|  |  |  |
| --- | --- | --- |
|  | Books Library Project Documentation |  |
|  |  |  |
| Group D |  |  |

**Team members**

* Dana Mostafa
* Ahmed Hossam
* Laura Barsoum
* Mahmoud Magdy

**Final Project Objective**

This project comprehensively integrates application development, Dockerization, Kubernetes deployment, CI/CD automation, and infrastructure management with Terraform.

**Project Overview:**

Part 1: Application Development

Part 2: Dockerization

Part 3: Infrastructure as Code with Terraform

Part 4: Kubernetes Deployment on EKS

Part 5: CI/CD Pipeline Setup

**Part 1: Application Development**

First, we set up a Python Virtual Environment by using **python3 -m venv venv** and **source venv/bin/activate** then we create python flask application

A screenshot of a computer program

Description automatically generated

* **Flask:** The web framework used to create the application.
* **render\_template:** Renders HTML templates.
* **request:** Handles incoming request data.
* **redirect and url\_for:** Used for redirecting after form submissions.
* **flash:** Provides feedback messages to the user.
* **os and json:** Used for file handling and JSON data management.

A black background with white text

Description automatically generated

* **Flask(name)**: Initializes the Flask application.
* **secret\_key**: A secret key for session management and flash messages.

A black screen with white text

Description automatically generated

* **inventory:** A list to store the available books.
* **borrowed\_books:** A dictionary to store borrowed books with borrower information.
* **borrowed\_books\_path and inventory\_path:** Paths to the JSON files for persisting data.

**A computer screen with text on it

Description automatically generated**

Checks if the JSON files exist and creates them if they don't.

Loads the data from the JSON files into memory.

A black screen with colorful text

Description automatically generated

* **add\_book:** Adds a book to the inventory.
* **remove\_book:** Removes a book from the inventory by title.
* **search\_book:** Searches for a book by title.
* **borrow\_book:** Borrows a book by removing it from the inventory and adding it to the borrowed books.
* **return\_book:** Returns a borrowed book to the inventory.
* **save\_data:** Saves the current state of inventory and borrowed books to the JSON files.

A screen shot of a computer program

Description automatically generated

* **/@app.route('/')**: The home page that lists the current inventory.
* **/@app.route('/add', methods=['GET', 'POST'])**: Handles adding a book to the inventory.
* **/@app.route('/remove', methods=['GET', 'POST'])**: Handles removing a book from the inventory.
* **/@app.route('/search', methods=['GET', 'POST'])**: Handles searching for a book.
* **/@app.route('/borrow', methods=['GET', 'POST'])**: Handles borrowing a book.
* **/@app.route('/return', methods=['GET', 'POST'])**: Handles returning a borrowed book.

**Finaly Test the application locally run command python app.py:**

A black screen with white text

Description automatically generated

Then browse using <http://127.0.0.1:5000>

A screenshot of a library management system

Description automatically generated

**Part 2: Dockerization**

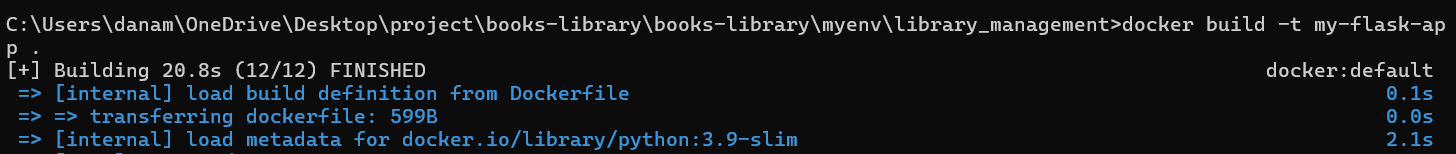
This Dockerfile creates a Docker image for a Python application using Flask

Uses the official Python 3.9 slim image from Docker Hub as the base image. This image is a minimal Python environment, which helps to keep the final image size small. Sets the working directory inside the container to /app. All subsequent commands will be run from this directory. Copies the requirements.txt file from your local directory into the container's working directory. Installs the Python dependencies listed in requirements.txt using pip. The --no-cache-dir option prevents caching of the installed packages, reducing the image size. Updates the package list and installs curl, a command-line tool for transferring data with URLs. Copies all files and directories from your local directory into the container's working directory. Informs Docker that the container will listen on port 5000 at runtime. This is the default port for Flask applications. Specifies the command to run the application when the container starts. It runs app.py using Python.

