Demo Project:
Configure Monitoring for Own Application

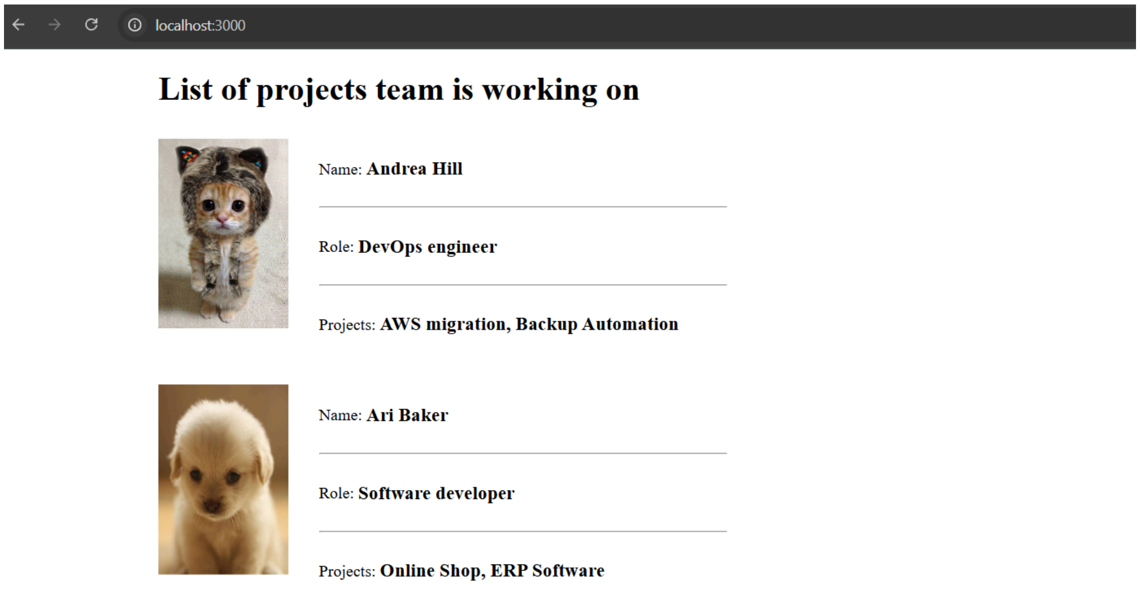
Project Description

In this project, we will:

- Configure our **NodeJS application** to collect & expose **metrics** with Prometheus Client Library.
- Deploy the **NodeJS application** into a **Kubernetes cluster**.
- Configure **Prometheus** to scrape these exposed metrics.
- Visualize the metrics in a **Grafana Dashboard**.

Step 1: Run NodeJS Application Locally

1. **Start the application:** `node app/server.js`
2. **Open your browser and visit:**
    - `http://localhost:3000` to see the application UI.



