# Documentation of Step 1: DevOps Challenge – Running Application on EC2 and EKS Kubernetes Cluster

**Overview:**

This document outlines the implementation of **Step 1** of the DevOps Engineer test assignment. The objective was to design and deploy a web server application using a language of choice and host it on both an **EC2 instance** and an **EKS (Elastic Kubernetes Service)** cluster on AWS.

As per the assignment, the deployed application must:

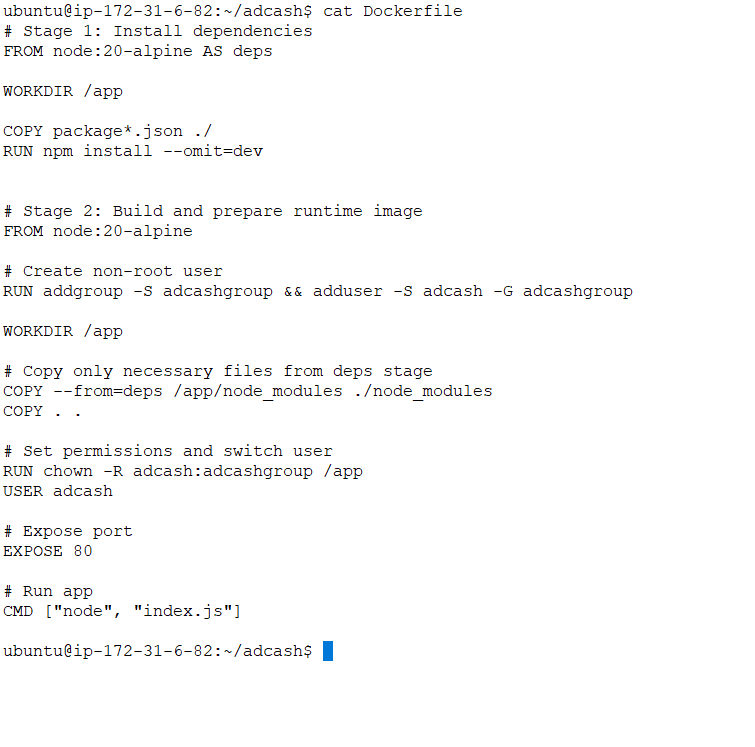
* Serve **Gandalf’s image** at the /gandalf endpoint.
* Display the **current time in Colombo, Sri Lanka** at the /colombo endpoint.
* Run on a **static IP**, **Loadbalancer** with **only port 80** exposed to the internet.
* Include a **built-in Prometheus exporter** that provides two metrics:
  + Total number of requests to /gandalf
  + Total number of requests to /colombo

This document covers the architecture, deployment process, and configurations used to successfully meet these requirements across both EC2 and EKS environments.

**Note:**  
Since the worker nodes in my EKS cluster are deployed within private subnets, I exposed the application using an AWS Elastic Load Balancer by configuring the Kubernetes service as a Load Balancer instead of a static IP on port 80. This approach ensures secure external access without exposing the worker nodes directly to the internet, which is a recommended best practice for security.  
  
Refer to the screenshots below for implementation details.

