Banque Misr internship 2024 Graduation Project Team5

Team members:

1- Ahmed Yehia (Scrum Master)
2- Marwan Mohammed
3- Safia Adel
4- Sama Hatem

Part 1: Application Development

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

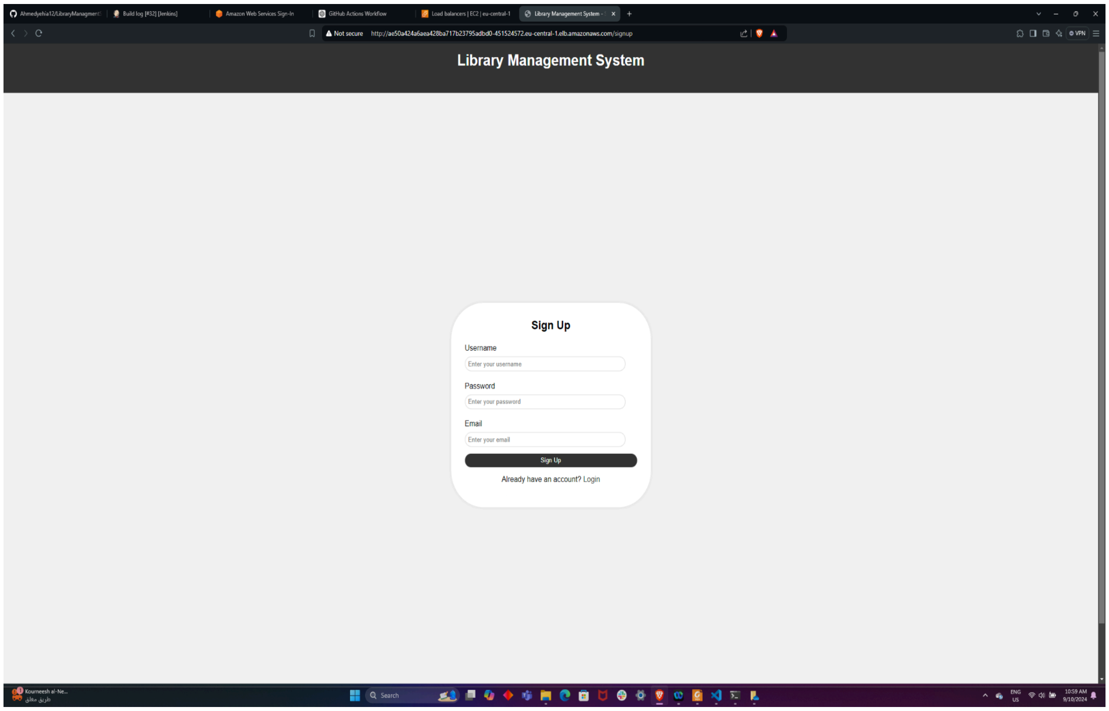
1- Users for this application are either admins or users.

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.

Here is the login and signup pages:


Ahmedyehia12/LibraryManagment
Build log [#32] [Jenkins]
Amazon Web Services Sign-In
GitHub Actions Workflow
Load balancers | EC2 | eu-central-1
Library Management System
Not secure

http://ae50a424a6aea428ba717b23795adbd0-451524572.eu-central-1.elb.amazonaws.com/signup

Library Management System

Sign Up

Username

Enter your username

Password

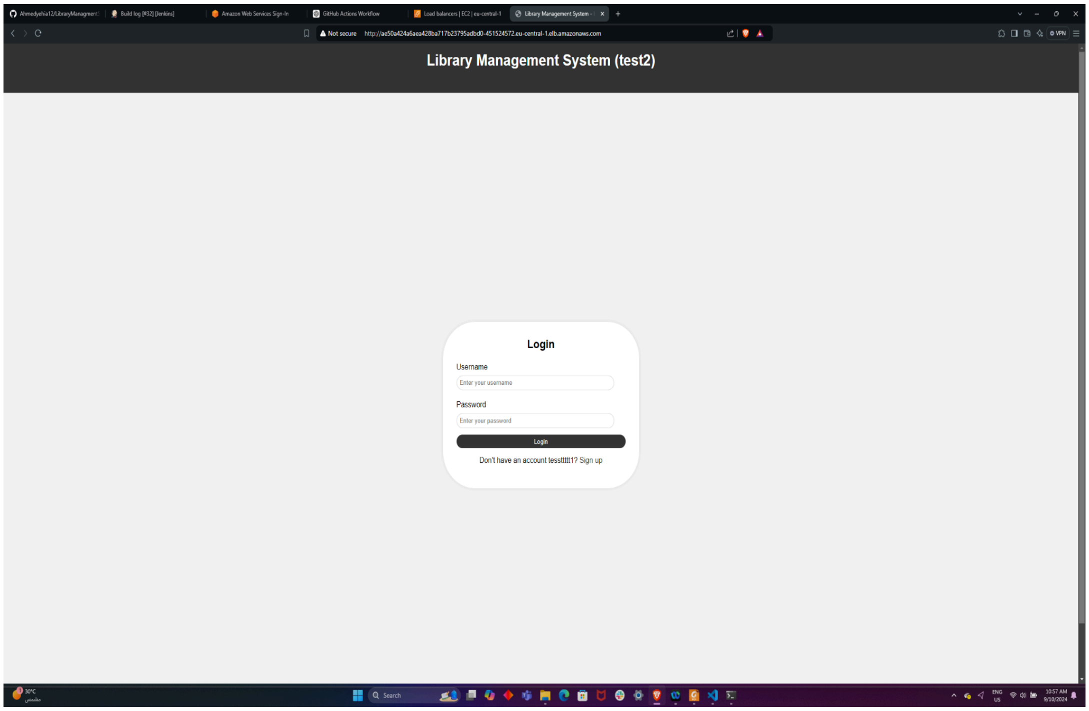
Enter your password

Email

Enter your email

Sign Up

Already have an account? Login


Ahmedyehia12/LibraryManagment
Build log [#32] [Jenkins]
Amazon Web Services Sign-In
GitHub Actions Workflow
Load balancers | EC2 | eu-central-1
Library Management System
Not secure

http://ae50a424a6aea428ba717b23795adbd0-451524572.eu-central-1.elb.amazonaws.com

Library Management System (test2)

Login

Username

Enter your username

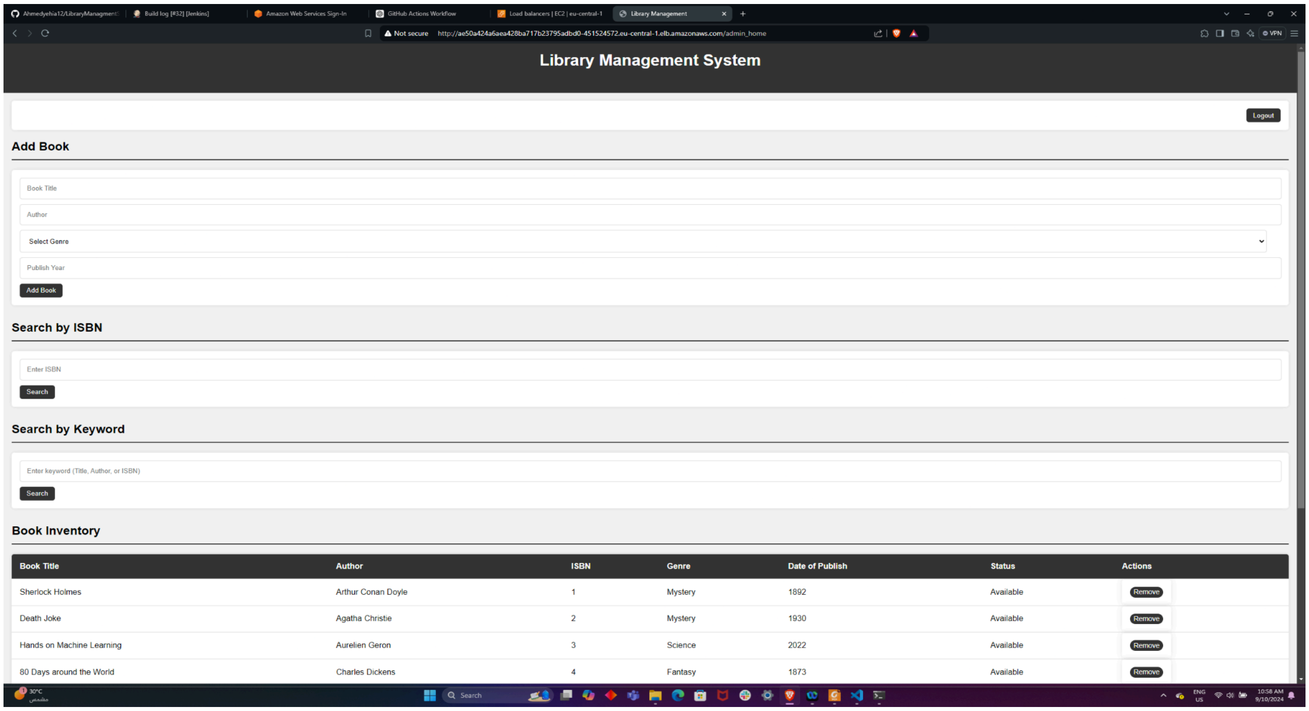
Password

Enter your password

Login

Don't have an account tessttttt1? Sign up

Here is the admin UI:



Ahmedyehia12/LibraryManagmentS
Build log [#32] [Jenkins]
Amazon Web Services Sign-In
GitHub Actions Workflow
Load balancers | EC2 | eu-central-1
Library Management
Not secure
http://ae50a424a6aea428ba717b23795adbd0-451524572.eu-central-1.elb.amazonaws.com/admin_home
VPN
Search
30°C

Library Management System

Logout

Add Book

Book Title

Author

Select Genre

Publish Year

Add Book

Search by ISBN

Enter ISBN

Search

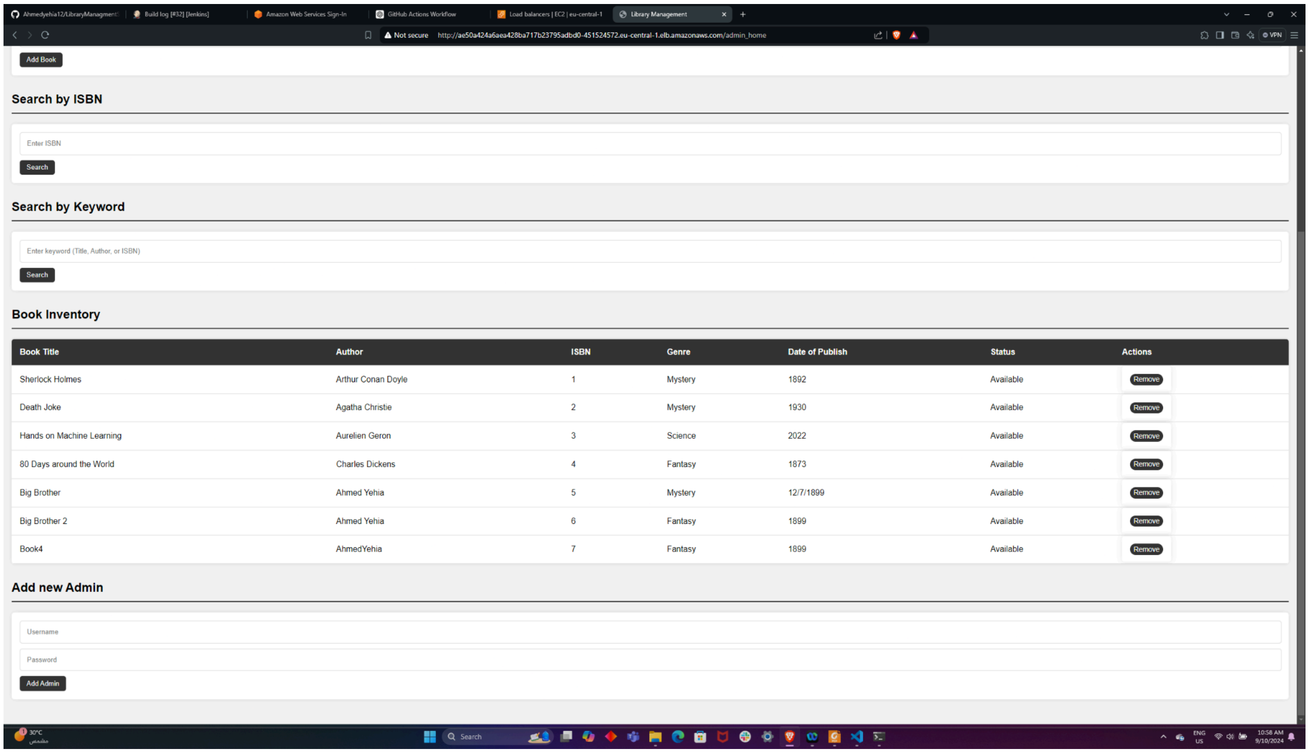
Search by Keyword

Enter keyword (Title, Author, or ISBN)

Search

Book Inventory

Book Title	Author	ISBN	Genre	Date of Publish	Status	Actions
Sherlock Holmes	Arthur Conan Doyle	1	Mystery	1892	Available	Remove
Death Joke	Agatha Christie	2	Mystery	1930	Available	Remove
Hands on Machine Learning	Aurelien Geron	3	Science	2022	Available	Remove
80 Days around the World	Charles Dickens	4	Fantasy	1873	Available	Remove



Ahmedyehia12/LibraryManagmentS
Build log [#32] [Jenkins]
Amazon Web Services Sign-In
GitHub Actions Workflow
Load balancers | EC2 | eu-central-1
Library Management
Not secure
http://ae50a424a6aea428ba717b23795adbd0-451524572.eu-central-1.elb.amazonaws.com/admin_home
VPN
Search
30°C

Add Book

Search by ISBN

Enter ISBN

Search

Search by Keyword

Enter keyword (Title, Author, or ISBN)

Search

Book Inventory

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

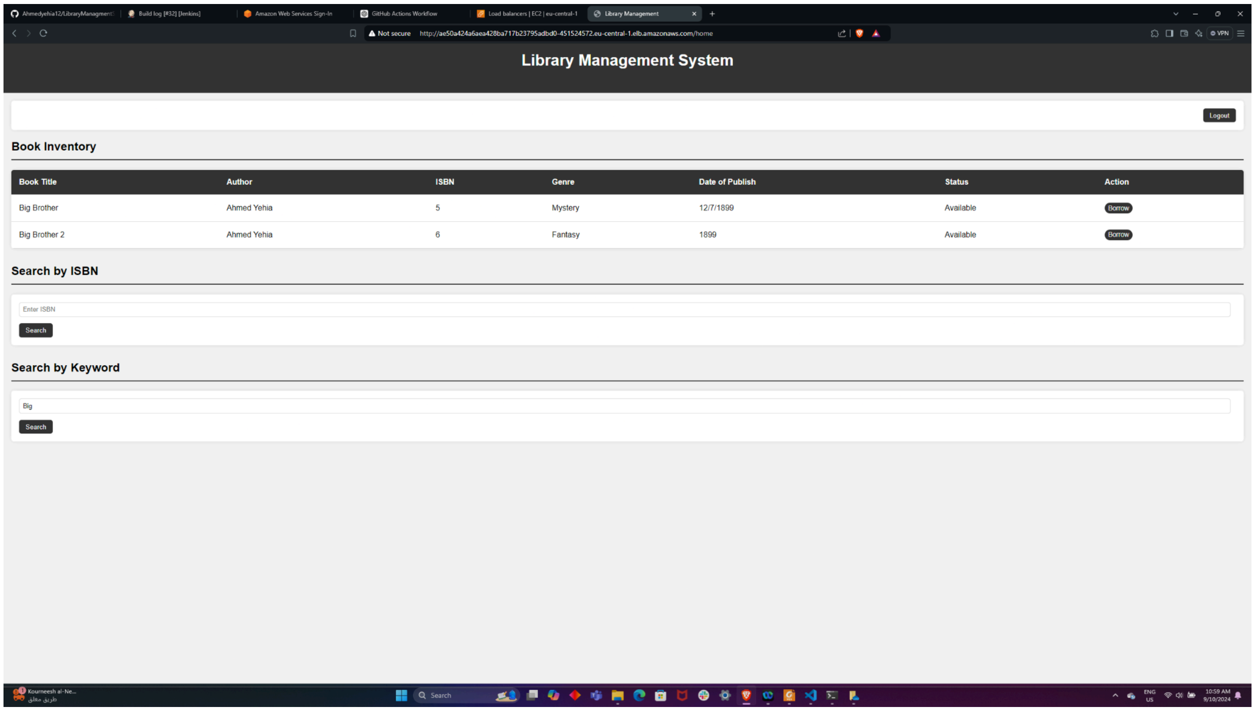
Add new Admin

Username

Password

Add Admin

Here is the User UI:



Ahmedyehia12/LibraryManagment
Build log [#32] [Jenkins]
Amazon Web Services Sign-In
GitHub Actions Workflow
Load balancers | EC2 | eu-central-1
Library Management
Not secure
http://ae50a424a6aea428ba717b23795adbd0-451524572.eu-central-1.elb.amazonaws.com/home
VPN

Library Management System

Logout

Book Inventory

Book Title	Author	ISBN	Genre	Date of Publish	Status	Action
Big Brother	Ahmed Yehia	5	Mystery	12/7/1899	Available	Borrow
Big Brother 2	Ahmed Yehia	6	Fantasy	1899	Available	Borrow

Search by ISBN

Enter ISBN

Search

Search by Keyword

Big

Search

Search
10:59 AM
9/10/2024

Tools Used:

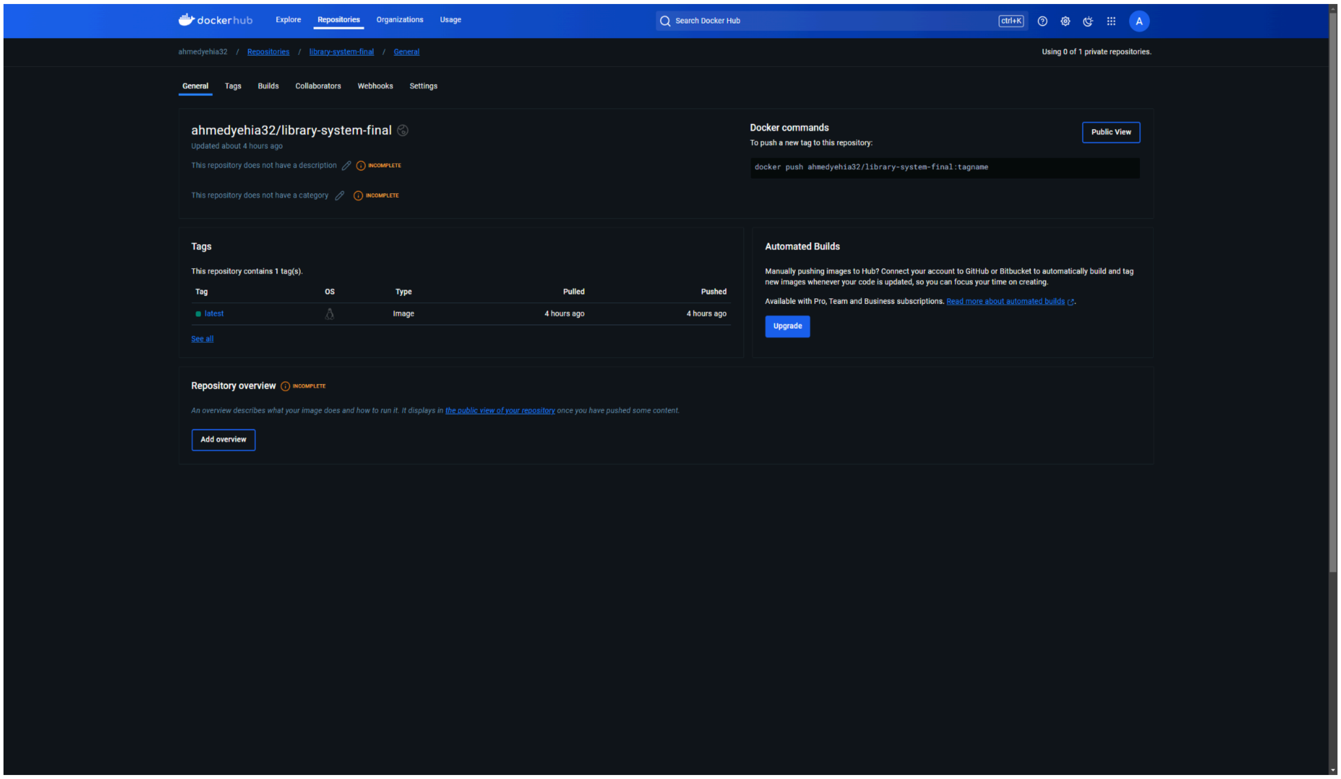
1- Frontend: (JavaScript, HTML, CSS)

2- Backend: (Python (Flask))

3- Database: our database is Json files.

Part 2: Dockerization:

We created a Dockerfile that contains the instructions to build a Docker image. After specifying the base image, setting the working directory, copying the application code, installing dependencies from the (requirements.txt) file, and exposing the necessary port, we containerized our application into a container. After building and testing our Docker Image locally, We pushed the Docker Image to Docker hub.



ahmedyehia32 / Repositories / library-system-final / General

Using 0 of 1 private repositories.

ahmedyehia32/library-system-final

Updated about 4 hours ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

Docker commands

To push a new tag to this repository:

```
docker push ahmedyehia32/library-system-final:tagname
```

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	4 hours ago	4 hours ago

See all

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

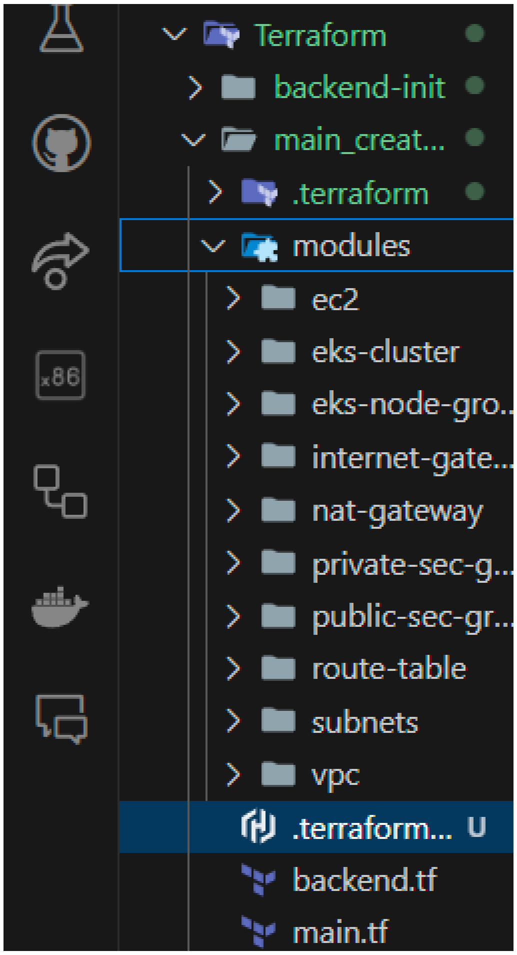
Available with Pro, Team and Business subscriptions. Read more about automated builds.

Repository overview INCOMPLETE

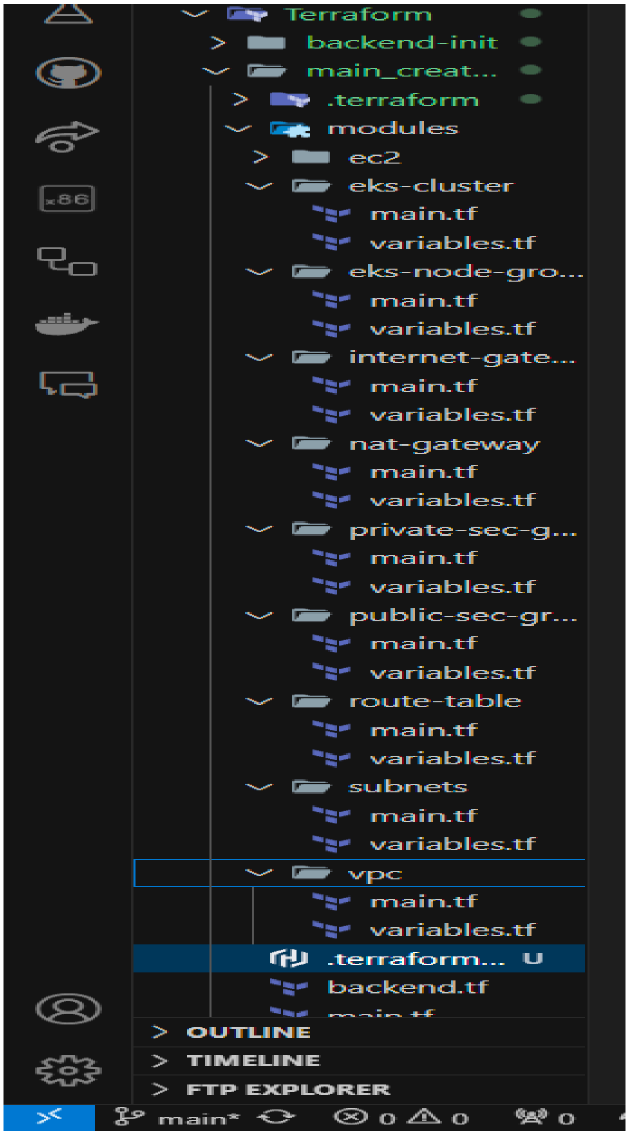
An overview describes what your image does and how to run it. It displays in the public view of your repository once you have pushed some content.

Part 3: Infrastructure as Code with Terraform

We used Terraform 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.



```
∨ Terraform
  > backend-init
  ∨ main_creat...
    > .terraform
    ∨ modules
      > ec2
      > eks-cluster
      > eks-node-gro...
      > internet-gate...
      > nat-gateway
      > private-sec-g...
      > public-sec-gr...
      > route-table
      > subnets
      > vpc
      .terraform... U
      backend.tf
      main.tf
```



```
∨ Terraform
  > backend-init
  ∨ main_creat...
    > .terraform
    ∨ modules
      > ec2
      ∨ eks-cluster
          main.tf
          variables.tf
      ∨ eks-node-gro...
          main.tf
          variables.tf
      ∨ internet-gate...
          main.tf
          variables.tf
      ∨ nat-gateway
          main.tf
          variables.tf
      ∨ private-sec-g...
          main.tf
          variables.tf
      ∨ public-sec-gr...
          main.tf
          variables.tf
      ∨ route-table
          main.tf
          variables.tf
      ∨ subnets
          main.tf
          variables.tf
      ∨ vpc
          main.tf
          variables.tf
      .terraform... U
      backend.tf
      main.tf
> OUTLINE
> TIMELINE
> FTP EXPLORER
main*  ⊗ 0 ⚠ 0  0
```

Here is the documentation for each module:

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**
Modules:
Public Subnet 1
Module: 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

Module: 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

Module: 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
Module: 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
Module: 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
Module: 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.

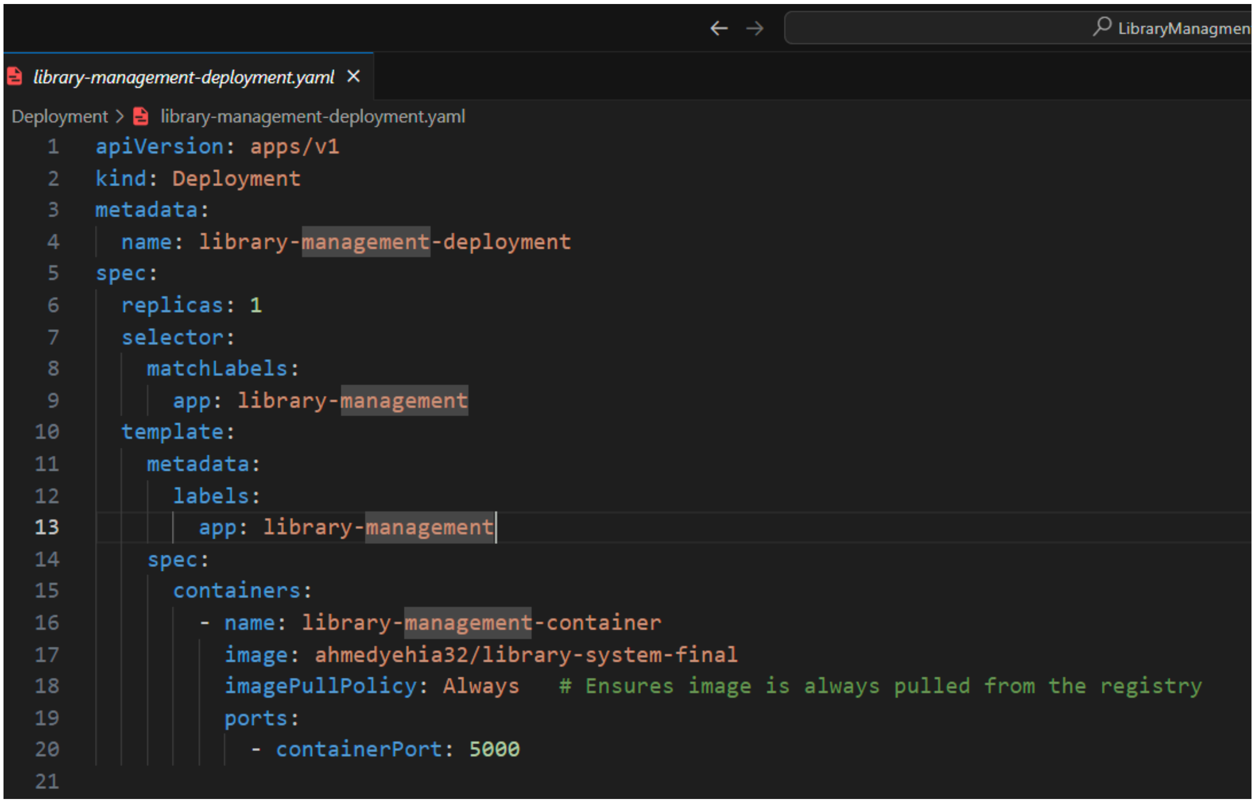
EKS Node Group

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:

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

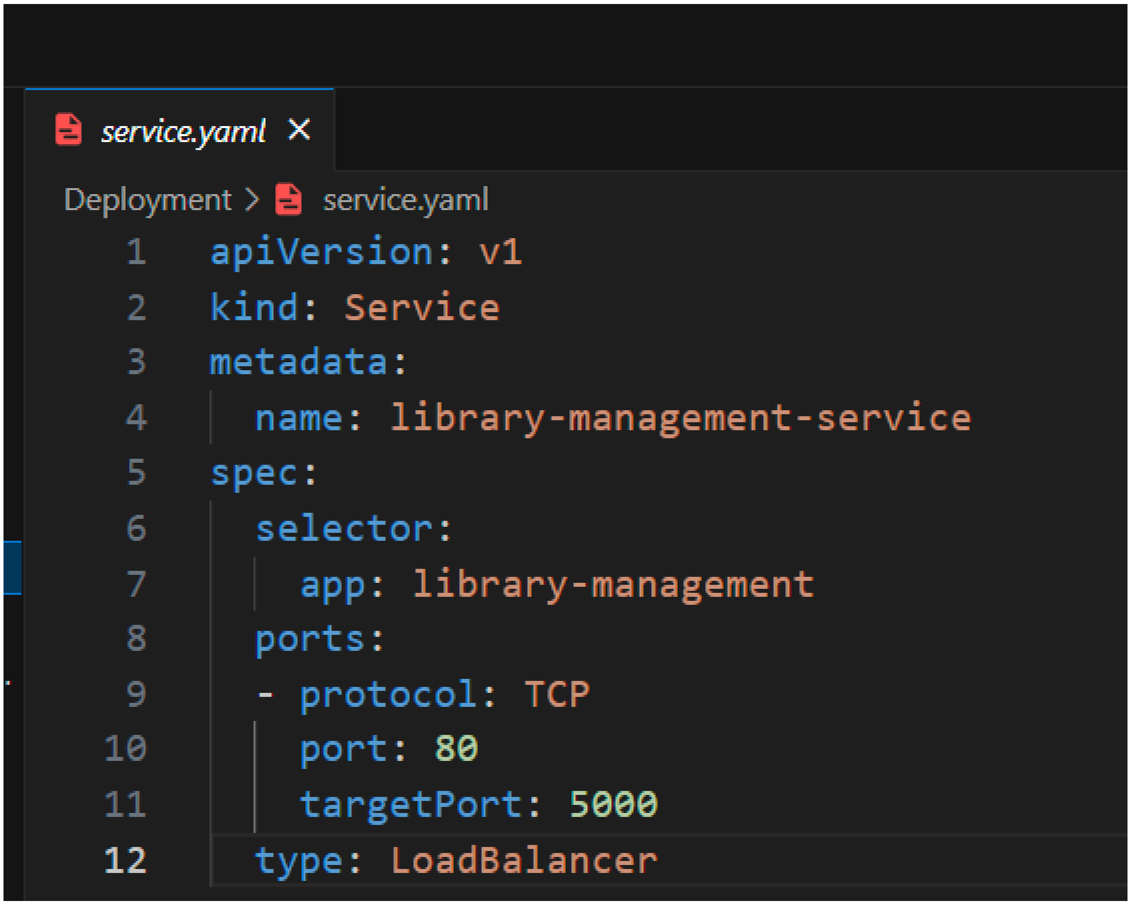
1- deployment.yaml file:


LibraryManagmen
library-management-deployment.yaml
Deployment > library-management-deployment.yaml

```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   # Ensures image is always pulled from the registry
          ports:
            - containerPort: 5000

```

2- Service.yaml file:



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

1. Deployment Configuration

(library-management-deployment.yaml)

apiVersion: 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. This name is used to manage and reference the deployment within the cluster.

spec:
replicas: 1 - Defines the number of pod replicas to be created. In this case, one instance of the application is deployed.

selector: This ensures the Deployment targets pods that match specific labels:

matchLabels:
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: Specifies the labels applied to each pod. In this case:
app: library-management

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. This image contains the library management system application.

imagePullPolicy: Always - Ensures the image is always pulled from the registry, regardless of whether it's cached locally.
Ports:

containerPort: 5000 - The port number inside the container where the application is accessible.

2. Service Configuration (service.yaml)

apiVersion: 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.

targetPort: 5000 - The port on the container where the application is running (as defined in the deployment).

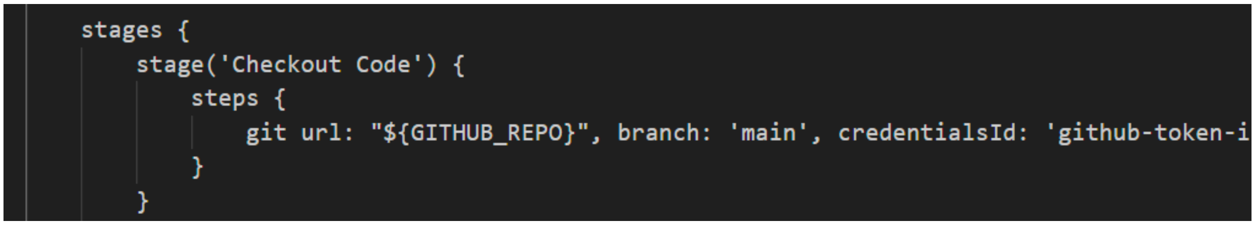
type: LoadBalancer - Specifies that this service is exposed to the internet via a load balancer, making it accessible outside the Kubernetes cluster.

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:

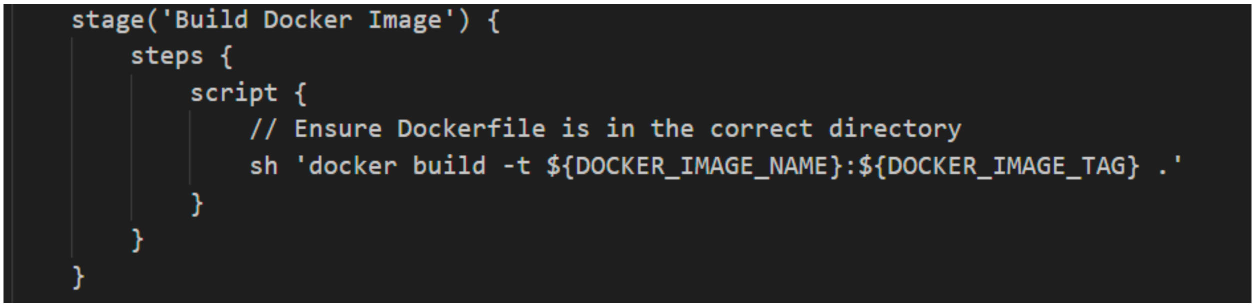
1. Checkout Code



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

- **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.

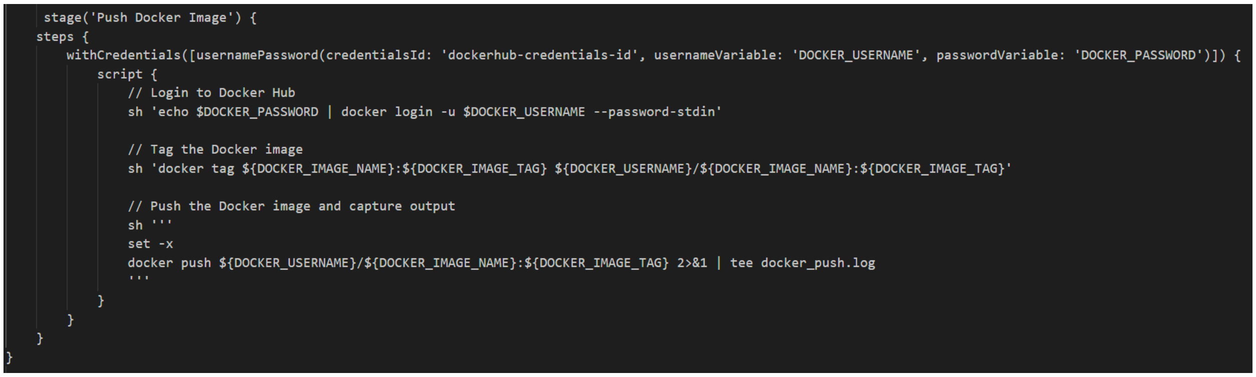
2- Build Docker Image:



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

- **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:



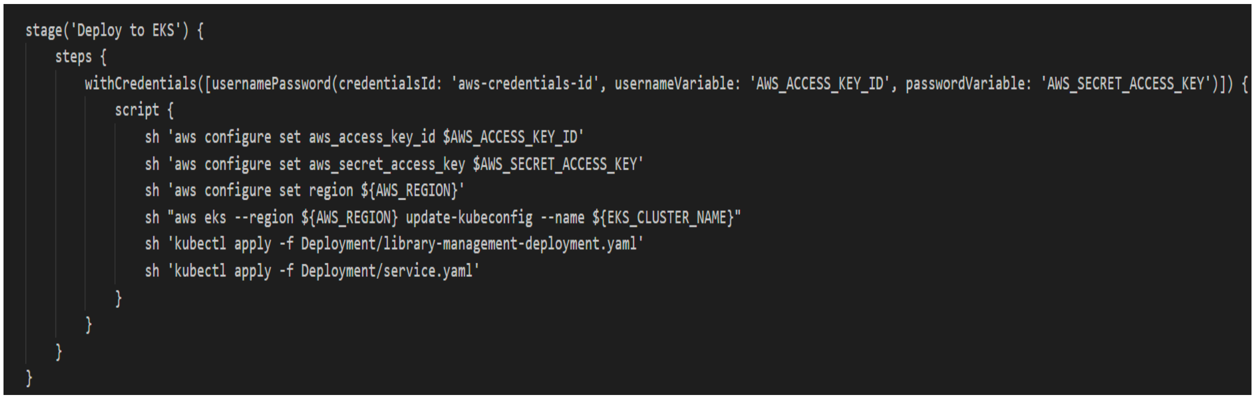
```
     stage('Push Docker Image') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials-id', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) {
            script {
                // Login to Docker Hub
                sh 'echo $DOCKER_PASSWORD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

- **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.

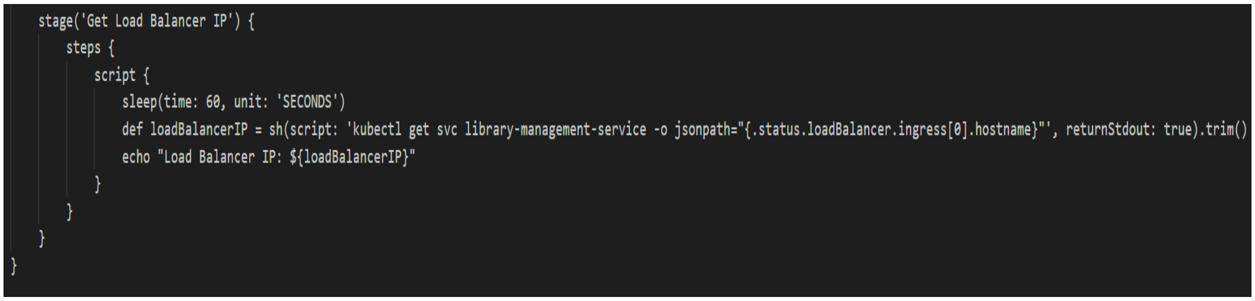
4- Deploy to EKS:



```
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'
            }
        }
    }
}
```

- **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}"
            }
        }
    }
}
```

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.