List of projects team is working on

Name: **Andrea Hill**

Role: **DevOps engineer**

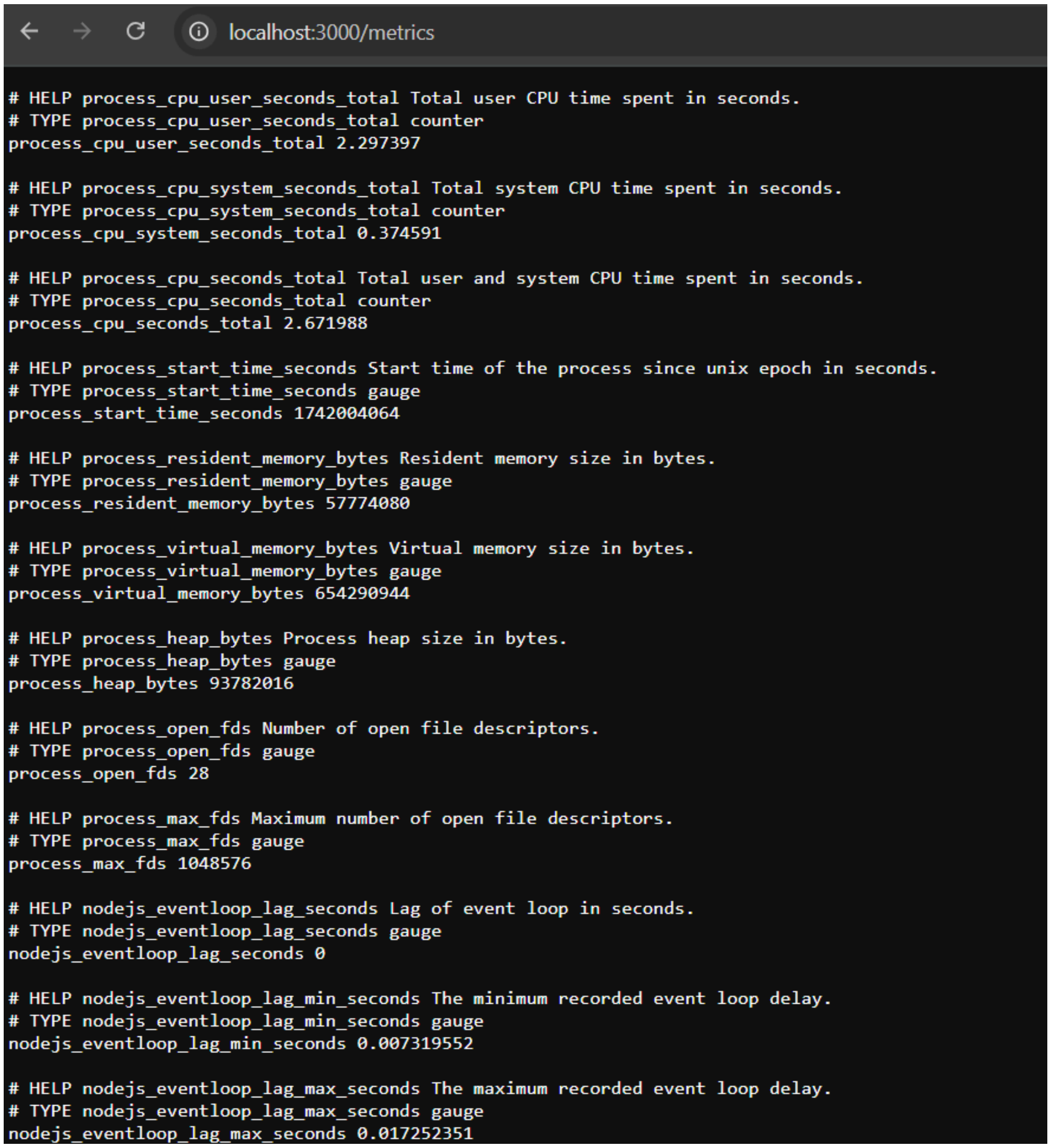
Projects: **AWS migration, Backup Automation**

Name: **Ari Baker**

Role: **Software developer**

Projects: **Online Shop, ERP Software**

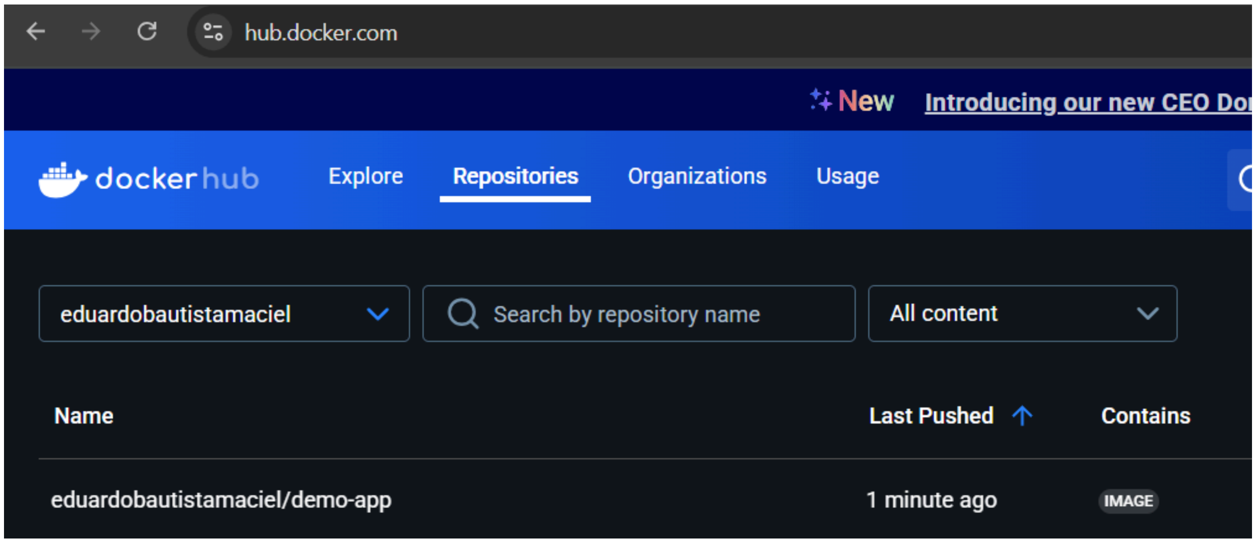
- `http://localhost:3000/metrics` to see Prometheus metrics.


