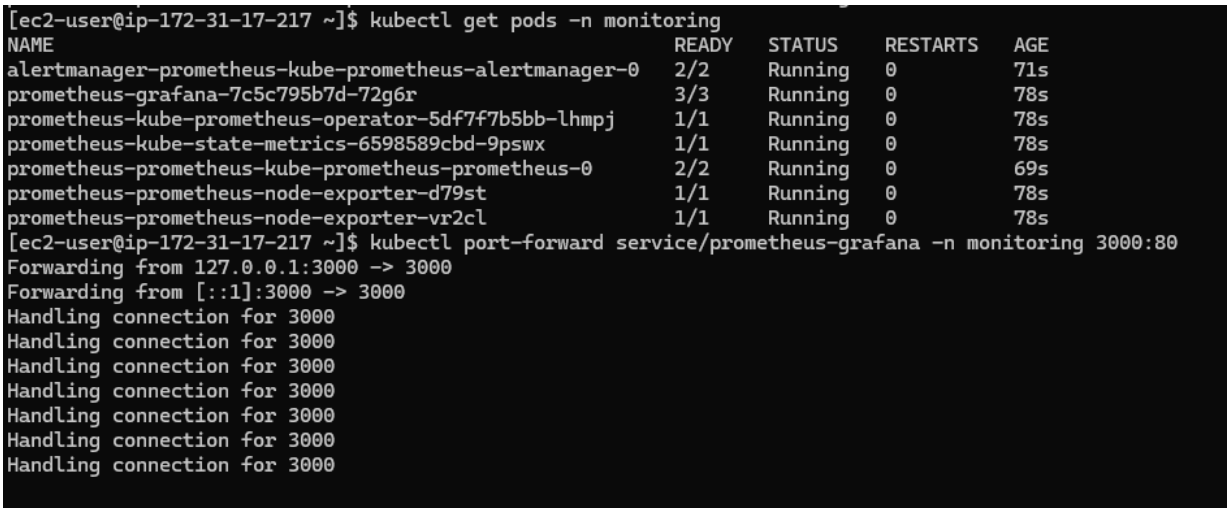
- We used eksctl to create an EKS cluster for deploying the application:

4. Deployed the Application in the Kubernetes Cluster.

- We ensured all pods in the otel-demo and monitoring namespaces were running successfully



```
[ec2-user@ip-172-31-17-217 ~]$ kubectl get pods -n otel-demo
NAME                                                READY   STATUS    RESTARTS      AGE
my-otel-demo-accountingservice-6ff67d7777-4dpfg     1/1     Running   0             15h
my-otel-demo-adservice-5cff5dfd86-wwnsr             1/1     Running   0             15h
my-otel-demo-cartservice-58fbdd6cc5-whpl9           1/1     Running   0             15h
my-otel-demo-checkoutservice-7b5bd59948-xpgtw       1/1     Running   0             15h
my-otel-demo-currencyservice-5bccf8cd54-4t744       1/1     Running   0             15h
my-otel-demo-emailservice-6ccfc87947-s6rnt          1/1     Running   0             15h
my-otel-demo-flagd-999d9999c-mnglc                  2/2     Running   1 (15h ago)   15h
my-otel-demo-frauddetectionservice-f755cf9df-mml6g  1/1     Running   0             15h
my-otel-demo-frontend-75f79dd6d7-mdbfz              1/1     Running   0             15h
my-otel-demo-frontendproxy-69ddfb6f67-nbhjh         1/1     Running   0             15h
my-otel-demo-grafana-78b47998cc-bn2x9               1/1     Running   0             15h
my-otel-demo-imageprovider-57cc8fd476-8dj5p         1/1     Running   0             15h
my-otel-demo-jaeger-f98495bcf-lgks7                 1/1     Running   0             15h
my-otel-demo-kafka-68d5d8675-mvgkt                  1/1     Running   0             15h
my-otel-demo-loadgenerator-5f5944f5d4-2q9h9         1/1     Running   0             15h
my-otel-demo-otelcol-f578c748f-hgbb7                1/1     Running   0             15h
my-otel-demo-paymentservice-744f6c768c-2vtnx        1/1     Running   0             15h
my-otel-demo-productcatalogservice-f47f75d67-v8cgg  1/1     Running   0             15h
my-otel-demo-prometheus-server-7cf69df97-c92jd      1/1     Running   0             15h
my-otel-demo-quoteservice-86ddfd7b8d-vkvg5          1/1     Running   0             15h
my-otel-demo-recommendationservice-5f66d5f4d6-nbkpz 1/1     Running   0             15h
my-otel-demo-shippingservice-7cf9c58985-r8dtx       1/1     Running   0             15h
my-otel-demo-valkey-69b9d4ff95-tdzpw                1/1     Running   0             15h
otel-demo-opensearch-0                              1/1     Running   0             15h
[ec2-user@ip-172-31-17-217 ~]$
```



```
[ec2-user@ip-172-31-17-217 ~]$ kubectl get pods -n monitoring
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0   2/2     Running   0          71s
prometheus-grafana-7c5c795b7d-72g6r                      3/3     Running   0          78s
prometheus-kube-prometheus-operator-5df7f7b5bb-lhmpj     1/1     Running   0          78s
prometheus-kube-state-metrics-6598589cbd-9pswx           1/1     Running   0          78s
prometheus-prometheus-kube-prometheus-prometheus-0       2/2     Running   0          69s
prometheus-prometheus-node-exporter-d79st                1/1     Running   0          78s
prometheus-prometheus-node-exporter-vr2cl                1/1     Running   0          78s
[ec2-user@ip-172-31-17-217 ~]$ kubectl port-forward service/prometheus-grafana -n monitoring 3000:80
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

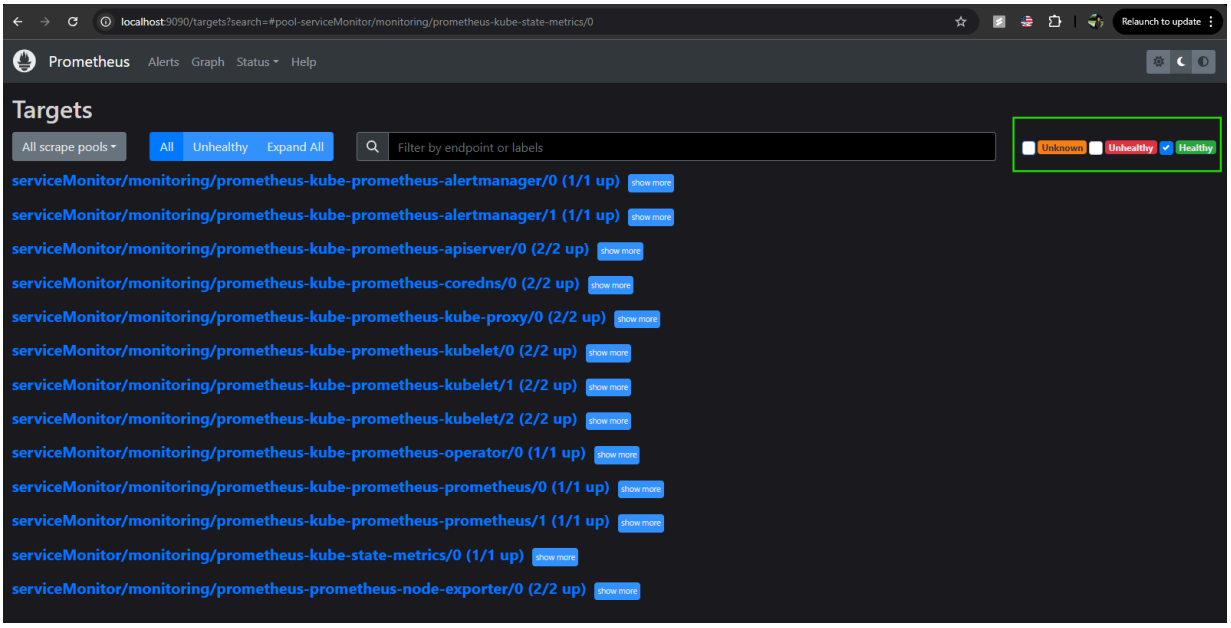
5. Deployed Prometheus and Grafana for Monitoring

- Deployed Prometheus, Grafana, and related components in the monitoring namespace.

6. Configured Prometheus to Scrape Metrics

Prometheus scrapes metrics from healthy, actively running endpoints to monitor system performance. These endpoints, typically exposing data via `/metrics`, provide insights on resource usage (CPU, memory), application-specific metrics, and system health. Using a pull model, Prometheus collects this data at regular intervals and stores it in its time-series

database. This data can be visualized in Grafana or queried with PromQL for analysis and alerts. The confirmed health of all endpoints ensures uninterrupted metric collection, enabling accurate monitoring and proactive issue detection.

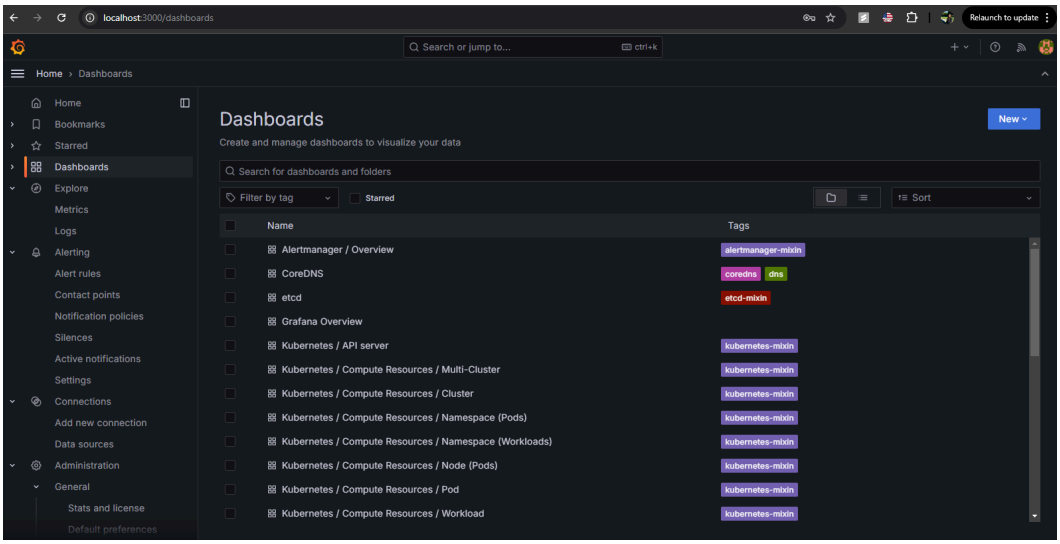


7. Accessed Grafana Dashboard

The command `kubectl port-forward service/prometheus-grafana -n monitoring 3000:80` establishes port-forwarding from port 3000 on the local machine to port 80 of the Grafana service in the `monitoring` namespace. The command output confirms that connections are being forwarded, indicating the port-forwarding is active. This allows access to the Grafana interface by navigating to **http://localhost:3000** in a web browser.