A screenshot of a computer program

Description automatically generated

* **Build the Docker Image:**

****

* **Run the Docker Container**A screen shot of a computer

  Description automatically generated**:**
* **A screenshot of a library management system

  Description automatically generated** **Push the Docker Image to Docker Hub**

**A screenshot of a computer

Description automatically generated**

**Part3: Infrastructure as Code with Terraform**

Terraform configuration defines an AWS infrastructure setup that includes a Virtual Private Cloud (VPC) and an Elastic Kubernetes Service (EKS) cluster.

**provider "aws"**: Configures the AWS provider with the region specified in the local variable region.

**locals: Defines local variables used throughout the configuration.**

* name: Identifier for the resources.
* region: AWS region to deploy the resources.
* vpc\_cidr: CIDR block for the VPC.
* azs: Availability zones to be used.
* public\_subnets, private\_subnets, intra\_subnets: CIDR blocks for different types of subnets.
* tags: Tags to be applied to resources.

**module "vpc": Configures a VPC using the terraform-aws-modules/vpc/aws module.**

* name: Name of the VPC.
* cidr: CIDR block for the VPC.
* azs: Availability zones for the VPC.
* private\_subnets, public\_subnets, intra\_subnets: Subnet CIDR blocks.
* enable\_nat\_gateway: Enables a NAT gateway for internet access from private subnets.
* public\_subnet\_tags, private\_subnet\_tags: Tags for public and private subnets.

**module "eks": Configures an EKS cluster using the terraform-aws-modules/eks/aws module.**

* cluster\_name: Name of the EKS cluster.
* cluster\_endpoint\_public\_access: Allows public access to the EKS cluster endpoint.
* cluster\_addons: Configures add-ons for CoreDNS, kube-proxy, and VPC CNI.
* vpc\_id: VPC ID where the EKS cluster will be deployed.
* subnet\_ids: Subnet IDs for private subnets.
* control\_plane\_subnet\_ids: Subnet IDs for control plane (intra) subnets.
* eks\_managed\_node\_group\_defaults: Defaults for managed node groups including AMI type and instance types.
* eks\_managed\_node\_groups: Configures a managed node group named Team4-cluster-wg with spot instances and specific tags.
* tags: Tags to be applied to the EKS cluster resources.

Used for start terraform: terraform init

For apply: terraform apply

**Part4: Kubernetes Deployment on EKS**

In Kubernetes, we used these components and objects to manage and orchestrate our containerized application. Here’s a brief explanation of each:

**1. Deployment**

A Deployment manages a set of identical pods, ensuring the desired number of replicas are running and updating them as needed.

**2. Ingress**

An Ingress is a resource that manages external access to services within a cluster, typically HTTP and HTTPS. It can provide load balancing, SSL termination, and name-based virtual hosting.

**3. Persistent Volume (PV)**

A Persistent Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically using Storage Classes. It abstracts the details of how storage is provided.

**4. Persistent Volume Claim (PVC)**

A Persistent Volume Claim (PVC) is a request for storage by a user. It specifies the size and access modes needed for the storage, and Kubernetes binds it to an appropriate PV.

**5. Service**

A Service is an abstraction that defines a logical set of pods and a policy by which to access them. It provides a stable IP address and DNS name to access the pods, even if the underlying pods change.

**6. Storage Class**

A Storage Class defines the different types of storage available in a cluster. It allows dynamic provisioning of storage resources and can specify parameters such as performance characteristics or replication policies.

A screenshot of a computer

Description automatically generated

**Part 5: CI/CD Pipeline Setup**

We create a Jenkinsfile to automate the process of building a Docker image, pushing it to Docker Hub, and deploying it to an AWS EKS cluster.

**Stages**

**Clone Repository**

**Purpose**: Clones the GitHub repository containing the application code.

**Credentials**: Uses github-cred for authentication.

**Build Docker Image**

**Purpose**: Builds a Docker image from the Dockerfile located at books- library/myenv/library\_management/Dockerfile.

**Image Name**: ahmedhosssam/flask-app.

**Push Docker Image to Docker Hub**

**Purpose**: Pushes the built Docker image to Docker Hub.

**Credentials**: Uses ec6825c0-99e6-4970-a914-0d16ad8cd8e9 for Docker Hub authentication.

**Deploy to EKS**

**Purpose**: Deploys the Docker image to an EKS cluster.

**AWS Credentials**: Uses aws\_creeed for AWS CLI access.

**Kubeconfig Path**: Location of the kubeconfig file.

**Commands**:

* + - Configures AWS CLI with the specified credentials and region.
    - Updates kubeconfig for EKS.
    - Uses kubectl to update the image for the flask-app-deployment in EKS.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated