

# **LIBRARY MANAGEMENT SYSTEM**

**Team 5**

**Ahmed Yehia (Scrum Master)**

**Marwan Mohammed**

**Sama Hatem**

**Safia Adel**

**Banque Misr internship 2024 Graduation Project  
DevOps Track**

**Egypt, September 2024**

# Contents

Introduction .....	4
Abstract .....	5
Part 1: Application Development .....	6
Tools Used: .....	6
User Manual .....	7
Login Page .....	7
Sign-up Page .....	8
Admin User Page .....	9
Part 2: Dockerization: .....	13
Part 3: Infrastructure as Code with Terraform .....	14
<b>Modules details</b> .....	15
1. VPC Configuration .....	15
2. Subnet Configuration .....	15
3. Internet Gateway .....	16
4. NAT Gateway .....	16
5. Route Tables .....	17
6. IAM Roles for EKS .....	18
7. EKS Cluster and Node Group .....	19
Part 4: Kubernetes Deployment on EKS: .....	20
1-library-management-deployment.yaml file: .....	20
2-service.yaml file: .....	21
Deployment Configuration (library-management-deployment.yaml) .....	22
Part 5: CI/CD Pipeline Setup: .....	23
Pipeline Stages: .....	23
1. Checkout Code .....	23
2- Build Docker Image: .....	23
3- Push Docker Image: .....	24
4- Deploy to EKS: .....	24
5-Get the Load Balancer IP: .....	25
Bonus Task: Set Up Monitoring and Logging: .....	26
<b>Setup Prometheus</b> .....	27
Prometheus ConfigMap ( .....	27
Prometheus Deployment .....	27
Prometheus Service (prometheus-service.yaml) .....	28
Grafana Setup .....	29
1- <b>Grafana Deployment</b> .....	29
2- <b>Grafana Service with LoadBalancer</b> .....	30
Integrating Prometheus with Grafana .....	31
Access Grafana .....	31
Add Prometheus as a Data Source .....	31
Create Dashboards .....	31
<b>Terraform Pipeline (Bonus task)</b> .....	33
<b>Terraform Infrastructure Setup</b> .....	33
Pipeline Stages .....	33
1. Checkout SCM .....	33
2. Setup Plugin Cache Directory .....	34
3. Terraform Init - Backend .....	34
4. Terraform Apply - Backend .....	34
5. Terraform Init - Main Creation .....	34
6. Terraform Apply - Main Creation .....	35

# Table of Figures

Figure 1 Login Page .....	7
Figure 2 Wrong Password Trigger .....	7
Figure 3 Sign Up Page.....	8
Figure 4 Admin Page .....	9
Figure 5 Adding book to the inventory .....	10
Figure 6 book added successfully to inventory.....	10
Figure 7 Keyword Searching Result.....	10
Figure 8 Searching Result applying ISBN filter .....	11
Figure 9 Adding admin to the system database .....	11
Figure 10 Logout Verification.....	11
Figure 11 Normal User Page .....	12
Figure 12 Search for a borrowed book .....	12
Figure 13 Docker image on dockerhub .....	13
Figure 14 Terraform Modules .....	14
Figure 15 Deployment Yaml File .....	20
Figure 16 Service Yaml File.....	21
Figure 17 Pipeline 1. Checkout Code .....	23
Figure 18 Pipeline Build Docker Image .....	23
Figure 19 pipeline Push Docker Image:.....	24
Figure 20 Pipeline Deploy to EKS .....	24
Figure 21 Pipeline Loadbalancer .....	25
Figure 22 configuration to the Prometheus deployment. ....	27
Figure 23 Prometheus deployment yaml file.....	27
Figure 24 Prometheus service File .....	28
Figure 25 Grafana Deployment yaml file .....	29
Figure 26 Grafana Service Loadbalancer Yaml File .....	30
Figure 27 Prometheus Chart .....	31
Figure 28 Prometheus Target .....	31
Figure 29 Number of http Request .....	32
Figure 30 Process CPU second .....	32

# Introduction

The **Library Management System** developed by Team 5 during the Banque Misr Internship 2024 aims to streamline library operations by offering an intuitive platform for both administrators and users. The project encapsulates the modern software development lifecycle by integrating core software engineering principles with DevOps practices. From backend application development to containerization, infrastructure management using Infrastructure as Code (IaC), and automated deployments via CI/CD pipelines, this project demonstrates a full stack of contemporary software tools and methodologies. This comprehensive system is designed to support both day-to-day operations of libraries and allow users to conveniently manage their borrowing and return processes.

This project also showcases the team's hands-on experience with cloud services like AWS (Amazon Web Services) and Kubernetes, ensuring scalability, resilience, and security for the application. By leveraging Amazon EKS (Elastic Kubernetes Service), the project enables efficient resource management while maintaining high availability. Additionally, continuous monitoring and logging were implemented using Prometheus and Grafana, facilitating real-time performance tracking and issue resolution.

The following document delves into the system's architecture, deployment process, and technologies used, outlining the steps taken to create a robust, scalable, and user-friendly library management system.

# Abstract

This document outlines the development and deployment of a cloud-native **Library Management System**, created as part of the Banque Misr Internship 2024. The project is built using Python Flask for the backend, HTML/CSS/JavaScript for the frontend, and managed via Docker containers. The entire infrastructure is deployed on Amazon Web Services (AWS) using Elastic Kubernetes Service (EKS) for container orchestration. Infrastructure provisioning was automated through Terraform, a widely used Infrastructure as Code (IaC) tool, ensuring the scalability and maintainability of the system.

A key feature of the project is the CI/CD pipeline designed with Jenkins, which automates the building, testing, and deployment of the application. Prometheus and Grafana were also integrated for monitoring and visualization of application metrics. The system supports both administrators, who can manage the book inventory and user roles, and users, who can search, borrow, and return books. The project delivers a scalable, efficient, and user-friendly solution for managing library resources.

# Part 1: Application Development

A simple Library Management website was developed with (CSS, HTML, and JavaScript) for the front end and Python Flask for handling the back end. Here is the breakdown of the application features:

1. Users for this application are either admins or users.
2. The application includes basic routes to get a list of books, retrieve a specific book, add a new book to the library(admin), and borrow and return a book (User).
3. Admin Users can add other admins.
4. Both admins and users can search for books either with the ISBN or with a keyword that is the title, author, or ISBN.

## Tools Used:

### 1- Frontend

- **JavaScript:** Used for creating interactive client-side functionality
- **HTML:** Used for structuring and organizing content on the web page
- **CSS:** Used for styling and layout of the web page

### 2- Backend

- **Python:** Used as the programming language for the backend
- **Flask:** Used as the web framework for building the backend API

### 3- Database

- **JSON files:** Used as the database storage system, where data is stored in JSON format

# User Manual:

## Login Page

- The user is presented with a login page where they can enter their:
  - **Username**
  - **Password**
- Upon successful login, the system checks the database to determine the user's role.

## User Roles

- There are two types of users in the system:
  - **Normal User:** A regular user with limited access and privileges.
  - **Admin User:** An administrator with elevated access and privileges.

## Post-Login Redirection

- Once the user's role is determined, the system redirects them to their respective user page:
  - **Normal User:** Redirected to the normal user page with limited features and access.
  - **Admin User:** Redirected to the admin user page with elevated features and access.

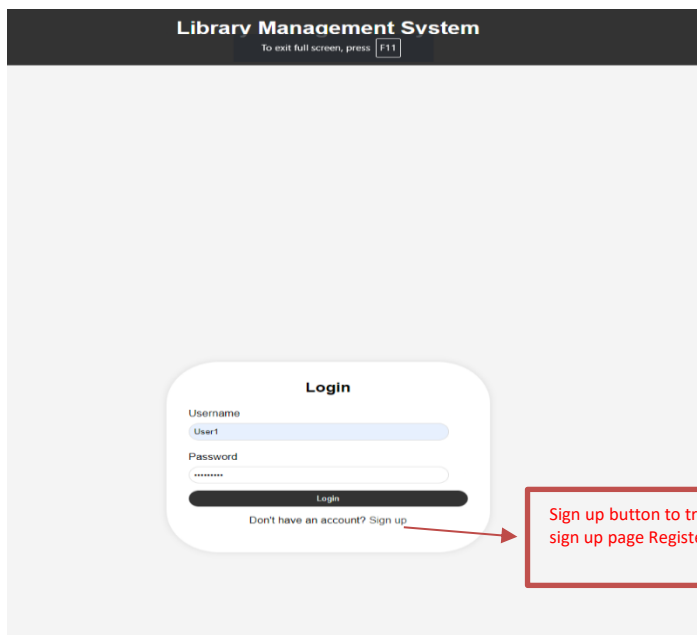


Figure 1 Login Page

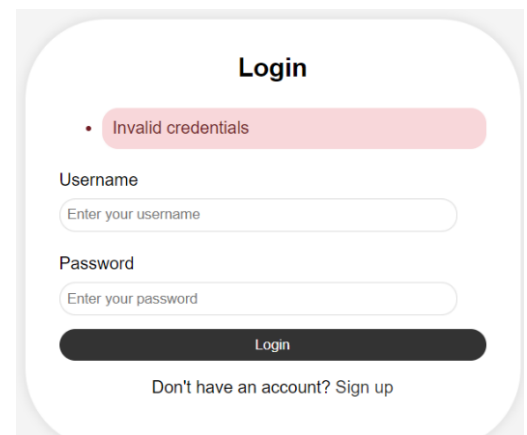


Figure 2 Wrong Password Trigger

## Sign-up Page

- The user is presented with a sign-up page where they can enter their:
  - Username
  - Preferred password
  - Email address
- Upon successful sign-up, the system creates a new account for the user as a **Normal User**.

### Default User Role

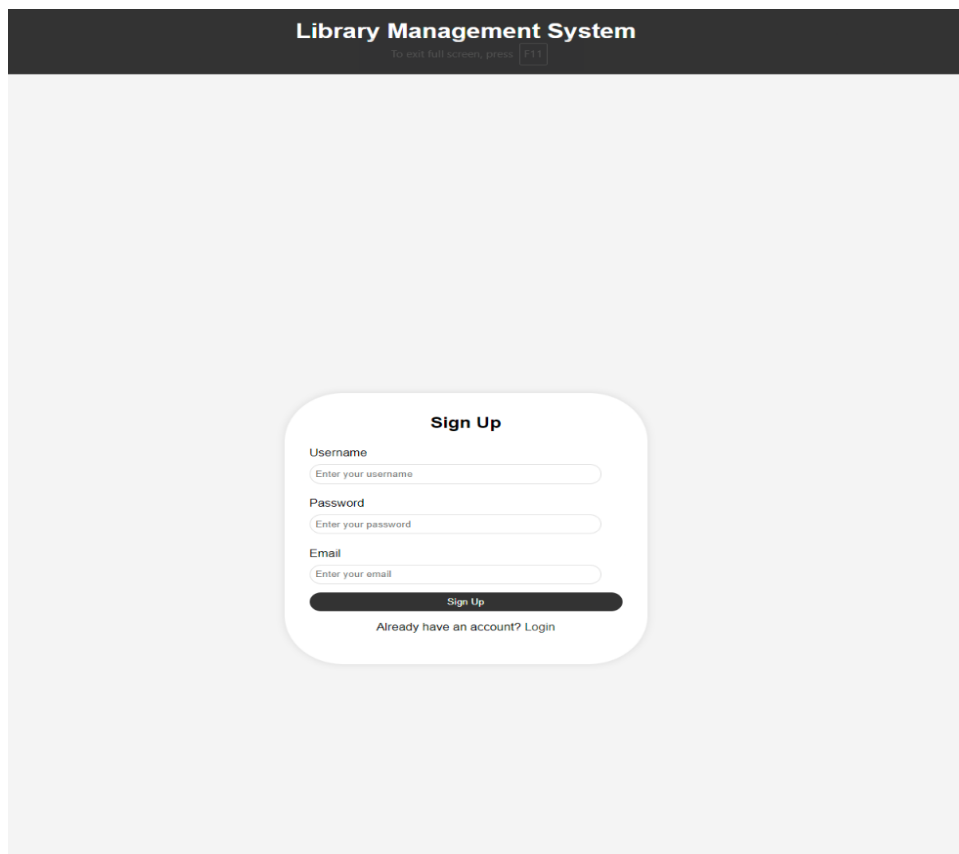
- All newly created users are assigned the role of **Normal User** by default.

### Immediate Access

- The new user can access the system immediately through the **Normal User Page** with limited features and access.

### Admin Promotion

- Normal users can be promoted to **Admin Users** by **existing admins** in the system.
- The promotion process is done through other admins in the system, granting the promoted user access to the **Admin Page** and elevated privileges.



The screenshot displays the 'Sign Up' page of a 'Library Management System'. The page has a dark header with the title 'Library Management System' and a subtitle 'To exit full screen, press [F11]'. The main content area is light gray and features a white, rounded rectangular sign-up form. The form includes three input fields labeled 'Username', 'Password', and 'Email', each with a placeholder text 'Enter your username', 'Enter your password', and 'Enter your email' respectively. Below these fields is a dark 'Sign Up' button. At the bottom of the form, there is a link that says 'Already have an account? Login'.

Figure 3 Sign Up Page



## Admin User Page

### Admin User Functionalities

- Manage books: add, search, view inventory, and remove
  - Manage admins: add new admins to the system
- The Admin User Page provides several functionalities to manage the system, including:

### Book Management

#### 1- Add Book

- The admin can add a new book to the system by providing the following details:
- **Book title, Author, Genre, Year**
- Note: The book is assigned automatically from the system to an ISBN that auto increments to each book**

#### 2- Search Book

- The admin can search for books using two filtration methods:
1. **ISBN:** Search by International Standard Book Number (ISBN)
  2. **Keywords:** Search by author name or book title, with the ability to detect parts of the name and search for the rest

#### 3- View Book Inventory

- The admin can view the current book inventory in the system.

#### 4- Remove Book

- The admin has the access to remove a book from the system's inventory.

### Admin Management

#### Add New Admin

- The admin can add a new admin to the system by entering the new admin's **Username & Password**
- The new admin will be added to the admins database.

Book Title	Author	ISBN	Genre	Date of Publish	Status	Actions
Sherlock Holmes	Arthur Conan Doyle	1	Mystery	1892	Available	<button>Remove</button>
Death Joke	Agatha Christie	2	Mystery	1930	Available	<button>Remove</button>
Hands on Machine Learning	Aurelien Geron	3	Science	2022	Available	<button>Remove</button>
80 Days around the World	Charles Dickens	4	Fantasy	1873	Available	<button>Remove</button>
Big Brother	Alfred Yehia	5	Mystery	12/1/1999	Available	<button>Remove</button>
Big Brother 2	Alfred Yehia	6	Fantasy	1999	Available	<button>Remove</button>
Book4	Alfred Yehia	7	Fantasy	1999	Available	<button>Remove</button>

Figure 4 Admin Page

## Adding a book to the inventory

### Add Book

Select Genre

Select Genre

Fantasy

Mystery

Horror

Historical Fiction

Romance

Biography

History

True Crime

Science

Other

Add Book

### Add Book

Book name

Book author

Book Genre

Release Year

Add Book

Figure 5 Adding book to the inventory

Book Title	Author	ISBN	Genre	Date of Publish	Status	Actions
Sherlock Holmes	Arthur Conan Doyle	1	Mystery	1892	Available	<button>Remove</button>
Death Joke	Agatha Christie	2	Mystery	1930	Available	<button>Remove</button>
Hands on Machine Learning	Aurelien Geron	3	Science	2022	Available	<button>Remove</button>
80 Days around the World	Charles Dickens	4	Fantasy	1873	Available	<button>Remove</button>
Big Brother	Ahmed Yehia	5	Mystery	12/7/1899	Available	<button>Remove</button>
Big Brother 2	Ahmed Yehia	6	Fantasy	1899	Available	<button>Remove</button>
Book4	Ahme	7	Fantasy	1899	Available	<button>Remove</button>
DevOps Graduation Project	Team 5	8	True Crime	2024	Available	<button>Remove</button>

Figure 6 book added successfully to inventory

### Search by Keyword

Search

### Book Inventory

Book Title	Author	ISBN	Genre	Date of Publish	Status	Actions
Big Brother	Ahmed Yehia	5	Mystery	12/7/1899	Available	<button>Remove</button>
Big Brother 2	Ahmed Yehia	6	Fantasy	1899	Available	<button>Remove</button>

Figure 7 Keyword Searching Result

### Search by ISBN

5

Search

### Search by Keyword

big

Search

### Book Inventory

Book Title	Author	ISBN	Genre	Date of Publish	Status	Actions
Big Brother	Ahmed Yehia	5	Mystery	12/7/1899	Available	<button>Remove</button>

Figure 8 Searching Result applying ISBN filter

### Add new Admin

SamaHatem

.....

Add Admin

```
main.tf ...\grafana U  {} users.json  variables.tf ...\grafana U  
}} "SamaHatem": {"password": "Admin1234", "role": "Admin"}}]
```

Figure 9 Adding admin to the system database

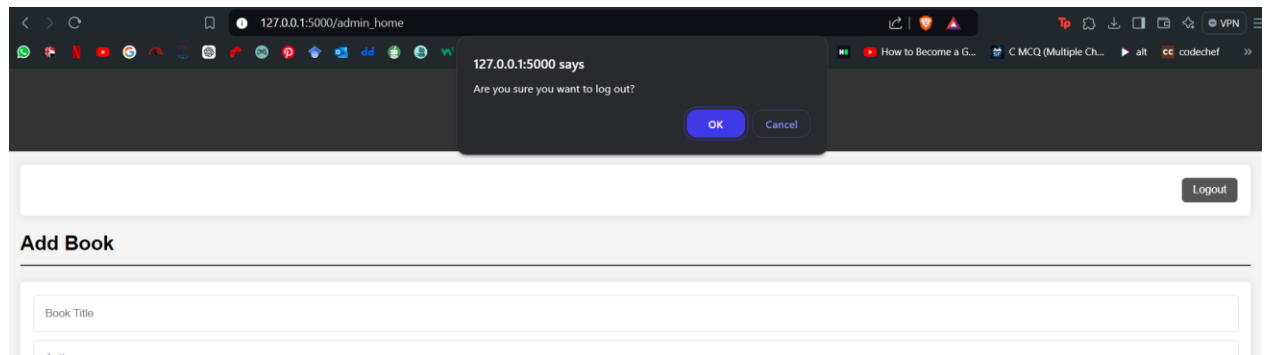


Figure 10 Logout Verification

## Normal User Page

- The Normal User Page provides several functionalities to interact with the library system, including:

### Library Inventory

- The user can view the current library inventory in the system.

### Search Book

- The user can search for books using two filtration methods:
  1. ISBN: Search by International Standard Book Number (ISBN)
  2. Keywords: Search by author name or book title

### Borrow/Return Books

- The user can borrow or return books from the library inventory.
- If a book is borrowed, its status will be updated to reflect this in the Book Status.

**Book Inventory**

Book Title	Author	ISBN	Genre	Date of Publish	Status	Action
Sherlock Holmes	Arthur Conan Doyle	1	Mystery	1892	Available	<button>Borrow</button>
Death Joke	Agatha Christie	2	Mystery	1930	Available	<button>Borrow</button>
Hands on Machine Learning	Aurelien Geron	3	Science	2022	Available	<button>Borrow</button>
80 Days around the World	Charles Dickens	4	Fantasy	1873	Available	<button>Borrow</button>
Big Brother	Ahmed Yehia	5	Mystery	12/7/1899	Available	<button>Borrow</button>
Big Brother 2	Ahmed Yehia	6	Fantasy	1899	Available	<button>Borrow</button>
Book4	AhmedYehia	7	Fantasy	1899	Available	<button>Borrow</button>
DevOps Graduation Project	Team 5	8	True Crime	2024	Available	<button>Borrow</button>

**Search by ISBN**

Search

**Search by Keyword**

Search

Figure 11 Normal User Page

**Book Inventory**

Book Title	Author	ISBN	Genre	Date of Publish	Status	Action
Sherlock Holmes	Arthur Conan Doyle	1	Mystery	1892	Borrowed	<button>Return</button>

**Search by ISBN**

Search

Book status change once the user borrows the book

User Can return the book

Figure 12 Search for a borrowed book

## Part 2: Dockerization:

- **Base Image:** The Dockerfile starts with a base image, which provides a foundation for the new image.
- **Working Directory:** The working directory is set to **/app**, which will be the root directory for the application code.
- **Copying Application Code:** The application code is copied into the container at the **/app** directory.
- **Installing Dependencies:** The dependencies specified in the **requirements.txt** file are installed using pip.
- **Exposing Port:** The necessary port (**5000**) is exposed, allowing the application to be accessed from outside the container.

### Building and Deploying the Docker Image

- **Building the Docker Image:** The Dockerfile is used to build a Docker image locally.
- **Testing the Docker Image:** The Docker image is tested locally to ensure it works as expected.
- **Pushing to Docker Hub:** The Docker image is pushed to Docker Hub, making it available for use in the Kubernetes deployment.

The resulting Docker image, is used in the Kubernetes deployment configuration to create a container running the library management system application.