Part (I) Leverage Pre-Built OpenTelemetry Dashboards

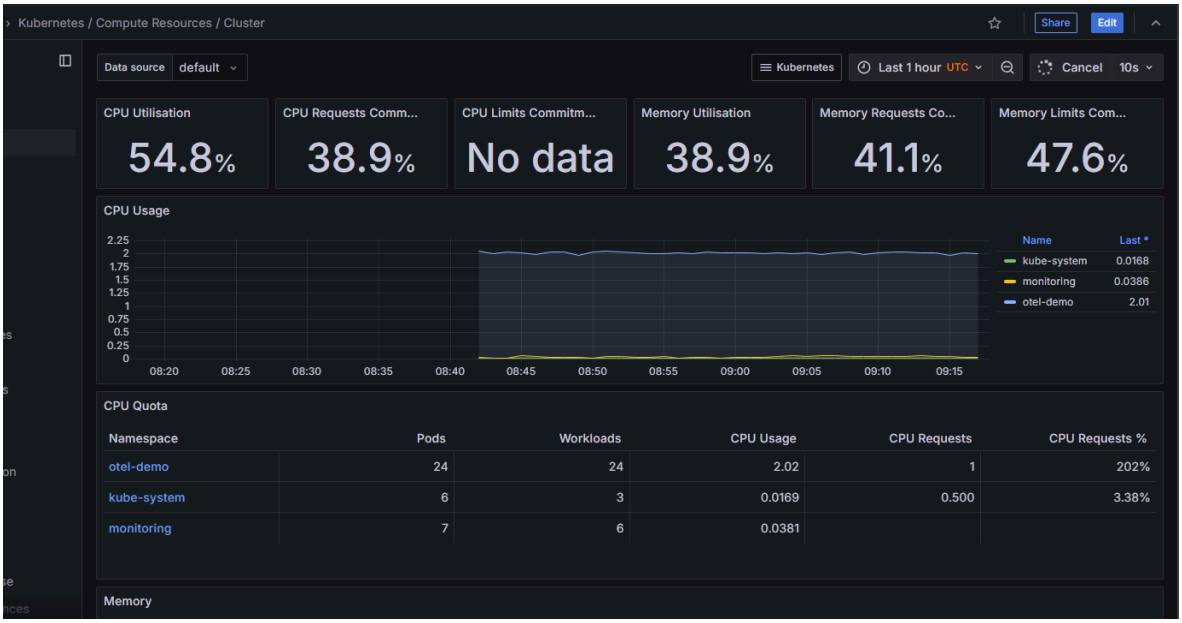
To streamline EKS monitoring, we imported pre-configured dashboards provided by OpenTelemetry into Grafana. These dashboards offered comprehensive visualizations for tracking traces, metrics, and logs collected from the EKS environment. By leveraging these pre-built solutions, we gained immediate insights into cluster health, resource utilization, and system performance, reducing the need for custom configurations and accelerating the monitoring setup process. The default dashboards effectively covered key aspects of node, pod, and service-level metrics, providing a solid foundation for further customization.

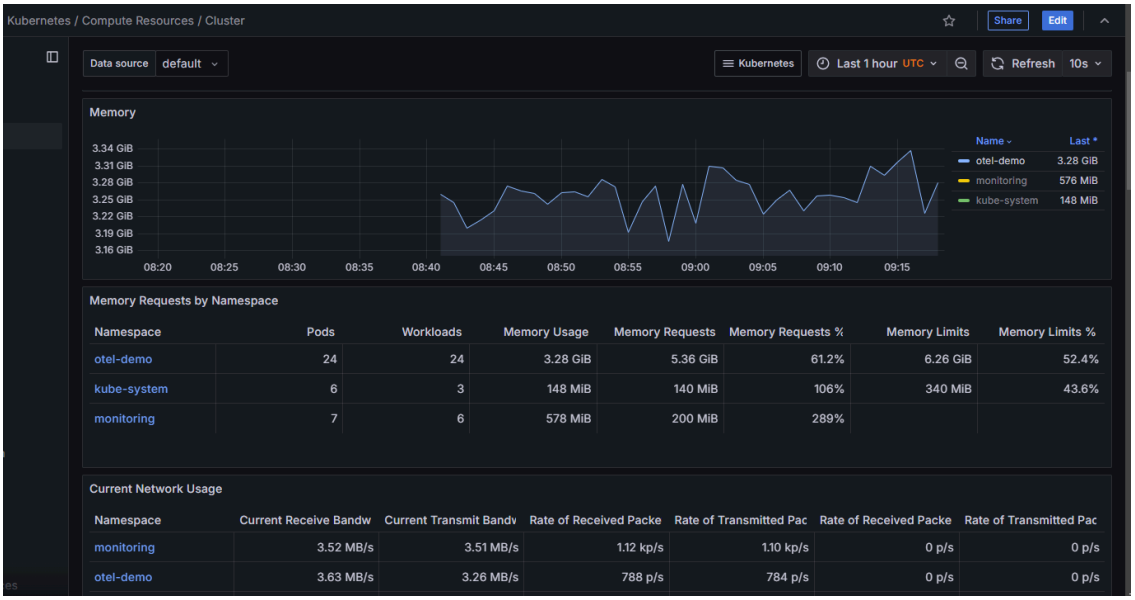


Let's explore few of the default dashboards:-

1. **Kubernetes Cluster Overview**

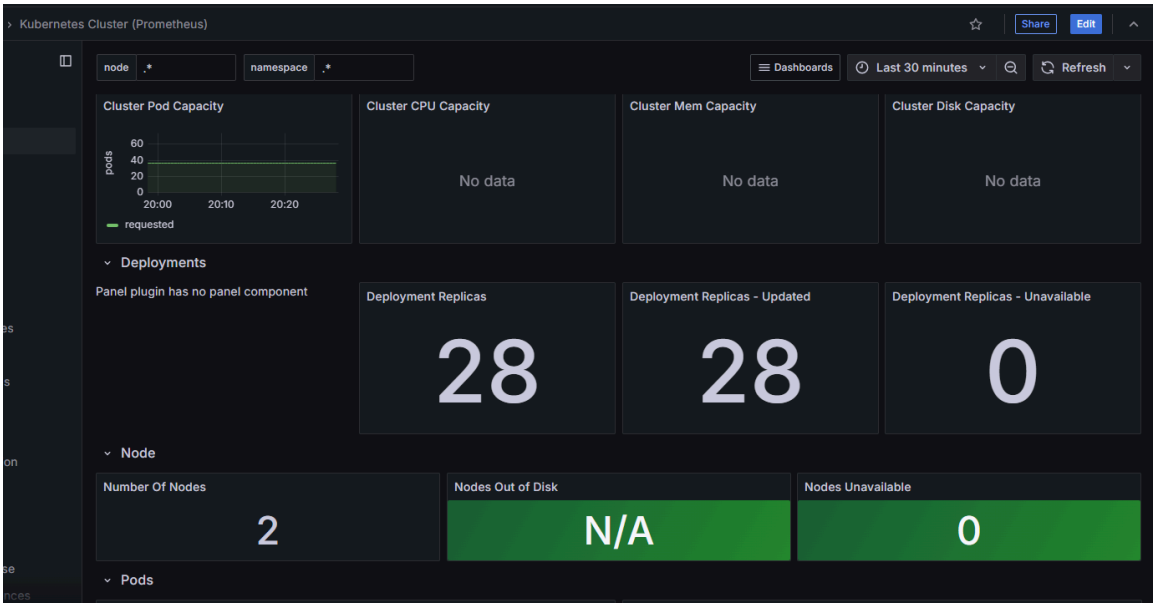
- Dashboard ID: 3119
- Provides an overview of the entire Kubernetes cluster, including CPU, memory, and disk usage, node status, and overall cluster health.





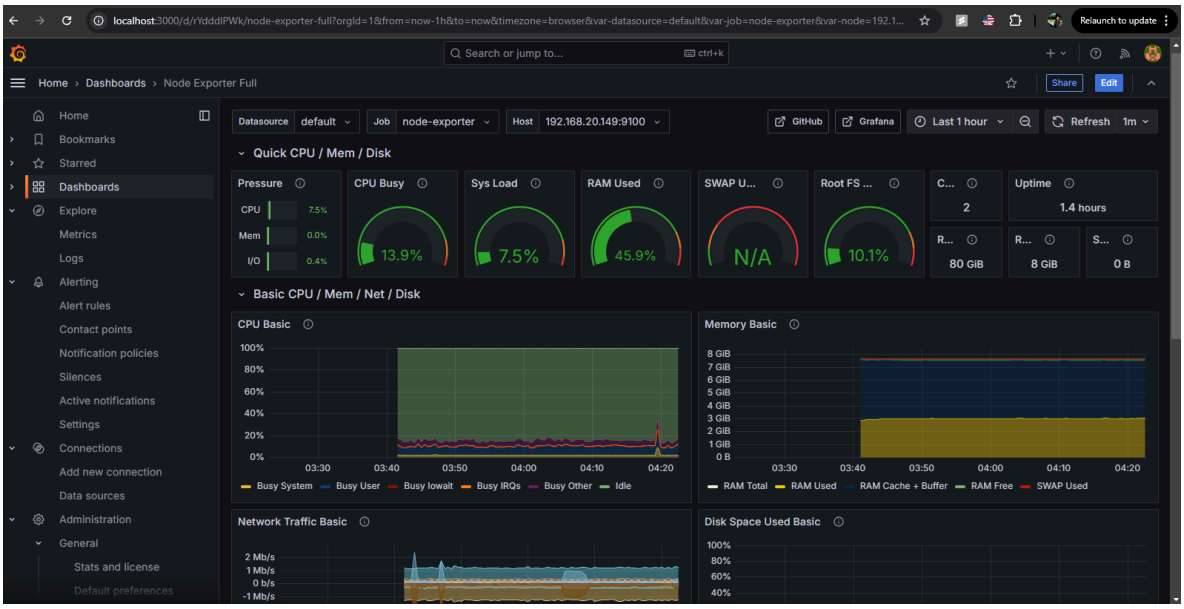
2. Kubernetes Services / Pods

- **Dashboard ID**: 6417
- Visualizes the health, resource usage, and status of services and pods in your Kubernetes cluster



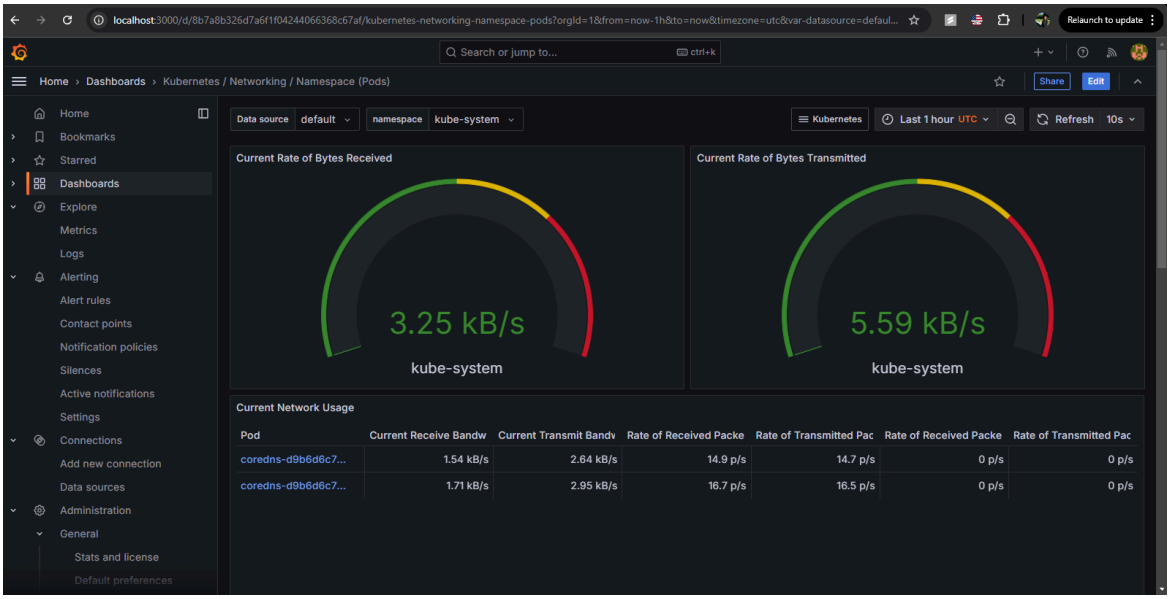
4. Node Exporter Full

- **Dashboard ID**: 1860
- **Description**: Provides detailed metrics for each node, including CPU, memory, disk, and network usage.