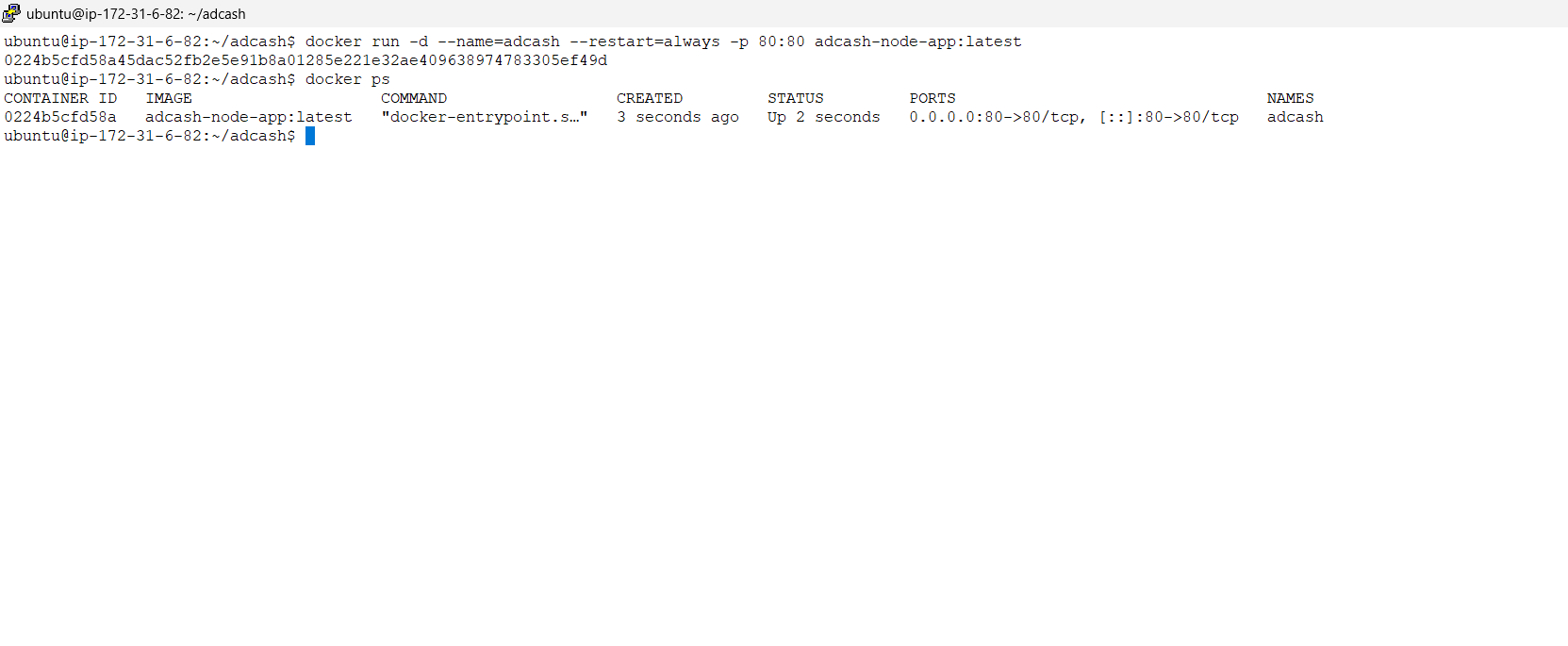
**Application Overview**

The Node.js application is containerized using Docker and utilizes a multi-stage build process to ensure a clean and optimized image. It runs on **Node.js v20** with **Alpine Linux** as the base image for a minimal footprint and enhanced performance.

**DockerFile:**  
  


**Key Features of the Dockerfile**

* **Multi-stage Build**: Streamlines dependency installation to produce a clean final image.
* **Non-root Execution**: Implements a dedicated non-root user (adcash) to enhance container security.
* **Lightweight Base Image**: Uses node:20-alpine for a smaller image size.
* **Port Exposure**: Exposes port 80 for external access.



**Building the Docker Image**

To build the Docker image, execute the following command:

**docker build -t adcash-node-app .**

This command builds the Docker image with the name **adcash-node-app** and assigns the **latest** tag by default.

**Running the Docker Container**

To run the container in detached mode with automatic restart enabled, use:

**docker run -d --name=adcash --restart=always -p 80:80 adcash-node-app:latest**

This will start the application with:

* Container name: adcash
* Port mapping: Host port 80 to container port 80

**Accessing the Application**

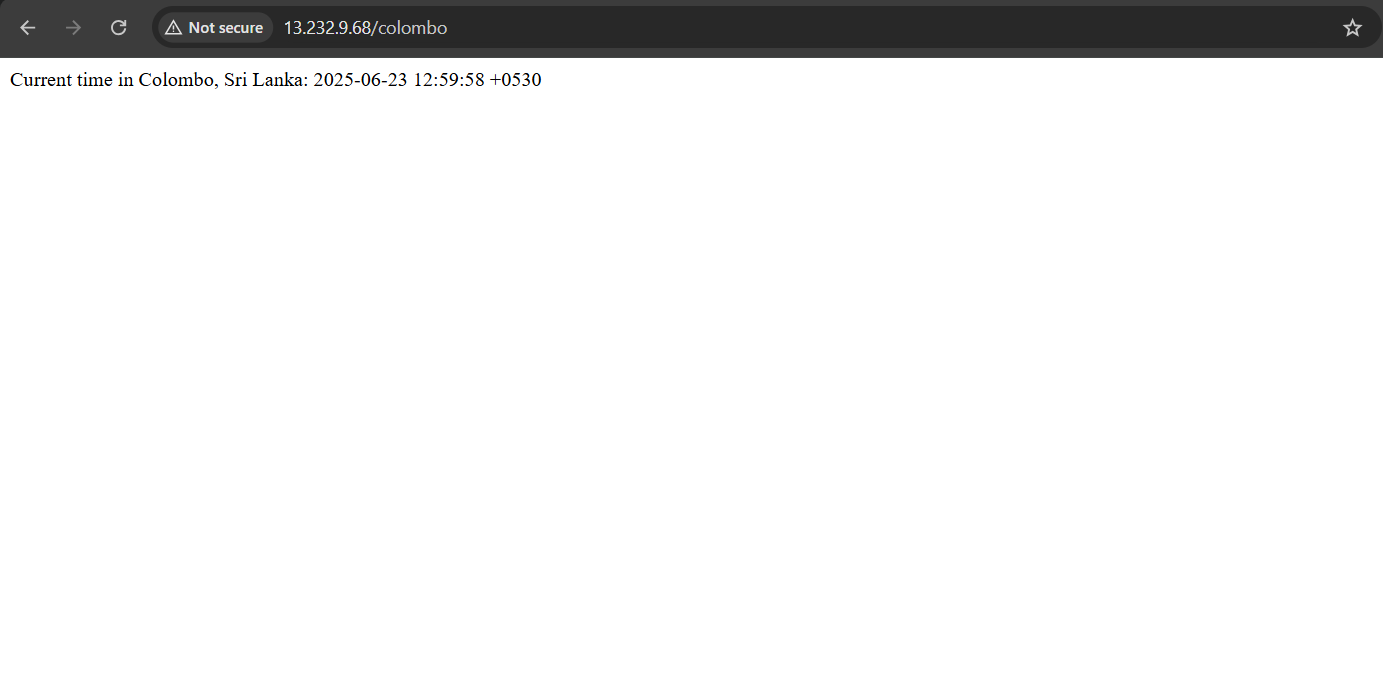
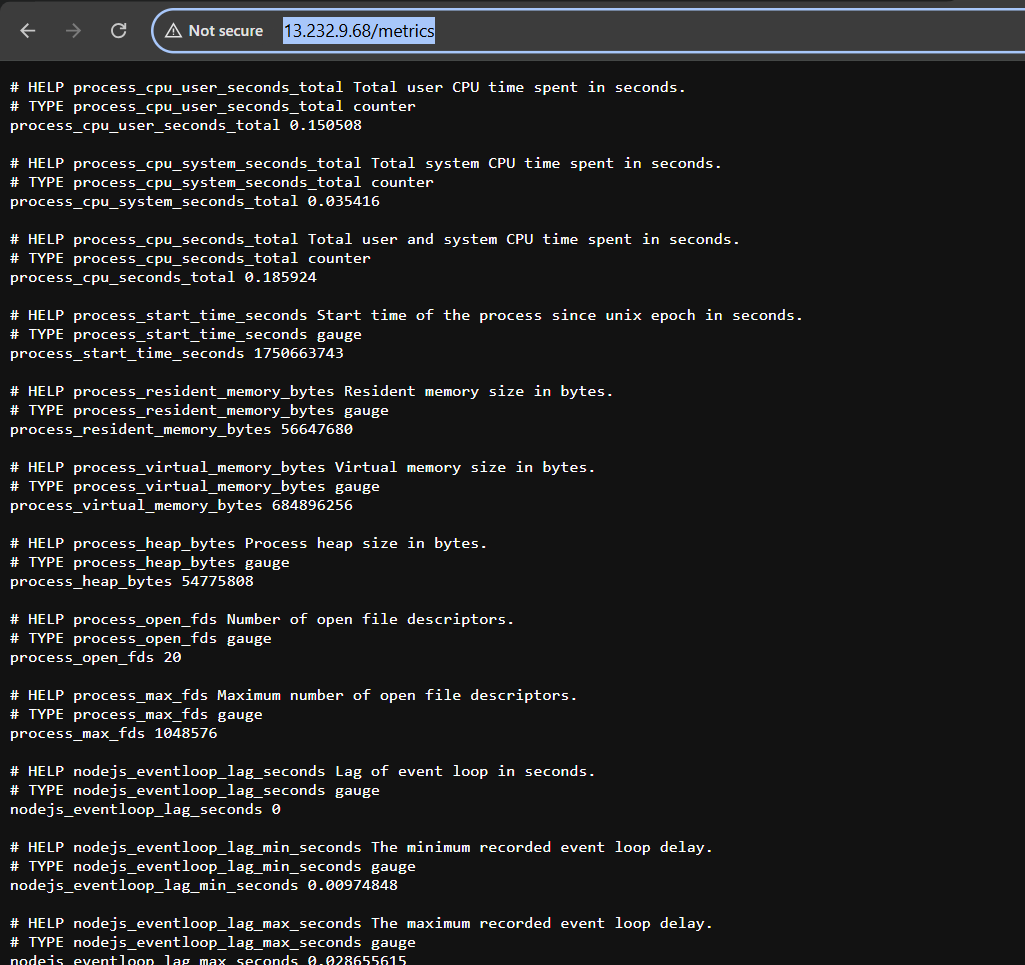
Once the container is running, the application can be accessed through a web browser using the following URLs:

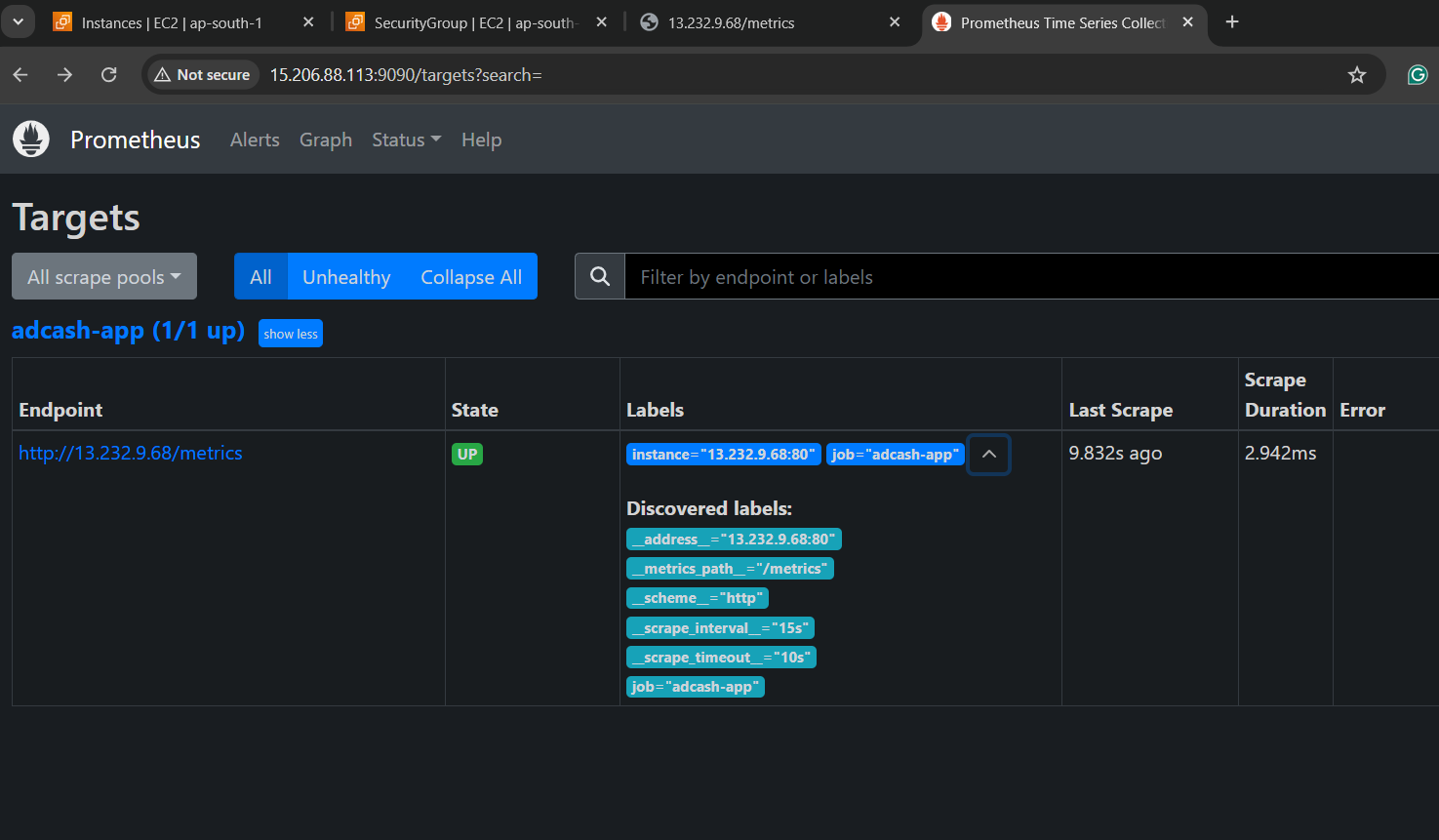
<http://13.232.9.68/gandalf>

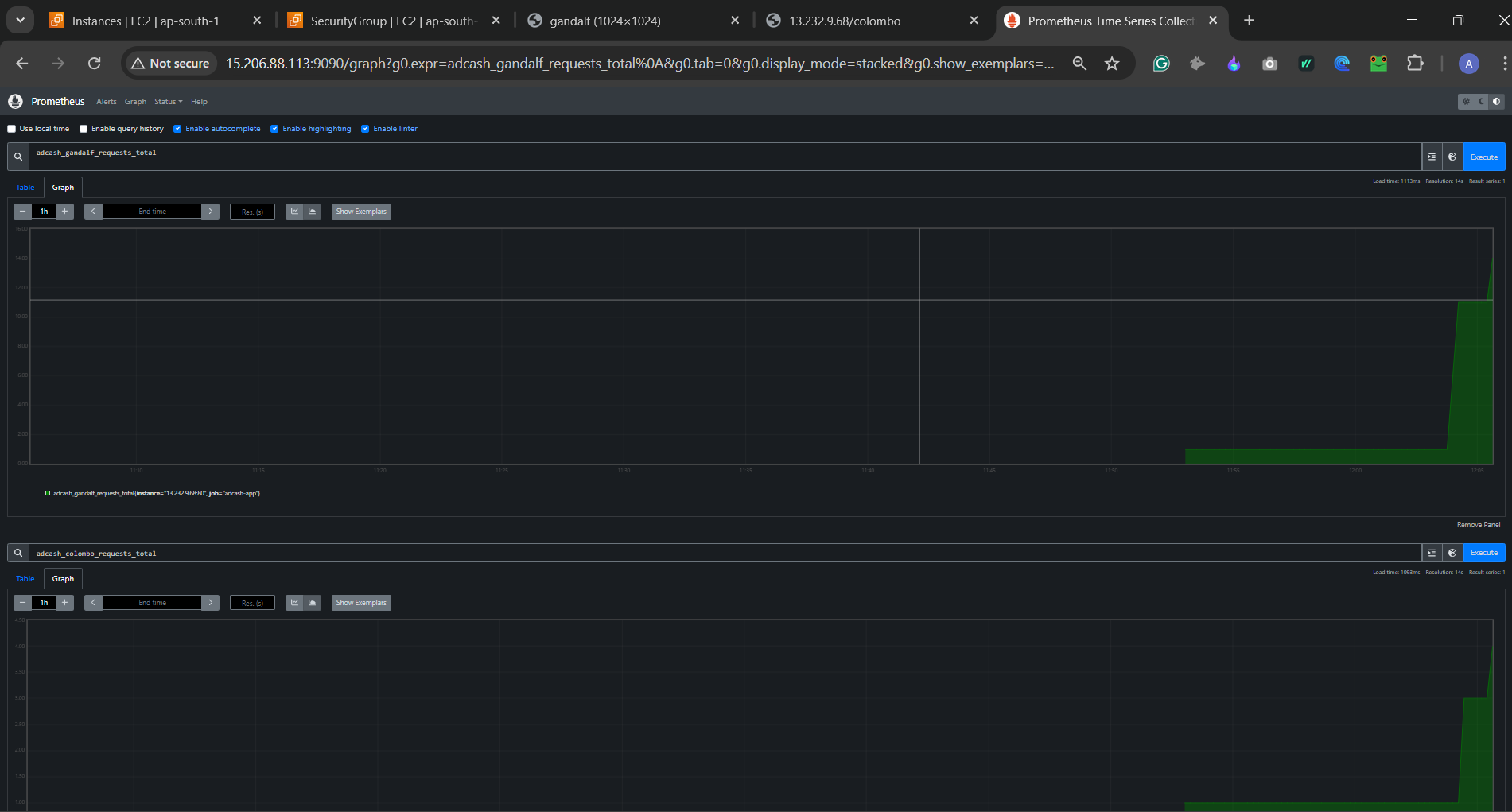
[http://13.232.9.68/colombo](http://13.203.66.119/colombo)

[http://13.232.9.68/metrics](http://13.203.66.119/metrics)



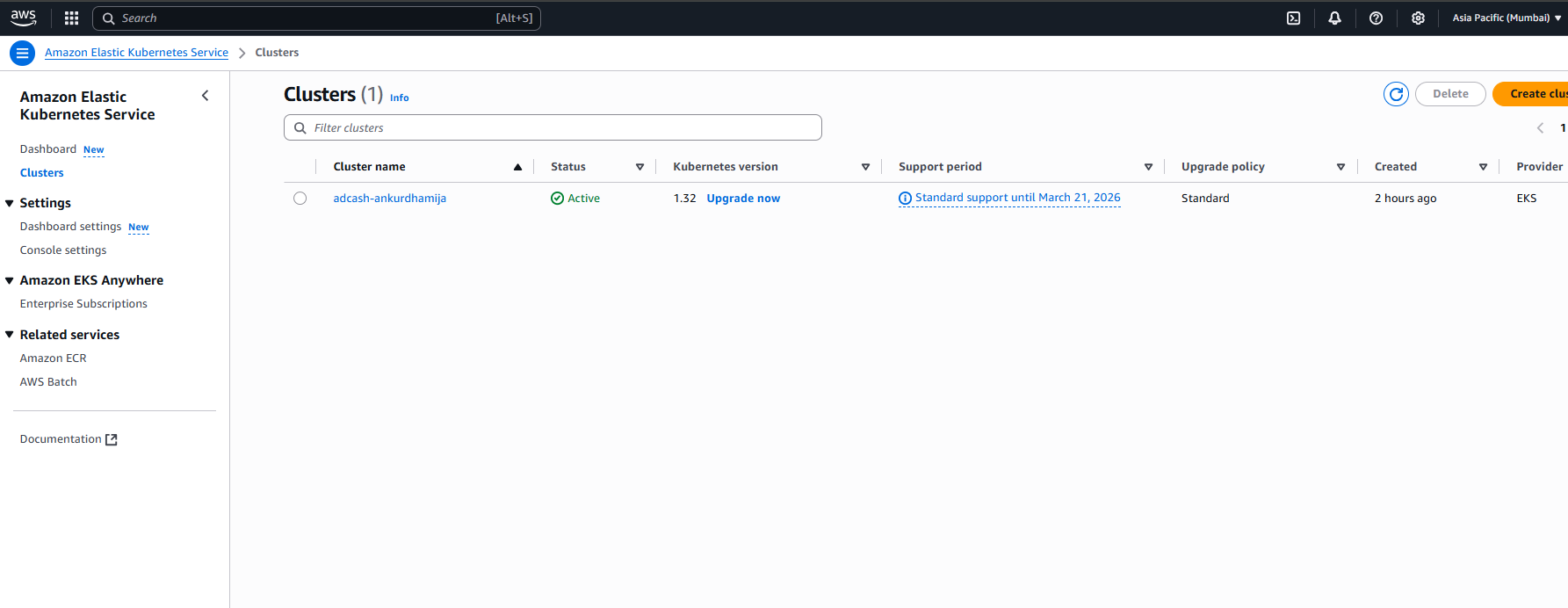
  
  
**Note: The screenshot below displays the total number of requests received by the /colombo and /gandalf endpoints as recorded by the Prometheus server.**



**# Step 1 (optional) Running the adcash Node.js Application in a EKS Kubernetes Cluster.**

These processes provide a comprehensive overview of the Kubernetes deployment for the **adcash** Node.js application running on our EKS cluster, **adcash-ankurdhamija**.

**Description:**

The deployment setup includes Kubernetes manifests for deploying and exposing the Adcash application, along with Prometheus annotations for monitoring.  
  


**Key Components**

* **Deployment**: Manages the lifecycle of application pods.
* **Service (Load Balancer)**: Exposes the application to external traffic.
* **Prometheus Annotations**: Facilitates monitoring through metrics scraping.

**Deployment Configuration**

* **Kind**: Deployment
* **Name**: adcash-deployment
* **Replicas**: 1
* **Container Image**: 761018875260.dkr.ecr.ap-south-1.amazonaws.com/adcash:latest
* **Container Port**: 8080

**Resource Allocation:**

* **Requests**:
  + CPU: 100m
  + Memory: 128Mi
* **Limits**:
  + CPU: 200m
  + Memory: 256Mi

**Service Configuration**

* **Kind**: Service
* **Type**: LoadBalancer
* **Name**: adcash-service
* **Ports**:
  + **Port**: 80
  + **TargetPort**: 8080

**Monitoring Integration (Prometheus):**

The following annotations are included to enable Prometheus scraping for metrics:

prometheus.io/scrape: "true"

prometheus.io/path: "/metrics"

prometheus.io/port: "8080"

**Commands:**

To apply the configurations to the cluster, use the following commands:

**kubectl apply -f deployment.yaml**

**kubectl apply -f service.yaml**