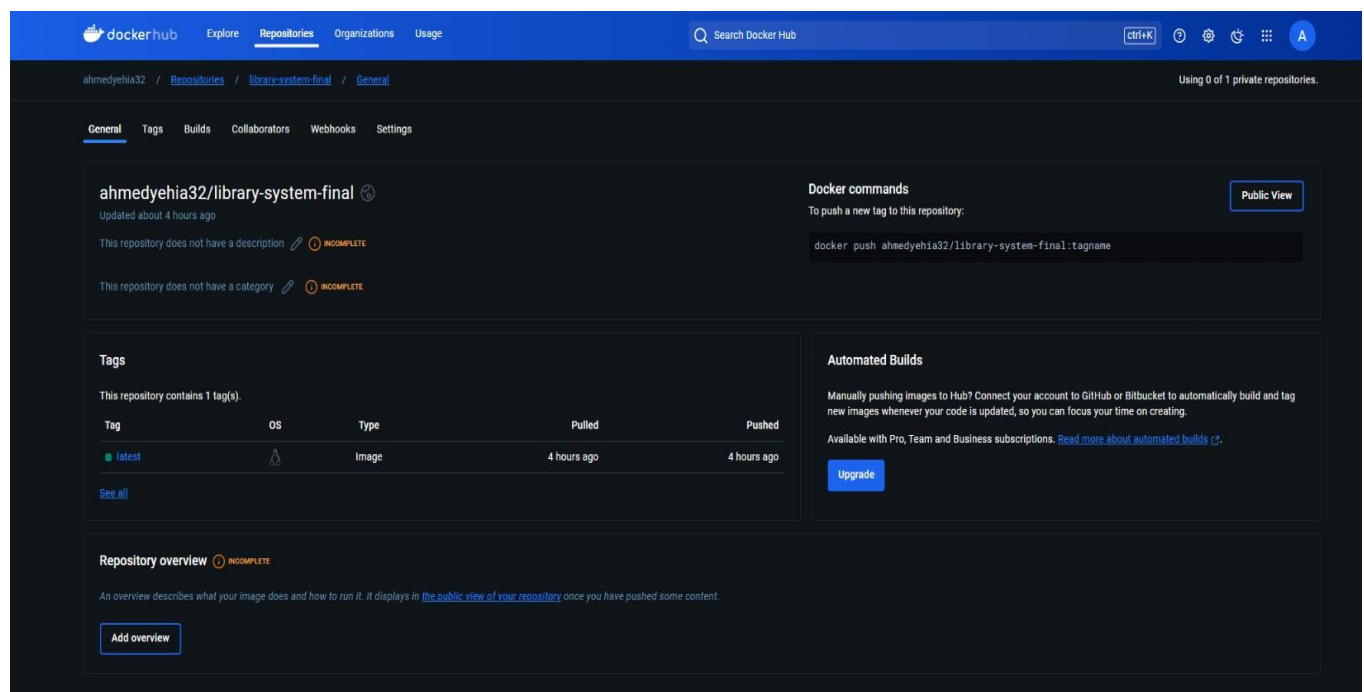


Figure 13 Docker image on dockerhub

## Part 3: Infrastructure as Code with Terraform

Terraform was used to create our infrastructure on AWS, we had two Terraform modules, one for creating the backend which is the S3 bucket for storing the state file, and the DynamoDB table for preventing multiple changes to the state file at the same time, and the other for creating our main infrastructure(VPC, subnets, Internet gateway, Nat gateway, EKS), our main module contains a module for every AWS resource.

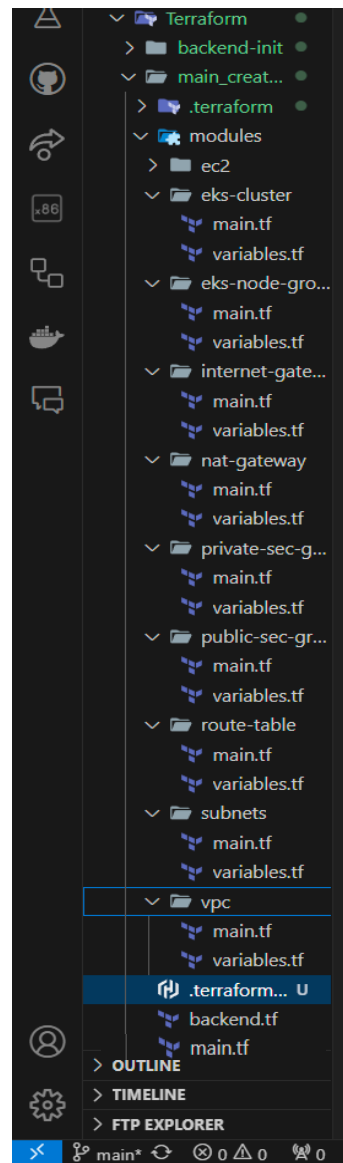


Figure 14 Terraform Modules

## Modules details:

### 1. VPC Configuration

- **Module:** `team5_vpc`
- **Source:** `./modules/vpc`
- **Description:** Creates a Virtual Private Cloud (VPC) with the specified CIDR block.
- **Parameters:**
  - **cidr\_block:** `10.0.0.0/16` - The IP address range for the VPC.
  - **name:** `team5-vpc` - Name of the VPC.

### 2. Subnet Configuration

- **Public Subnet 1:** `team5_public_subnet1`
  - **Source:** `./modules/subnets`
  - **Parameters:**
    - **vpc\_id:** Refers to the VPC ID created by `team5_vpc`.
    - **cidr\_block:** `10.0.1.0/24` - IP address range for the subnet.
    - **availability\_zone:** `eu-central-1a` - AWS availability zone.
    - **map\_public\_ip\_on\_launch:** `true` - Automatically assign public IP addresses.
    - **name:** `team5-public-subnet1` - Name of the subnet.
- **Public Subnet 2:** `team5_public_subnet2`
  - **Source:** `./modules/subnets`
  - **Parameters:**
    - **vpc\_id:** Refers to the VPC ID created by `team5_vpc`.
    - **cidr\_block:** `10.0.2.0/24` - IP address range for the subnet.
    - **availability\_zone:** `eu-central-1b` - AWS availability zone.
    - **map\_public\_ip\_on\_launch:** `true` - Automatically assign public IP addresses.
    - **name:** `team5-public-subnet2` - Name of the subnet.
- **Private Subnet 1:** `team5_private_subnet1`
  - **Source:** `./modules/subnets`
  - **Parameters:**
    - **vpc\_id:** Refers to the VPC ID created by `team5_vpc`.
    - **cidr\_block:** `10.0.3.0/24` - IP address range for the subnet.
    - **availability\_zone:** `eu-central-1a` - AWS availability zone.
    - **map\_public\_ip\_on\_launch:** `false` - Do not assign public IP addresses.
    - **name:** `team5-private-subnet1` - Name of the subnet.
- **Private Subnet 2:** `team5_private_subnet2`
  - **Source:** `./modules/subnets`
  - **Parameters:**
    - **vpc\_id:** Refers to the VPC ID created by `team5_vpc`.
    - **cidr\_block:** `10.0.4.0/24` - IP address range for the subnet.
    - **availability\_zone:** `eu-central-1b` - AWS availability zone.
    - **map\_public\_ip\_on\_launch:** `false` - Do not assign public IP addresses.
    - **name:** `team5-private-subnet2` - Name of the subnet.

### 3. Internet Gateway

- **Module:** `team5_internet_gateway`
- **Source:** `./modules/internet-gateway`
- **Parameters:**
  - **vpc\_id:** Refers to the VPC ID created by `team5_vpc`.
  - **name:** `team5-igw` - Name of the Internet Gateway.

### 4. NAT Gateway

- **Module:** `team5_nat_gateway`
- **Source:** `./modules/nat-gateway`
- **Parameters:**
  - **subnet\_id:** Refers to the public subnet ID where the NAT Gateway is deployed (`team5_public_subnet1`).
  - **name:** `team5-nat-gateway` - Name of the NAT Gateway.



## 5. Route Tables

- **Modules:**
  - **Public Route Table: team5\_public\_route\_table**
    - **Source:** ./modules/route-table
    - **Parameters:**
      - **vpc\_id:** Refers to the VPC ID created by team5\_vpc.
      - **cidr\_block:** 0.0.0.0/0 - Route all traffic.
      - **gateway\_id:** Refers to the Internet Gateway ID created by team5\_internet\_gateway.
      - **name:** team5-public-route-table - Name of the route table.
  - **Private Route Table: team5\_private\_route\_table**
    - **Source:** ./modules/route-table
    - **Parameters:**
      - **vpc\_id:** Refers to the VPC ID created by team5\_vpc.
      - **cidr\_block:** 0.0.0.0/0 - Route all traffic.
      - **nat\_gateway\_id:** Refers to the NAT Gateway ID created by team5\_nat\_gateway.
      - **name:** team5-private-route-table - Name of the route table.

### Route Table Associations

- **Public Subnet Associations:**
  - **team5\_public\_subnet\_a\_assoc:**  
Associates team5\_public\_subnet1 with team5\_public\_route\_table.
  - **team5\_public\_subnet\_b\_assoc:**  
Associates team5\_public\_subnet2 with team5\_public\_route\_table.
- **Private Subnet Associations:**
  - **team5\_priv\_subnet\_a\_assoc:**  
Associates team5\_private\_subnet1 with team5\_private\_route\_table.
  - **team5\_priv\_subnet\_b\_assoc:**  
Associates team5\_private\_subnet2 with team5\_private\_route\_table.

## 6. IAM Roles for EKS

- **IAM Role for EKS Cluster:**
  - **Resource:** `aws_iam_role.eks_cluster_role`
  - **Description:** Role for the EKS cluster to assume.
  - **Assume Role Policy:** Allows the EKS service to assume the role.
  - **Policy Attachments:**
    - `AmazonEC2FullAccess`
    - `AmazonEKSClusterPolicy`
    - `AmazonEKSServicePolicy`
- **IAM Role for EKS Node Group:**
  - **Resource:** `aws_iam_role.team5_eks_node_role`
  - **Description:** Role for the EKS node group.
  - **Assume Role Policy:** Allows EC2 instances to assume the role.
  - **Managed Policies:**
    - `AmazonEKSWorkerNodePolicy`
    - `AmazonEC2ContainerRegistryReadOnly`
    - `AmazonEC2ContainerServiceRole`
    - `AmazonEKS_CNI_Policy`

## 7. EKS Cluster and Node Group

- **EKS Cluster:**
  - **Module:** `eks_cluster`
  - **Source:** `./modules/eks-cluster`
  - **Parameters:**
    - **cluster\_name:** `team5-eks-cluster` - Name of the EKS cluster.
    - **cluster\_role\_arn:** ARN of the IAM role for the EKS cluster.
    - **subnet\_ids:** List of subnet IDs for the cluster.
- **EKS Node Group:**
  - **Module:** `eks_node_group`
  - **Source:** `./modules/eks-node-group`
  - **Parameters:**
    - **cluster\_name:** Name of the EKS cluster (`team5-eks-cluster`).
    - **node\_group\_name:** `team5-node-group` - Name of the node group.
    - **node\_role\_arn:** ARN of the IAM role for the node group.
    - **subnet\_ids:** List of public subnet IDs for the node group.
    - **desired\_size:** `1` - Desired number of nodes.
    - **max\_size:** `1` - Maximum number of nodes.
    - **min\_size:** `1` - Minimum number of nodes.

## Part 4: Kubernetes Deployment on EKS:

The website was deployed on Amazon Elastic Kubernetes Service (EKS), after applying the terraform code for creating the infrastructure, we configured our deployment.yaml and service.yaml files for deploying our website on the created EKS cluster.

Yaml Files

1-library-management-deployment.yaml file:

```
ent > 📄 library-management-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: library-management-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: library-management
  template:
    metadata:
      labels:
        app: library-management
    spec:
      containers:
        - name: library-management-container
          image: ahmedyehia32/library-system-final
          imagePullPolicy: Always
          ports:
            - containerPort: 5000
            - containerPort: 8000 # Expose the metrics port
```

Figure 15 Deployment Yaml File

2-service.yaml file:

```
apiVersion: v1
kind: Service
metadata:
  name: library-management-service
spec:
  selector:
    app: library-management
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 5000
    - name: metrics
      protocol: TCP
      port: 8000
      targetPort: 8000
  type: LoadBalancer
```

Figure 16 Service Yaml File

## Deployment Configuration (library-management-deployment.yaml)

- **API Version: apps/v1** - The API version used to manage deployments in Kubernetes.
- **Kind: Deployment** - Specifies that this YAML defines a Kubernetes Deployment object.
- **Metadata:**
  - **Name: library-management-deployment** - The name assigned to the Deployment object.
- **Spec:**
  - **Replicas: 1** - Defines the number of pod replicas to be created.
  - **Selector:** Ensures the Deployment targets pods that match specific labels:
    - **Match Labels: app: library-management** - Identifies the pods managed by the deployment based on this label.
  - **Template:** Defines the specifications for the pods that will be created by the Deployment.
    - **Metadata:**
      - **Labels: app: library-management** - Specifies the labels applied to each pod.
    - **Spec:** Defines the container configuration for each pod.
      - **Containers:**
        - **Name: library-management-container** - The name of the container within the pod.
        - **Image: ahmedyehia32/library-system-final** - The Docker image to be used.
        - **Image Pull Policy: Always** - Ensures the image is always pulled from the registry.
        - **Ports:**
          - **Container Port: 5000** - The port number inside the container where the application is accessible.
          - **Container Port: 8000** - Designated for exposing metrics from the application, typically used for Prometheus to scrape.

## Service Configuration (service.yaml)

- **API Version: v1** - The version for Kubernetes Service API.
- **Kind: Service** - Specifies that this YAML defines a Kubernetes Service object.
- **Metadata:**
  - **Name: library-management-service** - The name of the service, which will be used to reference this service within the cluster.
- **Spec:**
  - **Selector:**
    - **App: library-management** - This selector matches the labels on the pods created by the deployment, allowing the service to route traffic to them.
  - **Ports:**
    - **Protocol: TCP** - The protocol used by the service to communicate with the pods.
    - **Port: 80** - The port that external clients will use to access the service.
    - **Target Port: 5000** - The port on the container where the application is running.
    - **Type: LoadBalancer** - Specifies that this service is exposed to the internet via a load balancer, making it accessible outside the Kubernetes cluster.
  - **Note:** Port **8000** is used for metrics collection, routing traffic directly to port **8000** on the backend pods, which is used by Prometheus to scrape metrics from the application.