localhost:3000/metrics

```
# HELP process_cpu_user_seconds_total Total user CPU time spent in seconds.
# TYPE process_cpu_user_seconds_total counter
process_cpu_user_seconds_total 2.297397

# HELP process_cpu_system_seconds_total Total system CPU time spent in seconds.
# TYPE process_cpu_system_seconds_total counter
process_cpu_system_seconds_total 0.374591

# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 2.671988

# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1742004064

# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 57774080

# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 654290944

# HELP process_heap_bytes Process heap size in bytes.
# TYPE process_heap_bytes gauge
process_heap_bytes 93782016

# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 28

# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1048576

# HELP nodejs_eventloop_lag_seconds Lag of event loop in seconds.
# TYPE nodejs_eventloop_lag_seconds gauge
nodejs_eventloop_lag_seconds 0

# HELP nodejs_eventloop_lag_min_seconds The minimum recorded event loop delay.
# TYPE nodejs_eventloop_lag_min_seconds gauge
nodejs_eventloop_lag_min_seconds 0.007319552

# HELP nodejs_eventloop_lag_max_seconds The maximum recorded event loop delay.
# TYPE nodejs_eventloop_lag_max_seconds gauge
nodejs_eventloop_lag_max_seconds 0.017252351
```

Step 2: Build Docker Image and Push to Repository

1. **Build the Docker image:** `docker build -t eduardobautistamaciel/demo-app:nodeapp .`
2. **Log in to Docker Hub:** `docker login -u eduardobautistamaciel`
3. **Push the image to Docker Hub:** `docker push eduardobautistamaciel/demo-app:nodeapp`



Name | Last Pushed | Contains
eduardobautistamaciel/demo-app | 1 minute ago | IMAGE

Step 3: Deploy App into K8s cluster

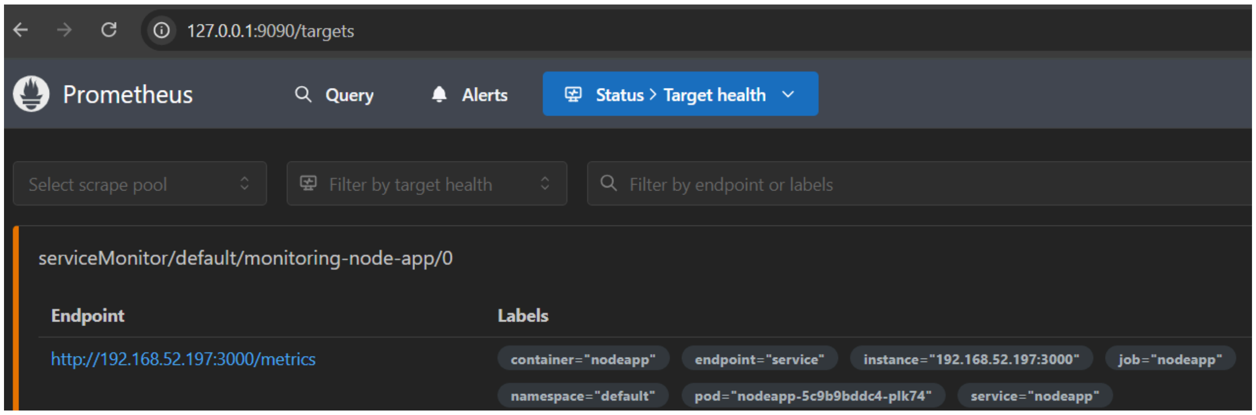
1. **Create** `k8s-config.yaml`:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp
  labels:
    app: nodeapp
spec:
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      labels:
        app: nodeapp
    spec:
      imagePullSecrets:
      - name: my-registry-key
      containers:
      - name: nodeapp
        image: nanajanashia/demo-app:nodeapp
        ports:
        - containerPort: 3000
        imagePullPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: nodeapp
  labels:
    app: nodeapp
spec:
  type: ClusterIP
  selector:
    app: nodeapp
  ports:
  - name: service
    protocol: TCP
    port: 3000
    targetPort: 3000
---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: monitoring-node-app
  labels:
    release: monitoring
    app: nodeapp
spec:
  endpoints:
  - path: /metrics
    port: service
    targetPort: 3000
  namespaceSelector:
    matchNames:
    - default
  selector:
    matchLabels:
      app: nodeapp
```

2. **Create Secret for Docker Registry Access:**

`kubectl create secret docker-registry my-registry-key --docker-server=https://index.docker.io/v1/ --docker-username=eduardobautistamaciel --docker-password=xxxxxxxx`

3. **Deploy the application:**
   - `kubectl apply -f k8s-config.yaml`
4. **Verify deployment:**
   - `kubectl get pod`
   - `kubectl get svc`
   - `kubectl port-forward svc/nodeapp 3000:3000`
5. **Verify ServiceMonitor in Prometheus UI**
   - Navigate to **Status → Targets** to see the **ServiceMonitor** entry for `nodeapp`.



127.0.0.1:9090/targets

Prometheus   Query   Alerts   Status > Target health

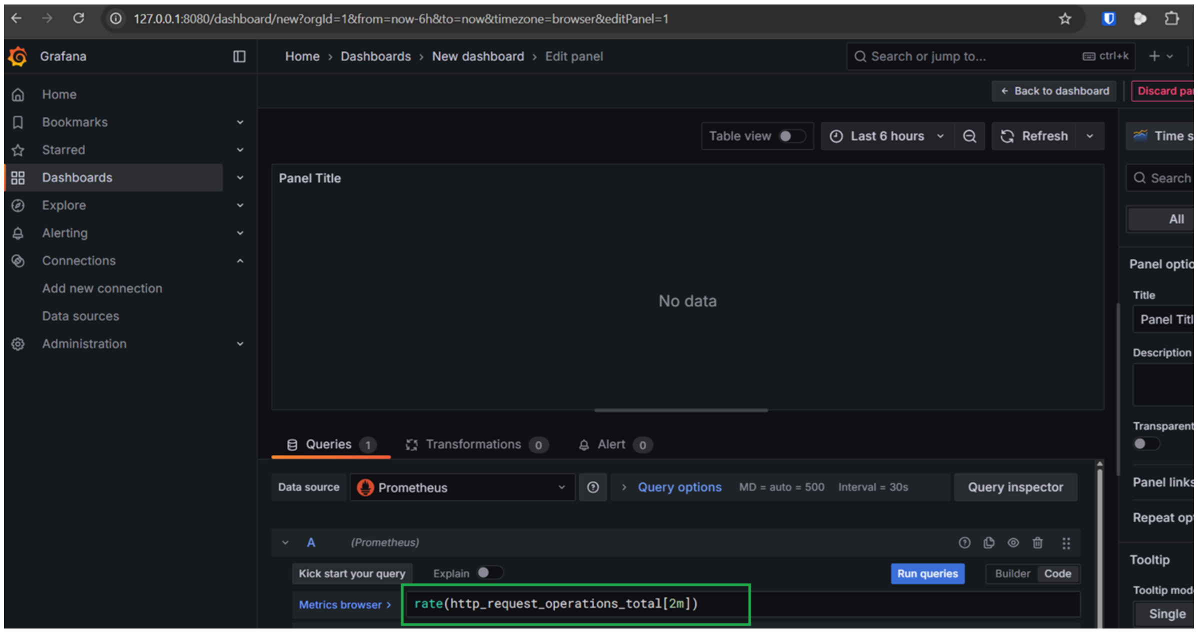
Select scrape pool   Filter by target health   Filter by endpoint or labels

serviceMonitor/default/monitoring-node-app/0

Endpoint	Labels
http://192.168.52.197:3000/metrics	container="nodeapp" endpoint="service" instance="192.168.52.197:3000" job="nodeapp" namespace="default" pod="nodeapp-5c9b9bddc4-plk74" service="nodeapp"

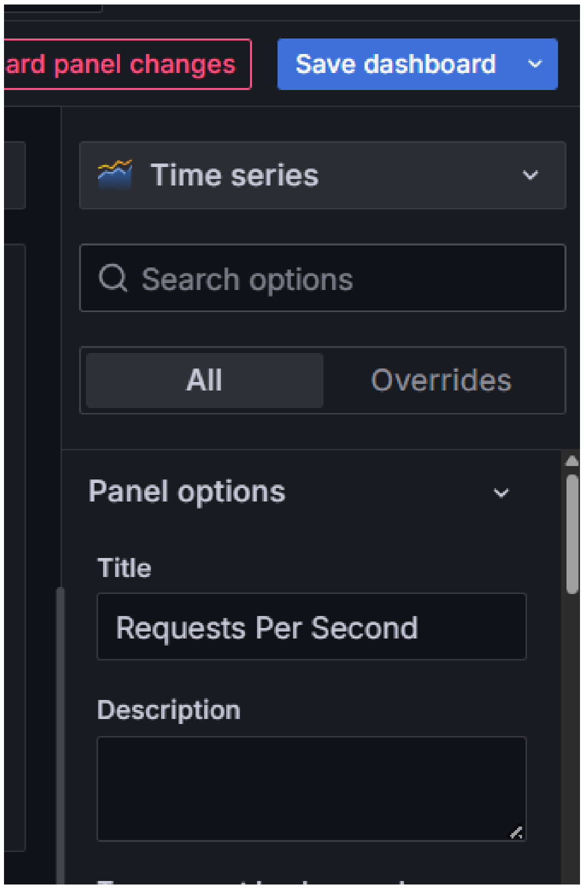
Step 4: Create Grafana Dashboard

1. In **Grafana**, click **New → New Dashboard → Add Visualization**.
2. **Add the following query:** `rate(http_request_operations_total[2m])`



```
rate(http_request_operations_total[2m])
```

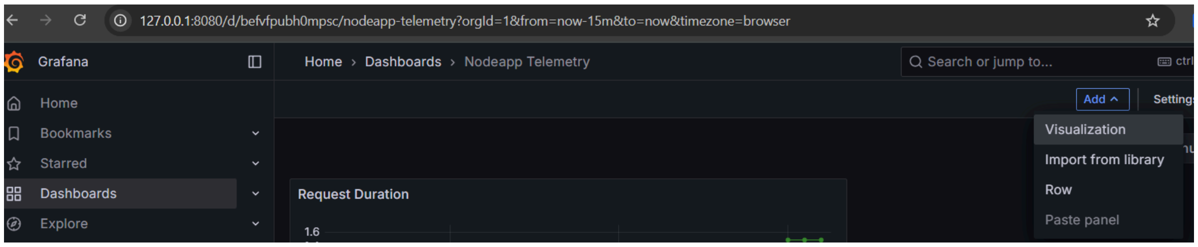
3. Click **Apply** and name the panel **"Nodeapp Telemetry"** and Title **“Requests Per Second”:**



Title: Requests Per Second

Description:

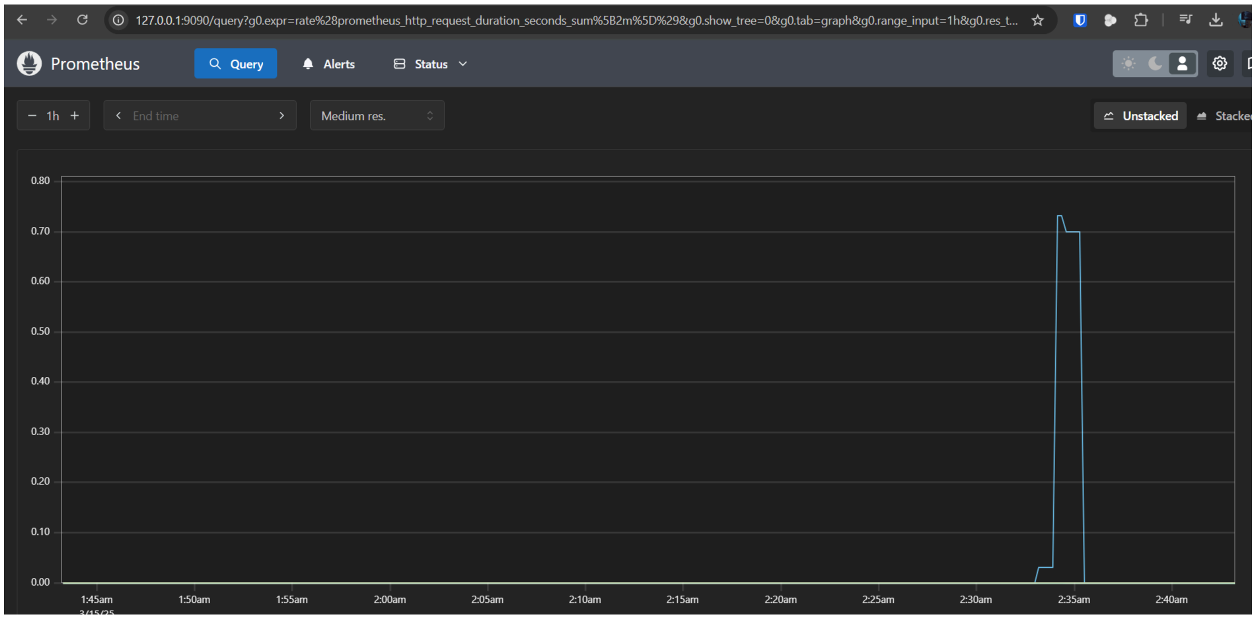
4. **Add another panel** with: `rate(prometheus_http_request_duration_seconds_sum[2m])` and Title "Request Duration"

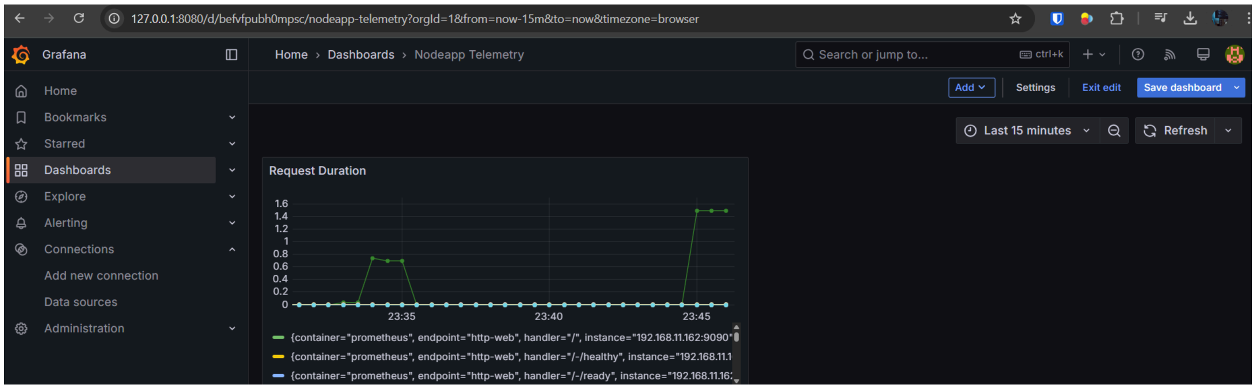

127.0.0.1:8080/d/befvfpubh0mpsc/nodeapp-telemetry?orgId=1&from=now-15m&to=now&timezone=browser

5. **Save the dashboard** as **"Nodeapp Metrics"**.

Step 5: Verify Metrics in Prometheus

1. Open **Prometheus UI**.
2. **View the real-time telemetry data** for `nodeapp`.




127.0.0.1:8080/d/befvfpubh0mpsc/nodeapp-telemetry?orgId=1&from=now-15m&to=now&timezone=browser
Grafana
Home
Dashboards
Nodeapp Telemetry
Search or jump to...
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Add
Settings
Exit edit
Save dashboard
Last 15 minutes
Refresh
Request Duration
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
23:35
23:40
23:45
{container="prometheus", endpoint="http-web", handler="/", instance="192.168.11.162:9090"
{container="prometheus", endpoint="http-web", handler="/-/healthy", instance="192.168.11.1
{container="prometheus", endpoint="http-web", handler="/-/ready", instance="192.168.11.16

Step 6: Cleanup

1. **Delete all the Kubernetes resources** (Deployment, Service, ServiceMonitor) specified in the: `kubectl delete -f k8s-config.yaml`
2. **Check EKS Cluster Name:** `eksctl get cluster`

   Example output:

```
NAME                           REGION       EKSCTL CREATED
extravagant-gopher-1741974792  us-east-1    True
```

2. **Delete EKS Cluster:**

   `eksctl delete cluster --name extravagant-gopher-1741974792`