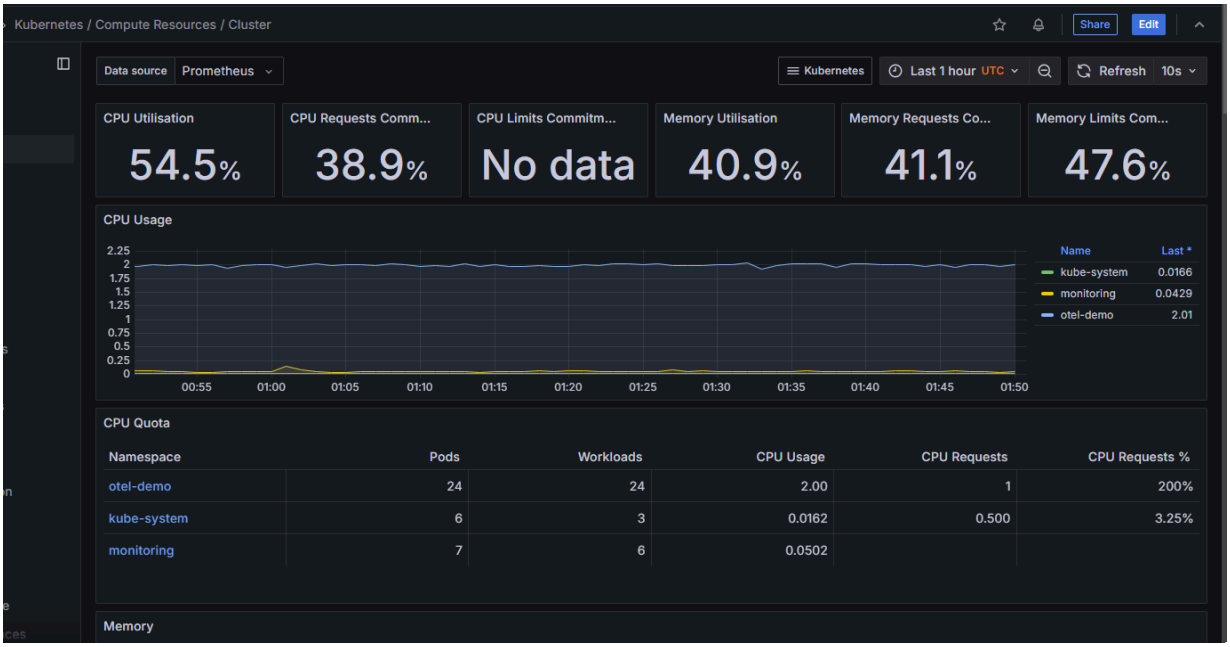


5. Kubernetes Networking Namespace (Pods)

- These Grafana dashboards provide detailed insights into **pod-level metrics** within a **Kubernetes cluster**, segmented by **namespaces** and focusing on networking performance.



Grafana Dashboard Overview: Kubernetes Cluster Resource Utilization



- **Top Panels**:
  - **CPU Utilization**: 54.7%
  - **CPU Requests Commitment**: 38.9%
  - **Memory Utilization**: 38.2%
  - **CPU Limits Commitment**: "No data" (indicating no CPU limits set or metric not collected)
- **CPU Usage Graph**:
  - Displays CPU usage over time segmented by namespace.
  - **Namespaces Monitored**:
    - `kube-system`
    - `monitoring`
    - `otel-demo` (highest consumption at 2.01 cores)
- **CPU Quota Table**:
  - **Metrics per Namespace**:
    - **otel-demo**:
      - **Pods**: 24
      - **Workloads**: 24
      - **CPU Usage**: 2.01 cores
      - **CPU Requests**: 1 core
      - **Request Percentage**: 201% (indicating CPU usage exceeds the requested amount)
- **Key Insights**:
  - Helps monitor overall cluster health and resource consumption.
  - Identifies potential issues such as:

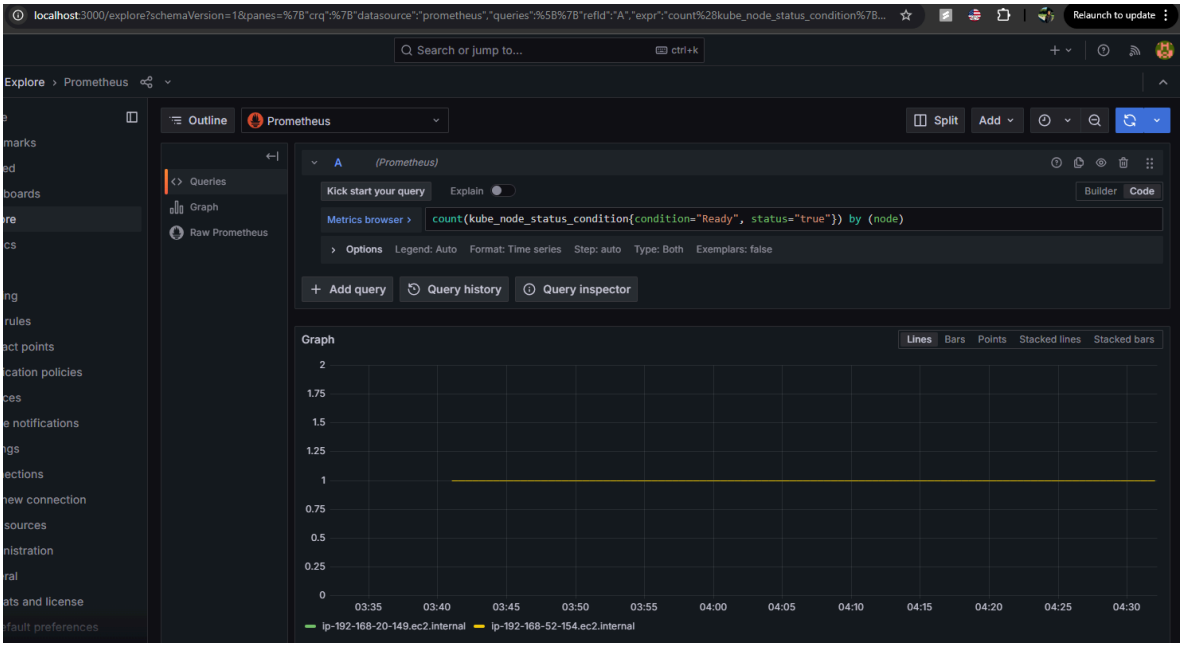
- **Over-utilization** of CPU and memory.
- **Misconfiguration** of resource requests and limits.

Part(II) Customize Dashboards for EKS Metrics

Cluster Health

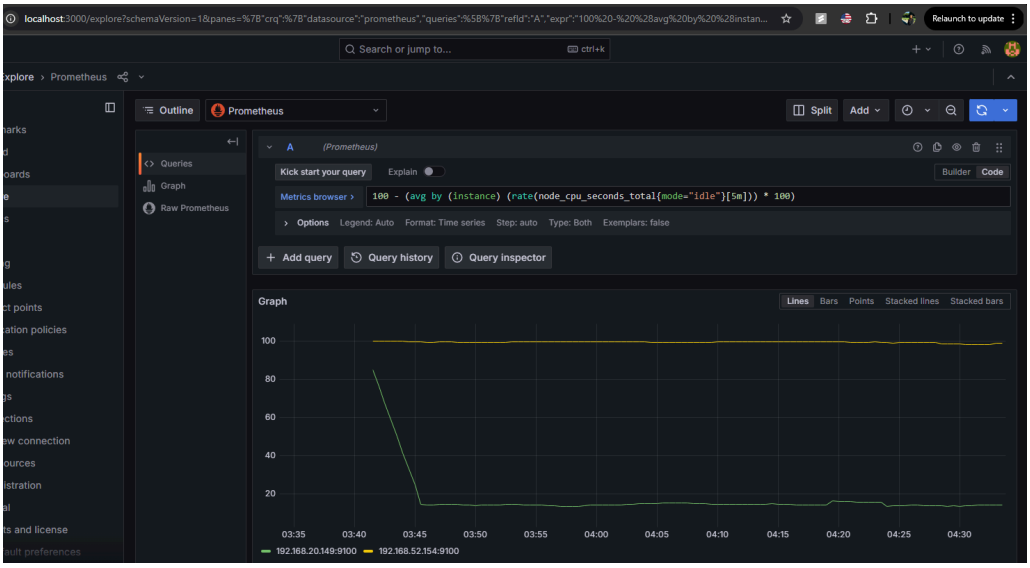
1. Node Status:

count(kube_node_status_condition{condition="Ready", status="true"}) by (node)



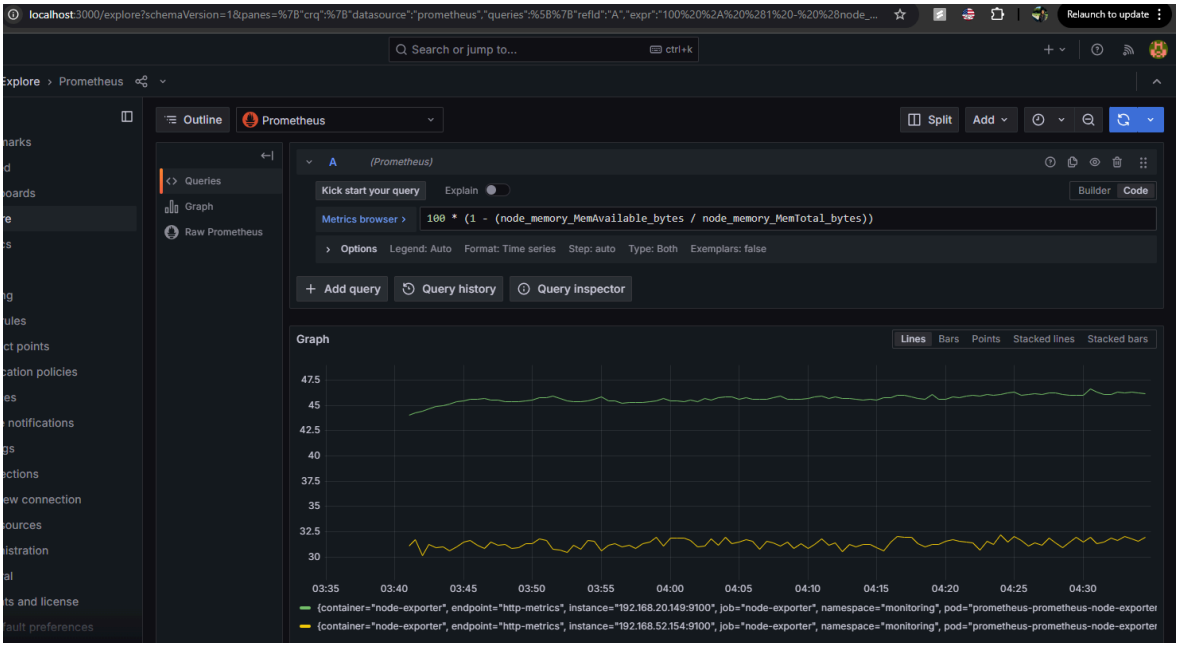
2. **CPU Utilization across nodes**:

100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)



3. Memory Utilization across nodes:

100 * (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes))



(II) Pod-Level Metrics:

CPU Usage per Pod: This query calculates the CPU usage rate for each pod over the last 5 minutes, grouping the result by namespace and pod.