## Part 5: CI/CD Pipeline Setup:

Our (CICD) pipeline involves automating the process of integrating code changes and deploying our website. We implemented our (CICD) pipeline using Jenkins. We configured the pipeline to automatically build, and deploy code whenever the pipeline is executed.

### Pipeline Stages:

#### 1. Checkout Code

```
stages {
    stage('Checkout Code') {
        steps {
            git url: "${GITHUB_REPO}", branch: 'main', credentialsId: 'github-token-i
        }
    }
}
```

Figure 17 Pipeline 1. Checkout Code

**Description:** This stage checks out the code from the specified GitHub repository (`GITHUB_REPO`) using the provided credentials (`github-token-id`). The pipeline will pull the `main` branch.

#### 2- Build Docker Image:

```
stage('Build Docker Image') {
    steps {
        script {
            // Ensure Dockerfile is in the correct directory
            sh 'docker build -t ${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG} .'
        }
    }
}
```

Figure 18 Pipeline Build Docker Image

- **Description:** In this stage, Jenkins builds a Docker image using the `Dockerfile` located at the root of the project. The image is tagged as `${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG}`, which is `library-system-final:latest` by default.

### 3- Push Docker Image:

```
stage('Push Docker Image') {
  steps {
    withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials-id', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) {
      script {
        // Login to Docker Hub
        sh 'echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin'

        // Tag the Docker image
        sh 'docker tag ${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG} ${DOCKER_USERNAME}/${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG}'

        // Push the Docker image and capture output
        sh '''
        set -x
        docker push ${DOCKER_USERNAME}/${DOCKER_IMAGE_NAME}:${DOCKER_IMAGE_TAG} 2>&1 | tee docker_push.log
        '''
      }
    }
  }
}
```

Figure 19 pipeline Push Docker Image:

- **Description:**

- Logs in to Docker Hub using credentials (**dockerhub-credentials-id**).
- Tags the built Docker image with the Docker Hub repository name.
- Pushes the Docker image to Docker Hub and logs the output.

### 4- Deploy to EKS:

```
stage('Deploy to EKS') {
  steps {
    withCredentials([usernamePassword(credentialsId: 'aws-credentials-id', usernameVariable: 'AWS_ACCESS_KEY_ID', passwordVariable: 'AWS_SECRET_ACCESS_KEY')]) {
      script {
        sh 'aws configure set aws_access_key_id $AWS_ACCESS_KEY_ID'
        sh 'aws configure set aws_secret_access_key $AWS_SECRET_ACCESS_KEY'
        sh 'aws configure set region ${AWS_REGION}'
        sh "aws eks --region ${AWS_REGION} update-kubeconfig --name ${EKS_CLUSTER_NAME}"
        sh 'kubectl apply -f Deployment/library-management-deployment.yaml'
        sh 'kubectl apply -f Deployment/service.yaml'
      }
    }
  }
}
```

Figure 20 Pipeline Deploy to EKS

- **Description:**

- Configures AWS CLI with credentials (**aws-credentials-id**), sets the region, and updates the kubeconfig to connect to the EKS cluster.
- Uses **kubectl apply** to deploy the application in Kubernetes by applying the **library-management-deployment.yaml** and **service.yaml** files.



## 5-Get the Load Balancer IP:

```
stage('Get Load Balancer IP') {  
    steps {  
        script {  
            sleep(time: 60, unit: 'SECONDS')  
            def loadBalancerIP = sh(script: 'kubectl get svc library-management-service -o jsonpath="{.status.loadBalancer.ingress[0].hostname}"', returnStdout: true).trim()  
            echo "Load Balancer IP: ${loadBalancerIP}"  
        }  
    }  
}
```

*Figure 21 Pipeline Loadbalancer*

### Description:

- Waits for 60 seconds to ensure the Load Balancer is fully initialized.
- Retrieves the Load Balancer IP address or hostname from the `library-management-service` using `kubectl`.
- Displays the Load Balancer IP/hostname in the console output.

# Bonus Task: Set Up Monitoring and Logging:

## Overview

Prometheus and Grafana are popular tools for monitoring and visualizing metrics. Prometheus is used to collect and store metrics data, while Grafana is used to visualize this data through dashboards.

## Components:

- 1- Prometheus: A monitoring system and time-series database.
- 2- Grafana: An open-source platform for monitoring and observability, which supports various data sources including Prometheus.

## Setup Prometheus

### Prometheus ConfigMap (prometheus-configmap.yaml)

- 1- Create a ConfigMap to provide the Prometheus

```
ing > prometheus-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s

    scrape_configs:
      - job_name: 'library-management'
        metrics_path: '/metrics'
        static_configs:
          - targets: ['library-management-service.default.svc.cluster.local:8000'] # Fully qualified domain name
```

Figure 22 configuration to the Prometheus deployment.

### Prometheus Deployment (prometheus-deployment.yaml)

- 2- Defines the deployment of Prometheus.

```
ing > prometheus-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus:latest
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
          volumeMounts:
            - name: prometheus-config
              mountPath: /etc/prometheus
          volumes:
            - name: prometheus-config
              configMap:
                name: prometheus-config
```

Figure 23 Prometheus deployment yaml file.

## Prometheus Service (prometheus-service.yaml)

- 3- Defines the service for Prometheus.

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: monitoring
spec:
  selector:
    app: prometheus
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9090
  type: LoadBalancer
```

*Figure 24 Prometheus service File*

### Deployment Commands to apply the configurations:

```
kubectl apply -f prometheus-configmap.yaml
```

```
kubectl apply -f prometheus-deployment.yaml
```

```
kubectl apply -f prometheus-service.yaml
```

# Grafana Setup

## 1- Grafana Deployment (grafana-deployment.yaml)

Defines the deployment of Grafana.

```
g > grafana-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana:11.2.0
          ports:
            - containerPort: 3000
          volumeMounts:
            - name: grafana-data
              mountPath: /var/lib/grafana
      volumes:
        - name: grafana-data
          emptyDir: {}
```

Figure 25 Grafana Deployment yaml file

## 2- Grafana Service with LoadBalancer

(grafana-service loadbalancer.yaml)

Defines the service for Grafana with a LoadBalancer to expose it.

```
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: monitoring
spec:
  selector:
    app.kubernetes.io/name: grafana
    app.kubernetes.io/instance: grafana
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

*Figure 26 Grafana Service Loadbalancer Yaml File*

### Deployment Commands Apply configurations:

```
kubectl apply -f grafana-deployment.yaml
```

```
kubectl apply -f grafana-service-loadbalancer.yaml
```

# Integrating Prometheus with Grafana

## Access Grafana

Get the external IP address of the Grafana service: `kubectl get svc grafana -n monitoring`  
Navigate to the Grafana URL provided by the LoadBalancer.

## Add Prometheus as a Data Source

Log in to Grafana (default credentials: admin/admin). Go to Configuration (Gear icon)  
> Data Sources.

Add Data Source and select Prometheus.

Configure Prometheus URL to `http://prometheus:9090` (assuming Prometheus is within the same namespace or adjust accordingly if using an external IP).

## Create Dashboards

Go to Dashboards and click New Dashboard.

Add Panels and configure queries to visualize metrics from Prometheus.

Example Query for HTTP Requests: `http_requests_total`

Some Screenshots from Prometheus:

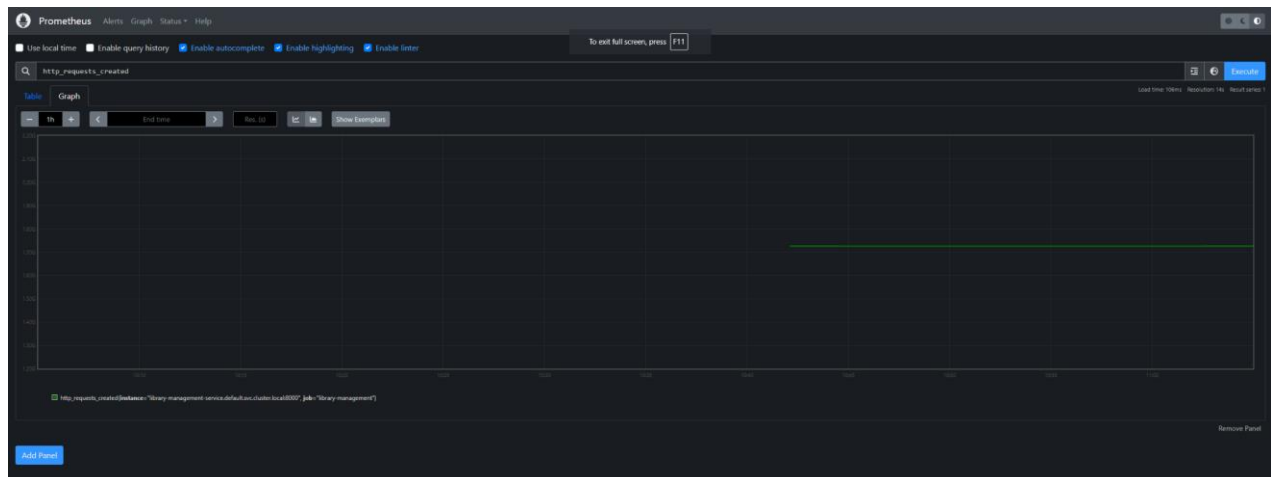
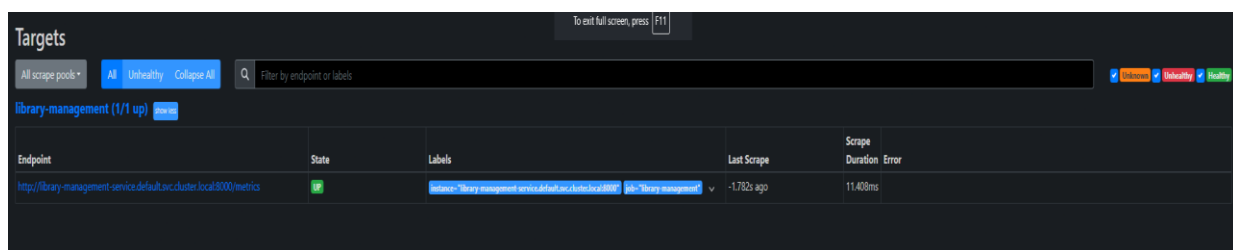


Figure 27 Prometheus Chart

The screenshot shows the 'Targets' page in the Prometheus web interface. It displays a table of scrape targets. The table has columns for 'Endpoint', 'State', 'Labels', 'Last Scrape', and 'Scrape Duration'. There is one target listed: 'http://library-management-service.default.svc.cluster.local:8000/metrics'. The state is 'UP', the last scrape was '~1.782s ago', and the scrape duration was '11.408ms'.

Endpoint	State	Labels	Last Scrape	Scrape Duration
http://library-management-service.default.svc.cluster.local:8000/metrics	UP	namespace="library-management-service" default.svc.cluster.local:8000	~1.782s ago	11.408ms

Figure 28 Prometheus Target

## Some Screenshots from Grafana:

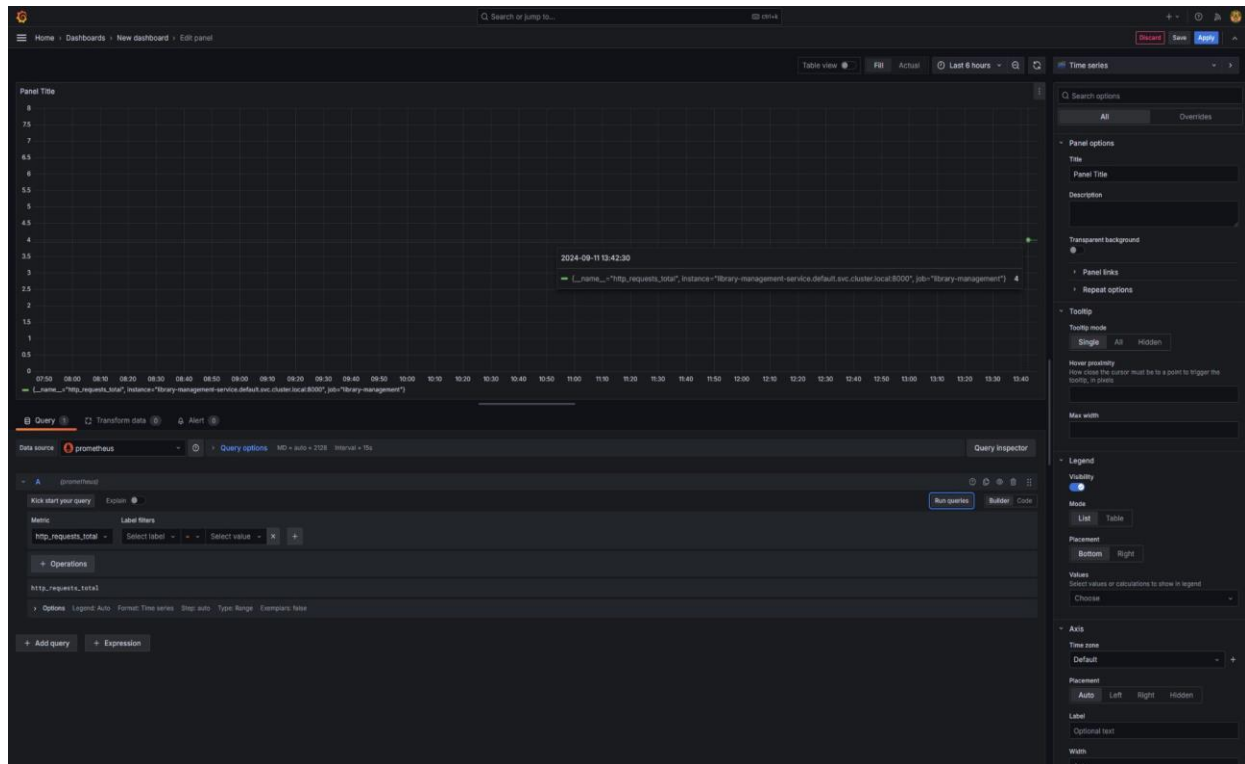


Figure 29 Number of http Request

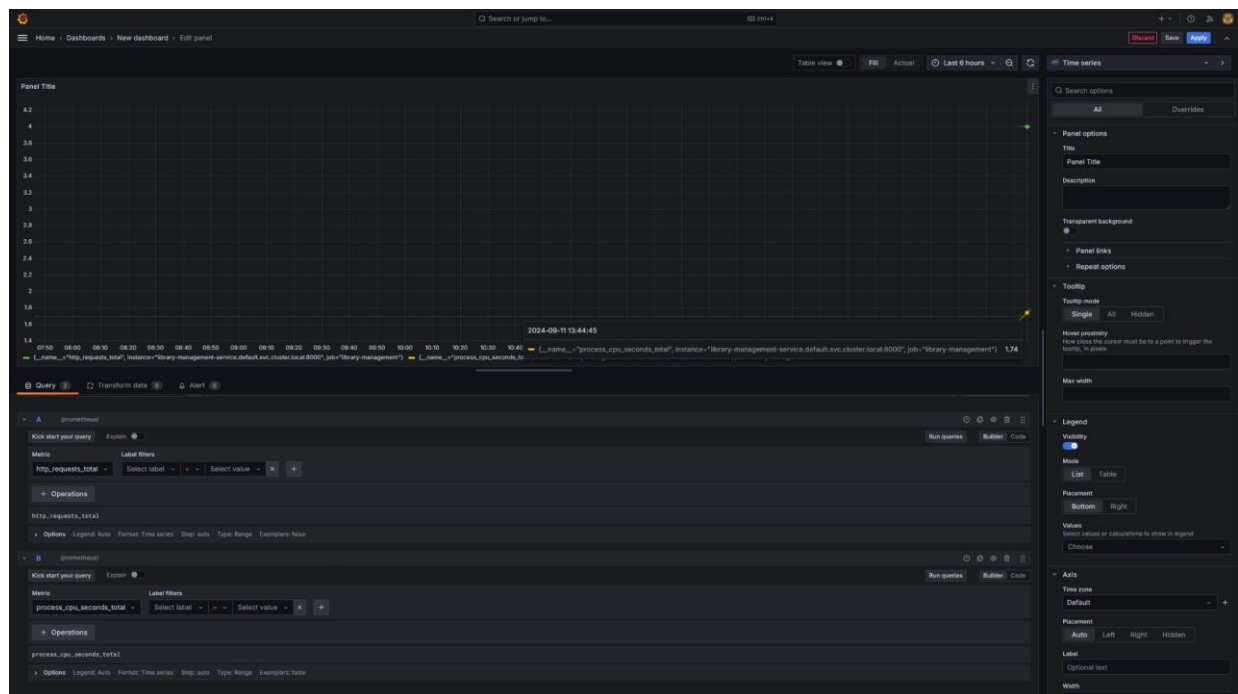


Figure 30 Process CPU second



## Terraform Pipeline (Bonus task)

### Terraform Infrastructure Setup

Terraform was used to create the infrastructure on AWS, consisting of two main modules:

#### *Backend Module*

- **S3 Bucket:** Stores the Terraform state file.
- **DynamoDB Table:** Prevents multiple changes to the state file at the same time.

#### **Figure 14:** *Terraform Modules*

#### **Main Infrastructure Module**

- **VPC:** Virtual Private Cloud for the infrastructure.
- **Subnets:** Organize and segment the network.
- **Internet Gateway:** Enables communication between the VPC and the internet.
- **NAT Gateway:** Allows outbound internet access from private subnets.
- **EKS:** Elastic Kubernetes Service cluster for the Kubernetes deployment.

### Pipeline Stages

The pipeline consists of the following stages:

#### 1. Checkout SCM

Purpose: This stage checks out the code from the specified Git repository and branch.

- Configuration:
  - Uses GitSCM plugin for checking out the code.
  - Branch: \*/main
  - Repository URL: <https://github.com/Ahmedyehia12/LibraryManagmentSystem>

- Credentials ID: c6d6be8b-c4b5-450b-a13c-1b8aca95fc69 (used for authentication with the Git repository).

## 2. Setup Plugin Cache Directory

### Setup Plugin Cache Directory Stage

- Purpose: This stage sets up a directory for caching Terraform plugins, which can speed up the Terraform operations by avoiding repeated downloads.
- Configuration:
  - Creates a cache directory at /var/lib/jenkins/.terraform.d/plugin-cache using the mkdir command.

## 3. Terraform Init - Backend

### Terraform Init - Backend Stage

- Purpose: Initializes the Terraform configuration for the backend, which is responsible for managing the remote state.
- Configuration:
  - Runs within the Terraform/backend-init directory.
  - Uses AWS credentials (aws-creds) to access AWS resources needed for the backend setup.
  - Executes terraform init to initialize the backend configuration.

## 4. Terraform Apply - Backend

### Terraform Apply - Backend Stage

- Purpose: Applies the Terraform configuration for the backend to set up the remote state storage.
- Configuration:
  - Runs within the Terraform/backend-init directory.
  - Uses AWS credentials (aws-creds).
  - Executes terraform apply -auto-approve to automatically apply the configuration without manual approval.
- Post Action:
  - On failure, the stage will:
    - Set the build result to FAILURE.
    - Echo a failure message.
    - Destroy the resources created during this stage using terraform destroy -auto-approve.

## 5. Terraform Init - Main Creation

### Terraform Init - Main Creation Stage

- Purpose: Initializes the main Terraform configuration responsible for creating infrastructure.
- Conditions:
  - Only runs if the previous stages did not result in failure.
- Configuration:

- Runs within the Terraform/main\_creation directory.
- Uses AWS credentials (aws-creds).
- Executes terraform init to initialize the main creation configuration.
- Post Action:
  - On failure, the stage will:
    - Set the build result to FAILURE.
    - Echo a failure message.
    - Clean up resources created in the backend stage by destroying them using terraform destroy - auto-approve.

## 6. Terraform Apply - Main Creation

### Figure 25: Terraform Apply - Main Creation Stage

- Purpose: Applies the main Terraform configuration to create the required infrastructure.
- Conditions:
  - Only runs if the previous stages did not result in failure.
- Configuration:
  - Runs within the Terraform/main\_creation directory.
  - Uses AWS credentials (aws-creds).
  - Executes terraform apply -auto-approve to automatically apply the configuration.
- Post Action:
  - On failure, the stage will:
    - Set the build result to FAILURE.
    - Echo a failure message.
    - Destroy the resources created in the main creation and backend stages using terraform destroy - auto-approve.

### Post Actions

- Always:
  - Cleans up the workspace (cleanWs()), ensuring that the workspace is clean after every pipeline run. This helps to avoid issues caused by leftover files from previous builds.
- Failure:
  - Echoes 'Pipeline failed!' to indicate that the pipeline encountered errors.
- Success:
  - Echoes 'Pipeline succeeded!' to indicate that all stages were completed successfully.