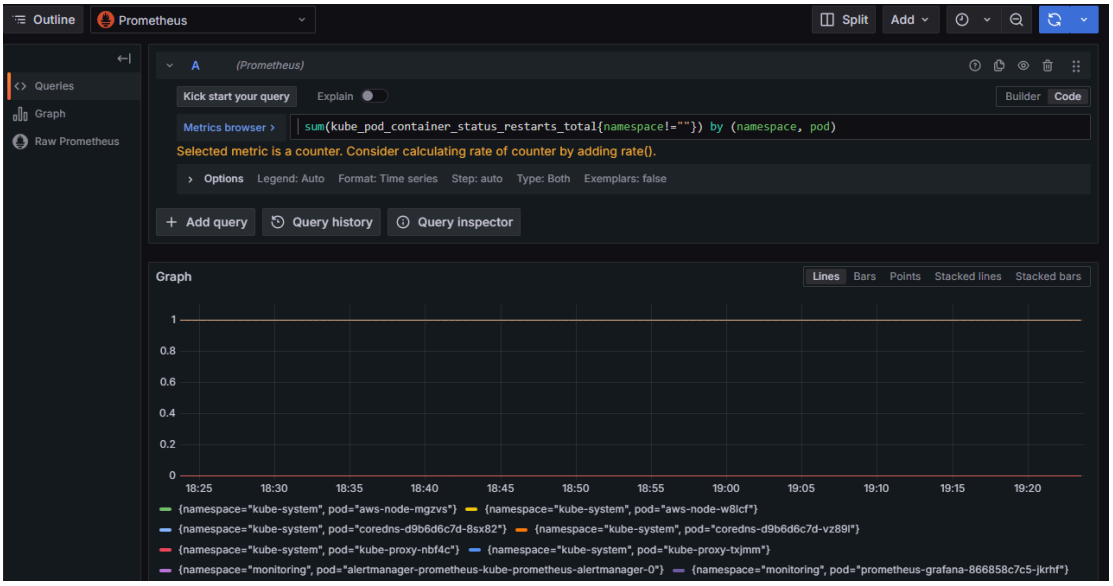
PromQL query: sum(rate(container_cpu_usage_seconds_total{container!="", pod!=""}[5m])) by (namespace, pod)

Memory Usage per Pod: This query shows the current memory usage for each pod, grouped by namespace and pod.

PromQL Query: sum(container_memory_usage_bytes{container!="", pod!=""}) by (namespace, pod)

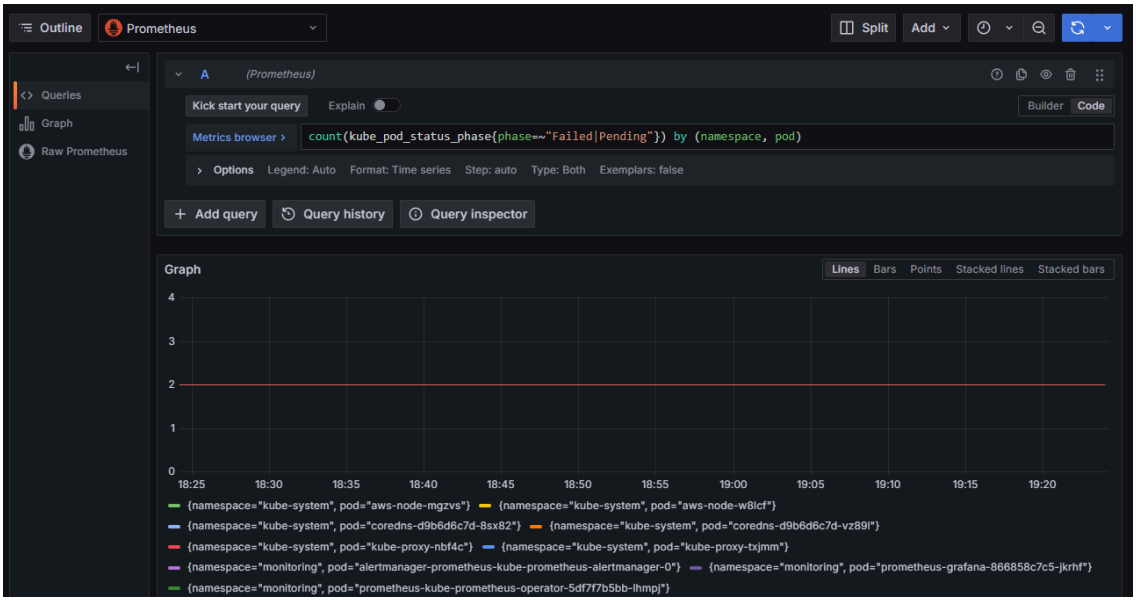
Pod Restarts:
This query returns the total number of container restarts for each pod, grouped by namespace and pod.

PromQL query: sum(kube_pod_container_status_restarts_total{namespace!=""}) by (namespace, pod)



Pod Failures:
This query counts the number of pods in a Failed or Pending state, grouped by namespace and pod.

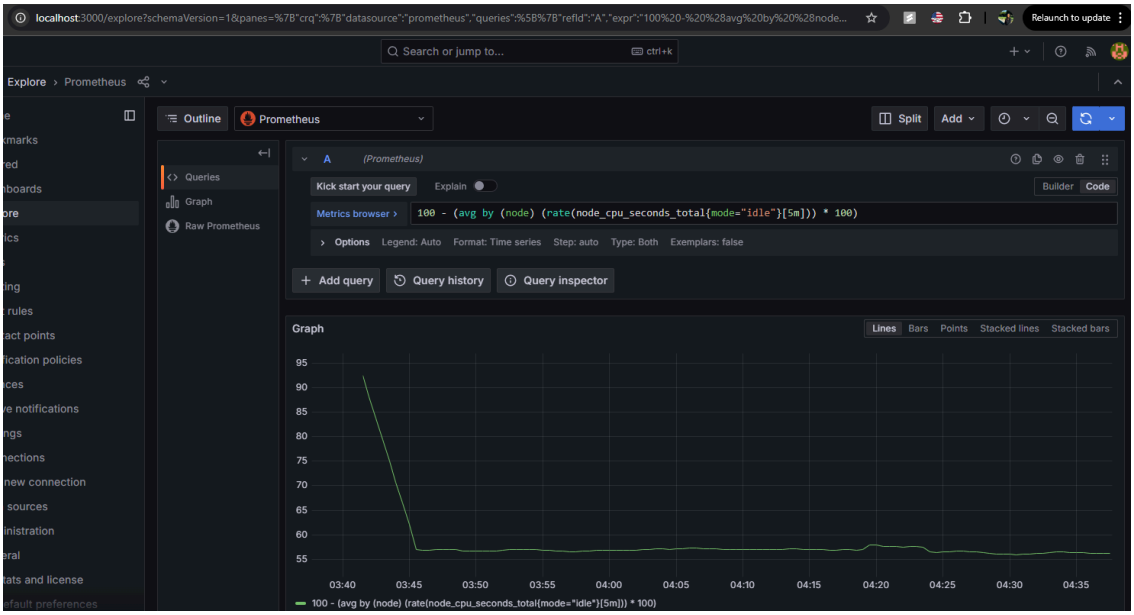
PromQL Query: count(kube_pod_status_phase{phase=~"Failed|Pending"}) by (namespace, pod)



(III) Node Details:

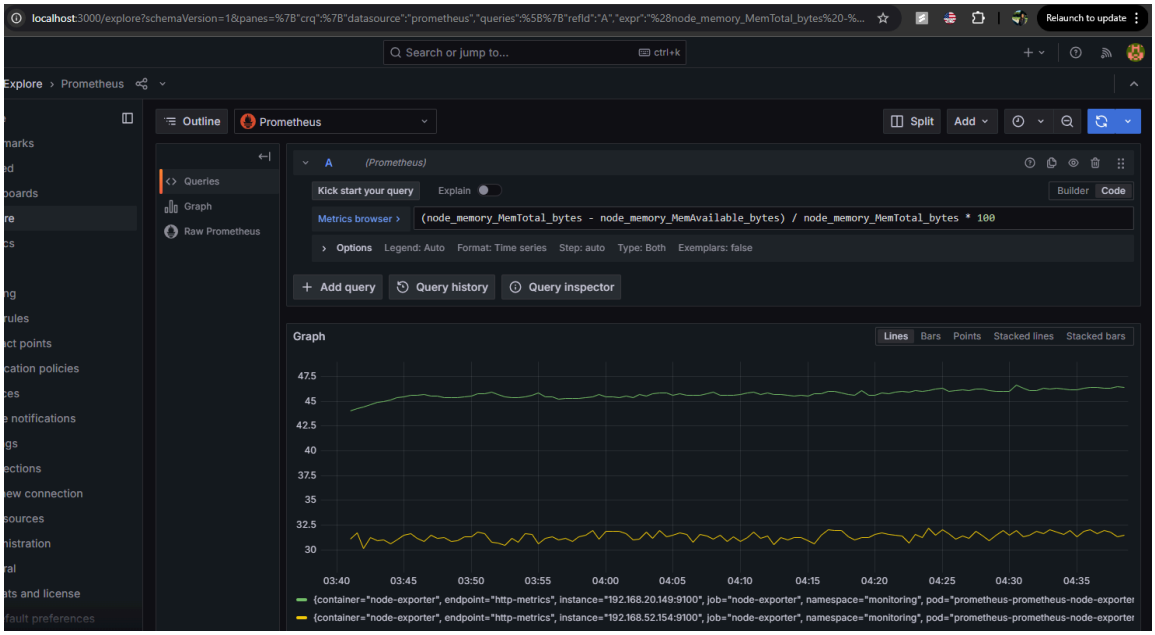
Node CPU Usage:

PromQL Query: 100 - (avg by (node)
(rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

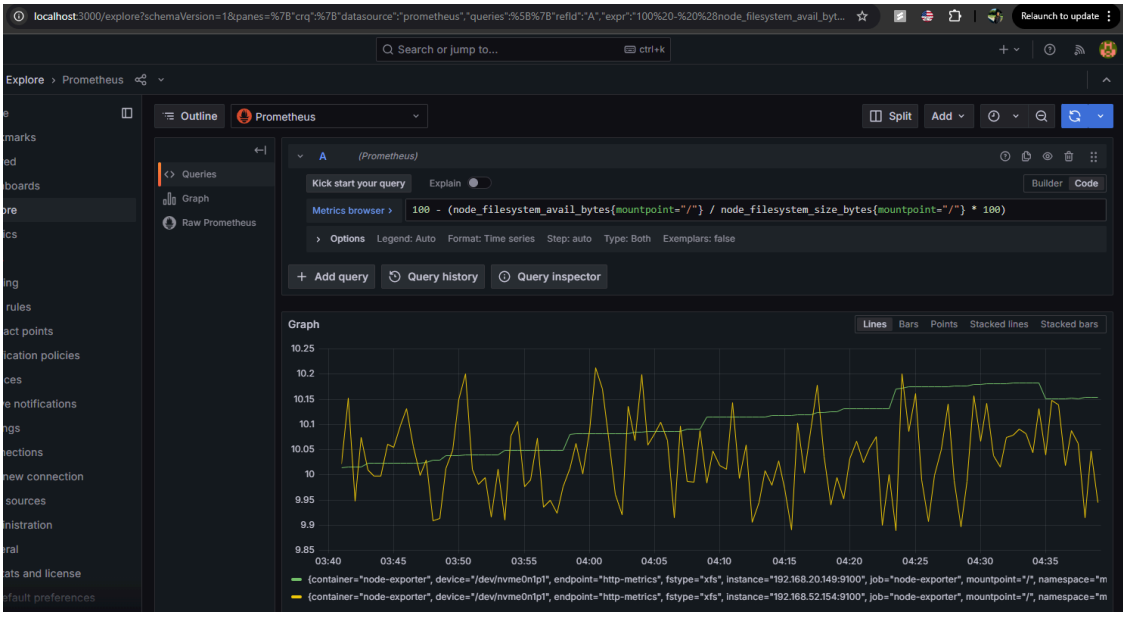


2. Node Memory Usage

(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) /
node_memory_MemTotal_bytes * 100

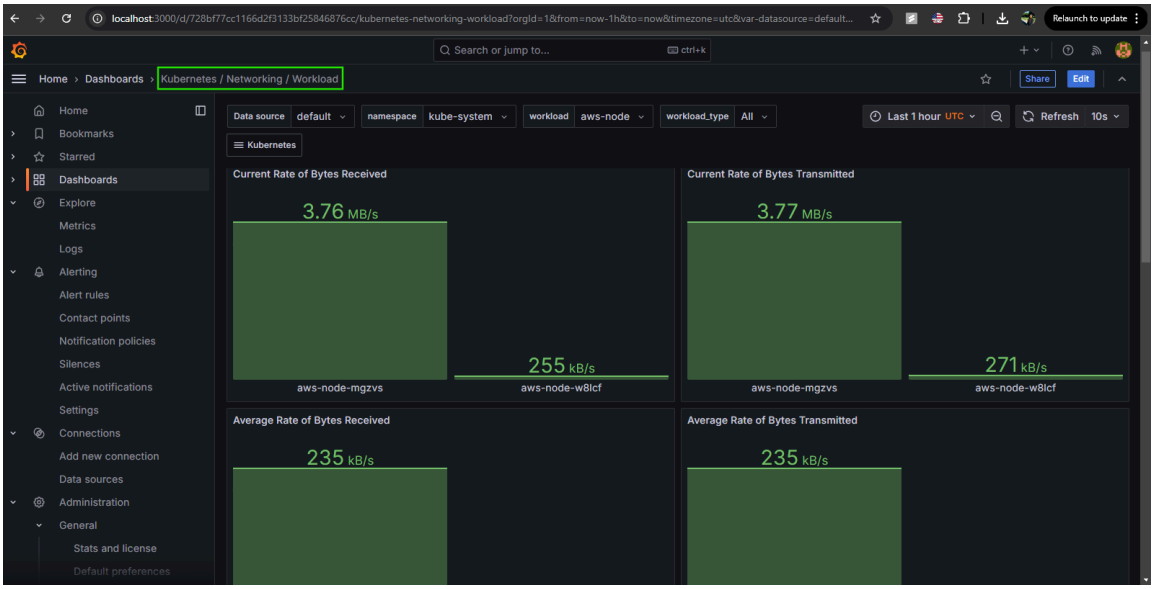


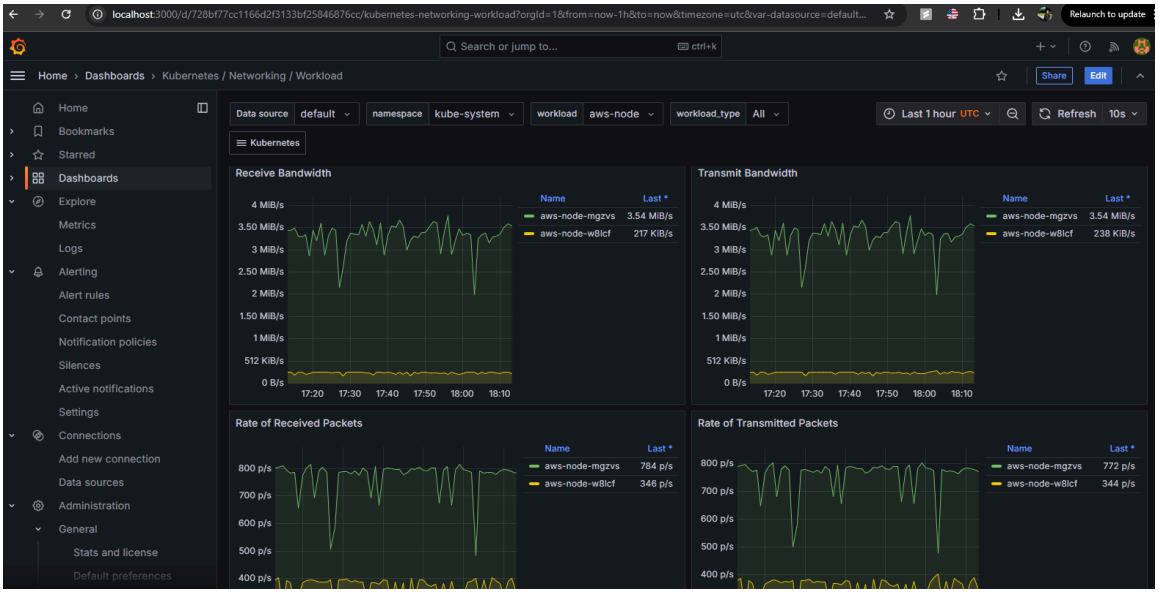
3. Node Disk Space Usage : 100 - (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} * 100)



Part (III) Add Network and Storage Monitoring

Kubernetes Network Monitoring dashboard





Node-Level Network Traffic

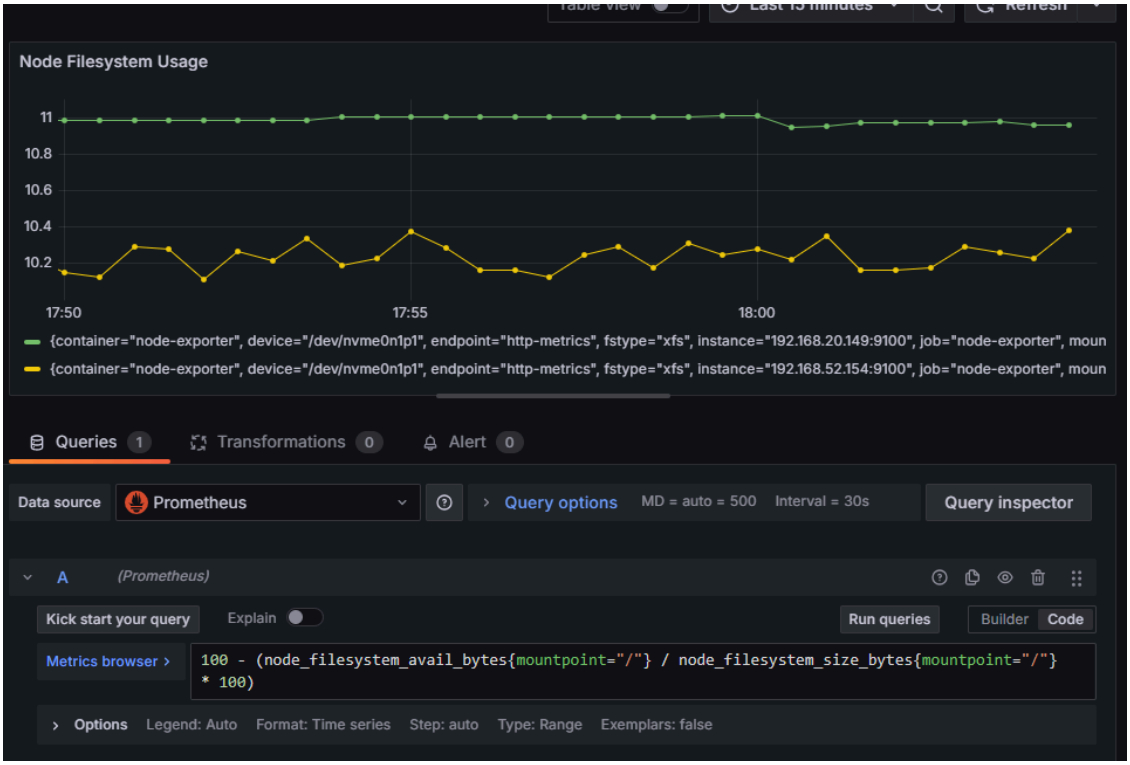
Total Network Traffic Per Node: Sums up both received and transmitted network traffic per node.

PromQL Query:

```
sum(rate(node_network_receive_bytes_total[5m])) by (instance) +
sum(rate(node_network_transmit_bytes_total[5m])) by (instance)
```

Node Filesystem Usage

PromQL Query:
100 - (node_filesystem_avail_bytes{mountpoint="/"} /
node_filesystem_size_bytes{mountpoint="/"} * 100)

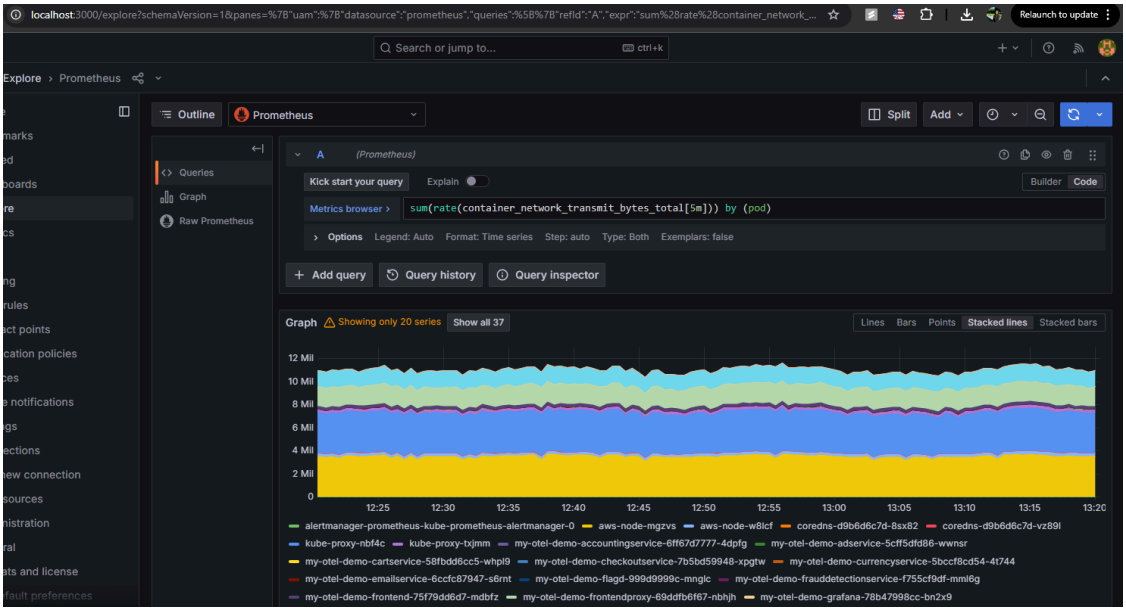


Pod Network :

Bytes Received by Pod (Ingress) : PromQL Query:

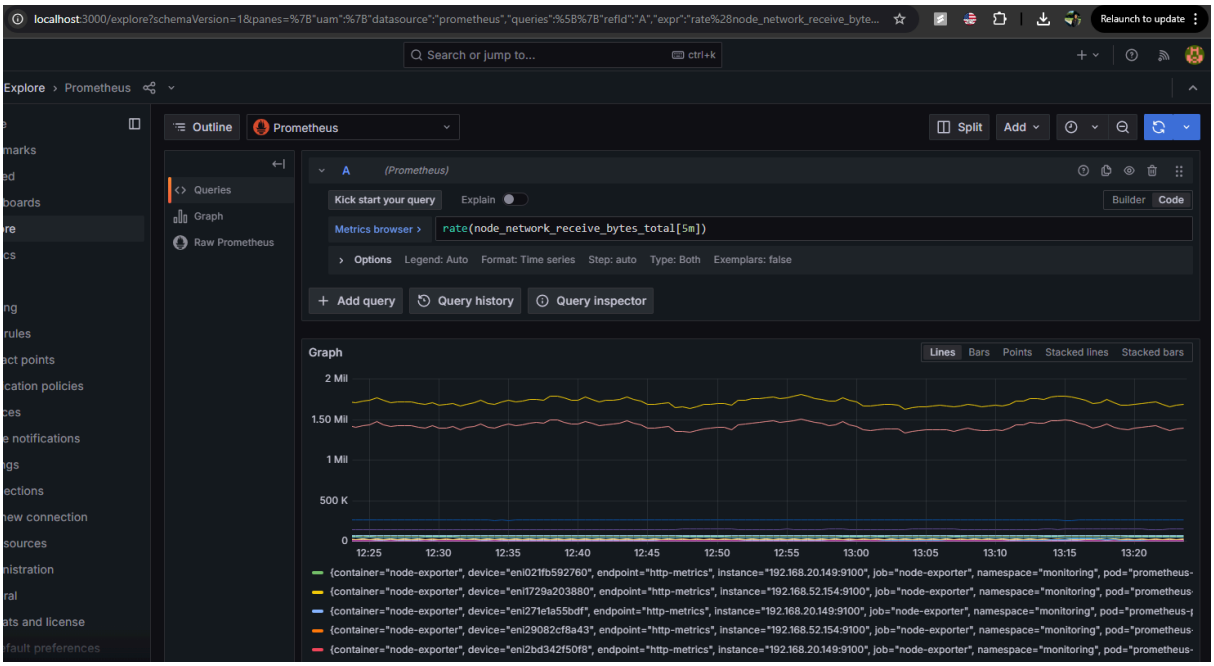
sum(rate(container_network_receive_bytes_total[5m])) by (pod)

Bytes Transmitted by Pod (Egress): PromQL Query:
sum(rate(container_network_transmit_bytes_total[5m])) by (pod)



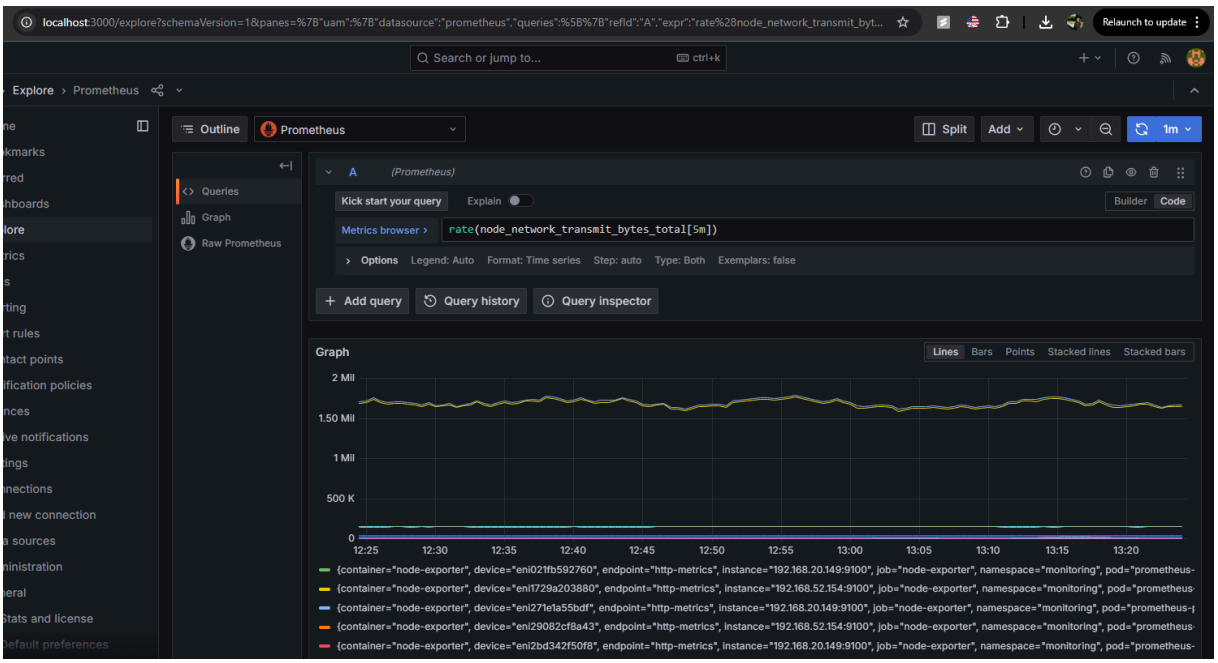
Node Network

Bytes Received by Node
PromQL Query: rate(node_network_receive_bytes_total[5m])



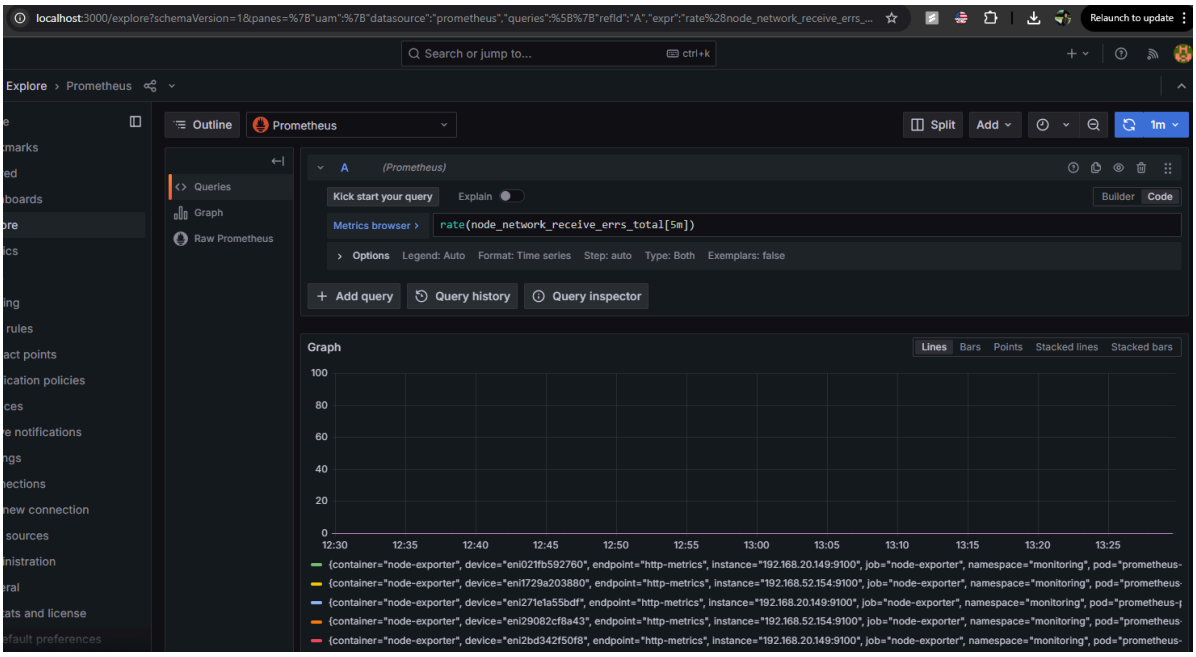
Bytes Transmitted by Node

PromQL Query: rate(node_network_transmit_bytes_total[5m])



Network Errors:

PromQL Query: rate(node_network_receive_errs_total[5m])

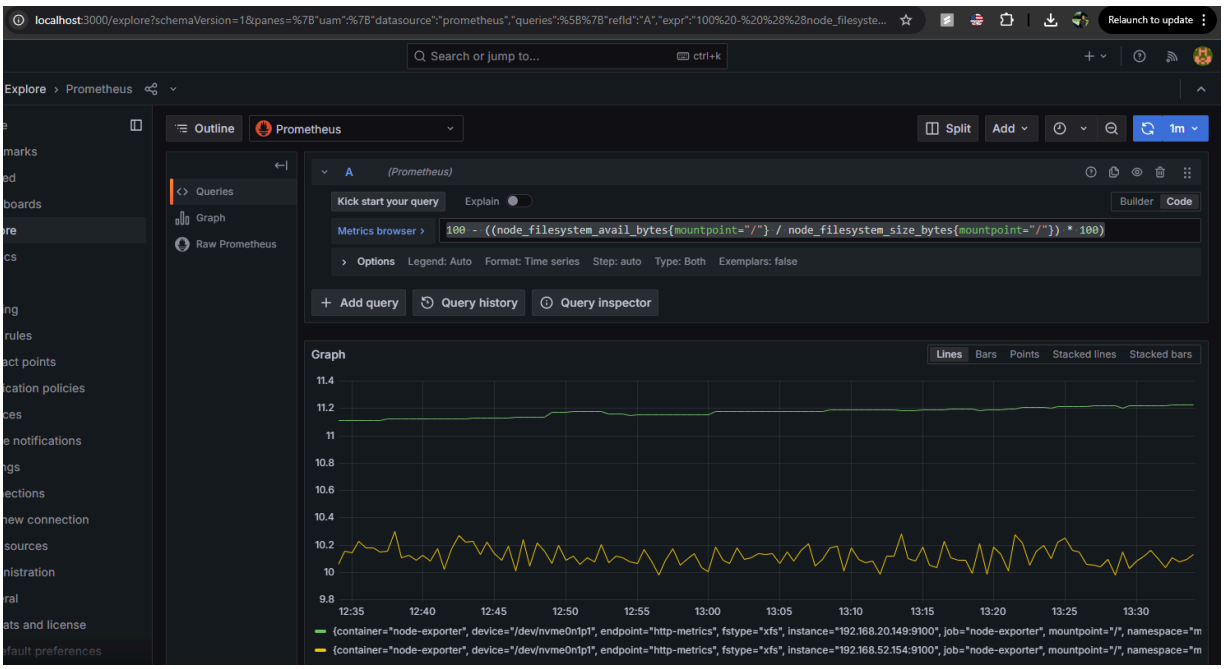


The above chart depicts no error.

Node Disk

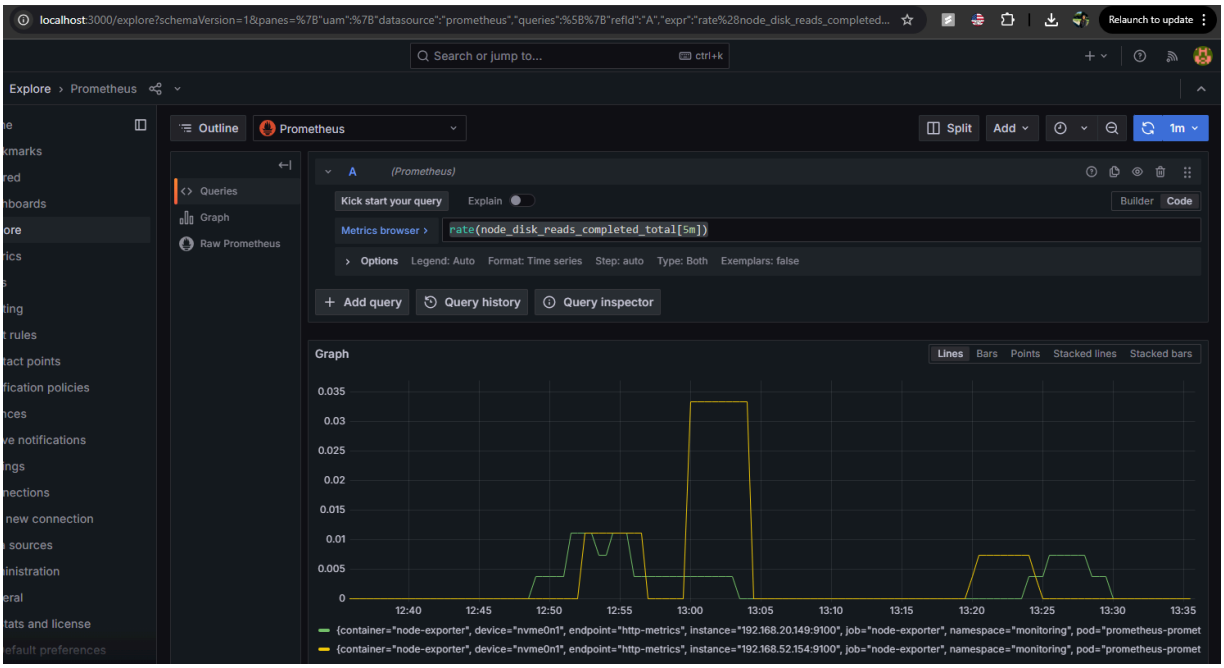
Disk Space Usage

PromQL Query: 100 - ((node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint="/"}) * 100)



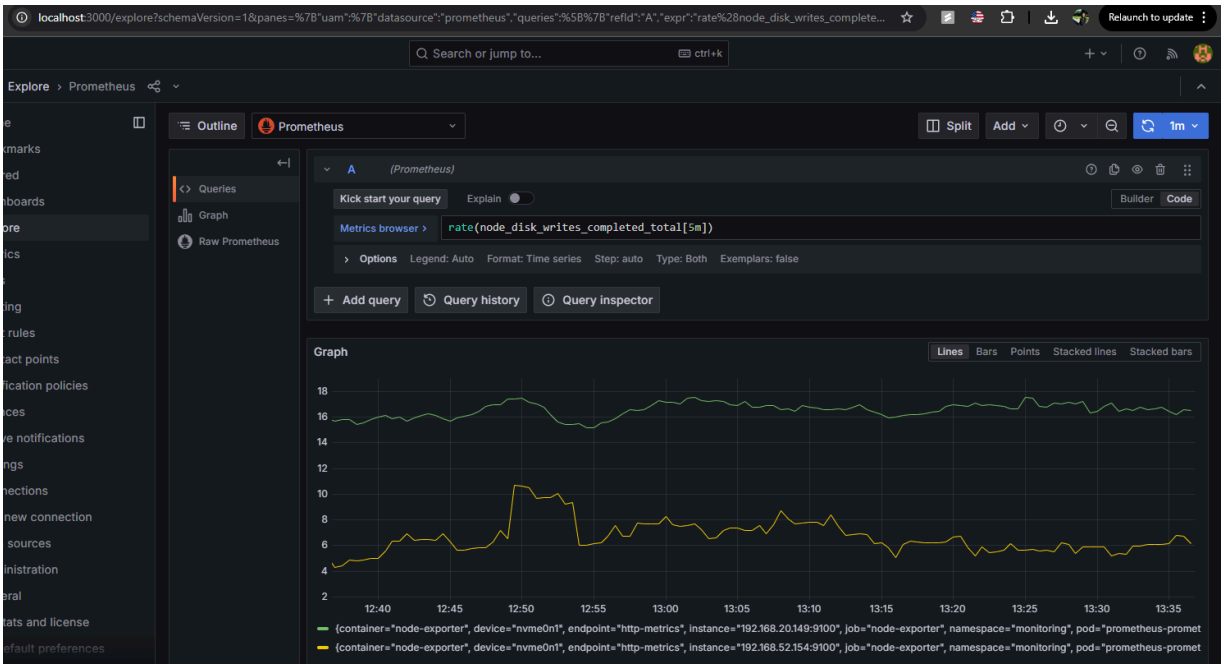
Disk Read IOPS

PromQL Query: rate(node_disk_reads_completed_total[5m])



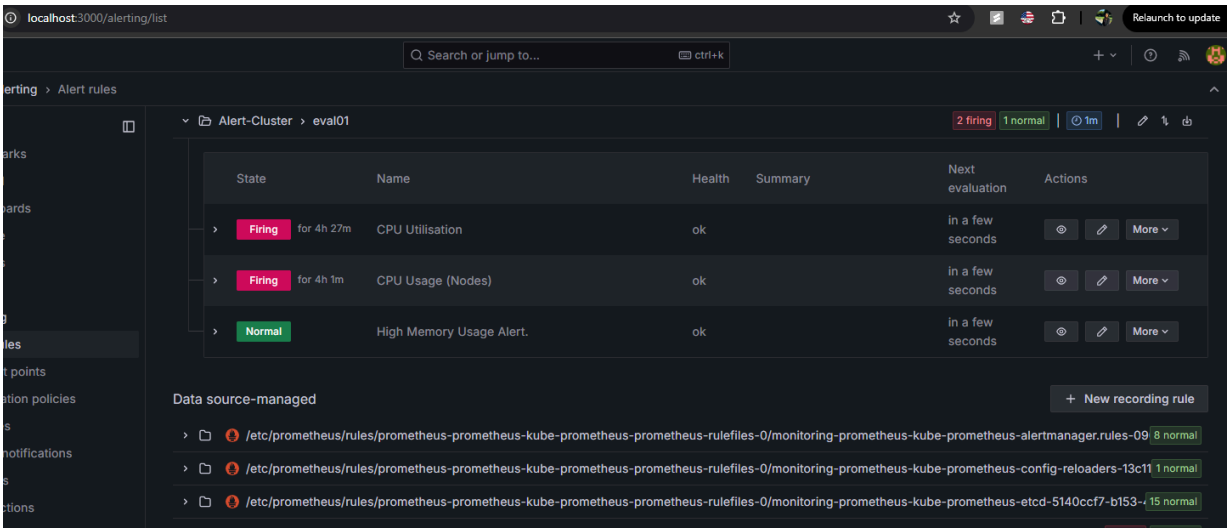
Disk Write IOPS

PromQL Query: rate(node_disk_writes_completed_total[5m])



Part (IV) Set Up Alerts in Grafana

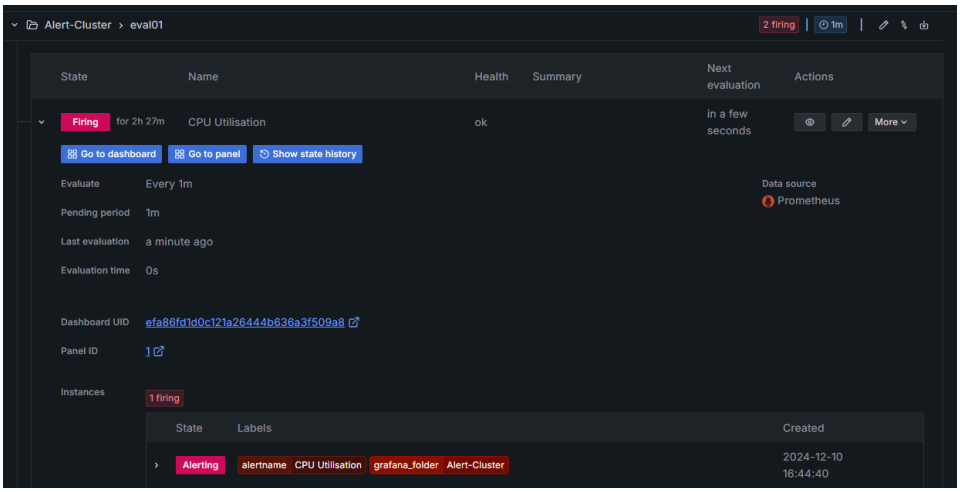
Alert Rules:



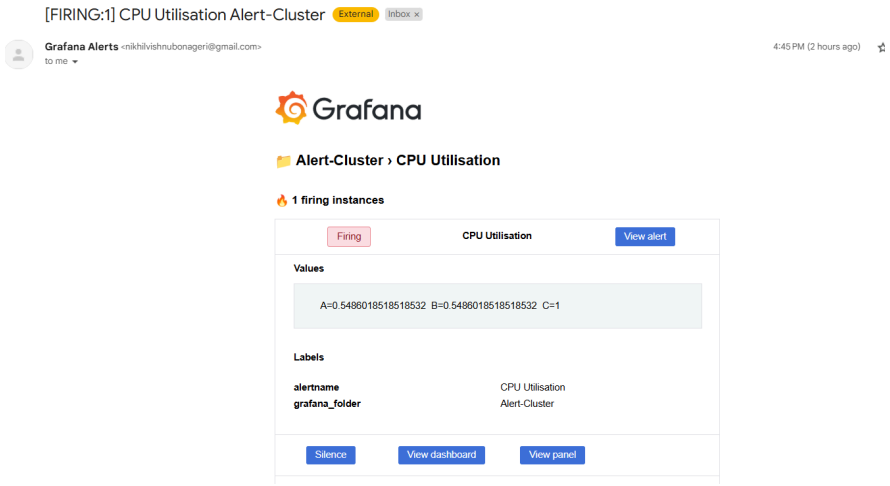
1. Alerts for CPU Utilization:

Metric: Cluster:cpu_usage:ratio

Threshold: Whenever the CPU Utilization is more than **80%**, we send alert notifications.

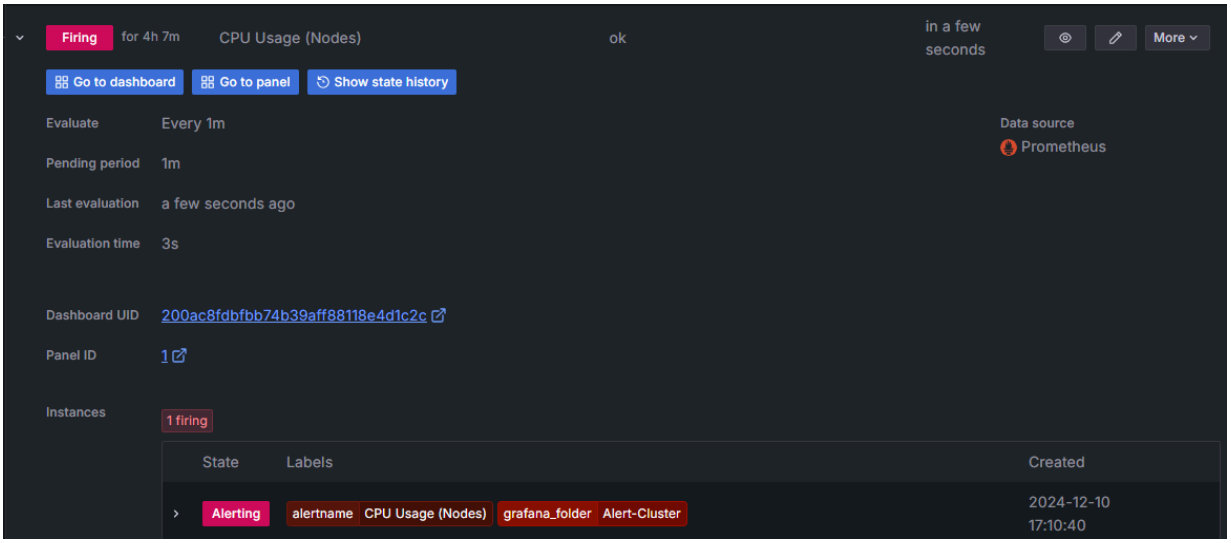


The Alert email that was sent to Gmail

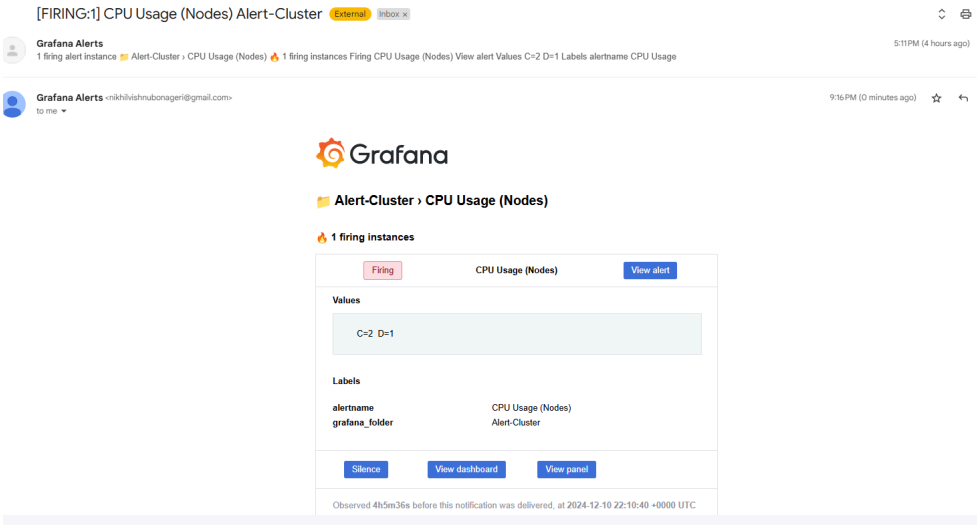


Here on the **Kubernetes / Compute Resources / Node (Pods) dashboard**, which is used for monitoring the compute resource usage (CPU and memory) of Kubernetes nodes and their associated pods.- we can now see that since the CPU usage is greater than the set threshold, they are showing the alerting symbol.

2. Alerts for CPU Usage (Nodes)
Threshold of greater than 80% would trigger an alert.



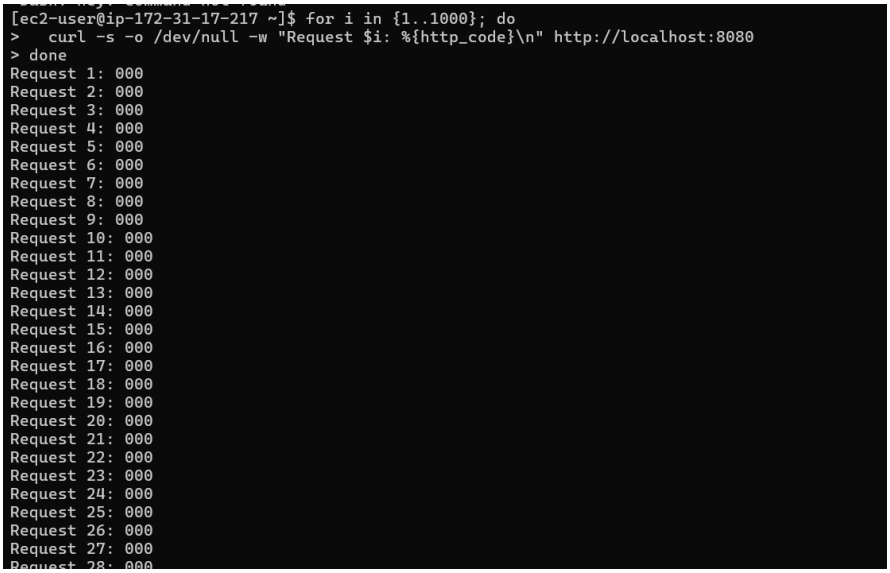
Email notification:



Part (V) Validate and Optimize Dashboards

We have validated that the dashboards reflect real-time data by simulating 1000 sequential requests to localhost using `curl` and observing the metrics being updated in Grafana. The dashboards displayed changes in request rate, response time, and other key performance indicators, confirming accurate real-time data visualization.

```
for i in {1..1000}; do
>   curl -s -o /dev/null -w "Request $i: %{http_code}\n" http://localhost:8080
> done
```



Challenges

1. **No Data Displayed on Dashboards**
   Initially, the dashboards did not show any data due to misconfigured data sources and Prometheus queries that did not align with the OpenTelemetry data being collected.
2. **Only 1/8 Prometheus Endpoints Were Up**
   - **Issue:** Out of the 8 scrape targets configured in Prometheus, only 1 endpoint was up and collecting data (8888).
   - **Cause:** The endpoints were not properly configured, leading to incomplete metrics collection from the EKS cluster.
   - **Impact:** This caused gaps in the data for CPU, memory, and other key metrics, making the dashboards ineffective for monitoring.
3. **Incomplete Metrics Collection**
   Metrics for certain nodes and pods were missing due to gaps in Prometheus scraping configurations, resulting in incomplete visualizations on the dashboards.

Solutions

1. **Fixing Data Source Configuration**
   - **Action Taken:** Verified and updated Grafana's data source to correctly point to the Prometheus endpoint.
   - **Outcome:** Accurate data began populating on the dashboards once the data source was configured correctly.
2. **Configuring Prometheus Endpoints**
   - **Action Taken:** Properly configured all 7 Prometheus scrape targets to ensure they were up and actively collecting data.
   - **Steps Taken:**
     - Verified the service discovery configuration in Prometheus.
     - Ensured that the endpoints were reachable and not blocked by network policies.
     - Restarted Prometheus to apply the changes.
   - **Outcome:** All 7 endpoints were brought up successfully, enabling comprehensive data collection across the cluster.
3. **Adjusting Scrape Configurations**
   - **Action Taken:** Reviewed and corrected the Prometheus scrape intervals and target configurations for nodes and pods.
   - **Outcome:** Metrics for CPU, memory, and other resources began appearing consistently on the dashboards.
4. **Aligning PromQL Queries**
   - **Action Taken:** Updated the PromQL queries to match the OpenTelemetry data schema.
   - **Outcome:** The panels displayed accurate metrics, resolving the issue of missing or incorrect data.

References

- **Deploying and Configuring Grafana in Kubernete**s:https://grafana.com/docs/grafana/latest/setup-grafana/installation/kubernetes/
- **Kubernetes Monitoring-** https://www.youtube.com/watch?v=EeiYpnBHnhY&list=PLdpzxOOAlwvJdsW6A0jCz_3VaANuFMLpc&index=15