**Task 1: Develop the Python API Server**

Outline

- Part 1 - Write an application with MySQL and database implementation on GitHub Repo

- Part 2 - Install Python and Flask framework on Ubuntu server (I used AWS ec2 ubuntu)

##Part 1 - Write a Sample Web Application with MySQL and database implementation on GitHub Repo

- Write an application with database implementation using `MySQL` and save the complete code as `app-with-mysql.py` under ` task1-flaskapp` folder. Normally there should be a database in here.However, I want to show the database in task2 and task3.Because the pyhton code is the same, so you will see it works.

- configure required environmental variables for MySQL.

- Create users table within MySQL db and populate with sample data

Execute the code below only once.

Write sql code for initializing users table.

- Write a function named `find\_emails` which find emails using keyword from the user table in the db,

and returns result as tuples `(name, email)`.

# Write a function named `find\_emails` which find emails using keyword from the user table in the db,

# and returns result as tuples `(name, email)`.

def find\_emails(keyword):

    query = f"""

    SELECT \* FROM users WHERE username like '%{keyword}%';

    """

    cursor.execute(query)

    result = cursor.fetchall()

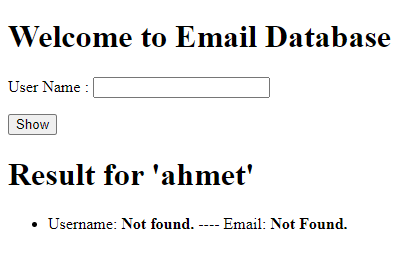
    user\_emails = [(row[0], row[1]) for row in result]

    # if there is no user with given name in the db, then give warning

    if not any(user\_emails):

        user\_emails = [('Not found.', 'Not Found.')]

    return user\_emails



- Write a function named `insert\_email` which adds new email to users table the db.

# Write a function named `insert\_email` which adds new email to users table the db.

def insert\_email(name, email):

    query = f"""

    SELECT \* FROM users WHERE username like '{name}';

    """

    cursor.execute(query)

    result = cursor.fetchall()

    # default text

    response = ''

    # if user input are None (null) give warning

    if len(name) == 0 or len(email) == 0:

        response = 'Username or email can not be empty!!'

    # if there is no same user name in the db, then insert the new one

    elif not any(result):

        insert = f"""

        INSERT INTO users

        VALUES ('{name}', '{email}');

        """

        cursor.execute(insert)

        response = f'User {name} and {email} have been added successfully'

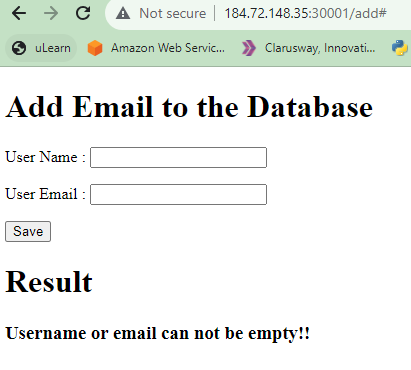
    # if there is user with same name, then give warning

    else:

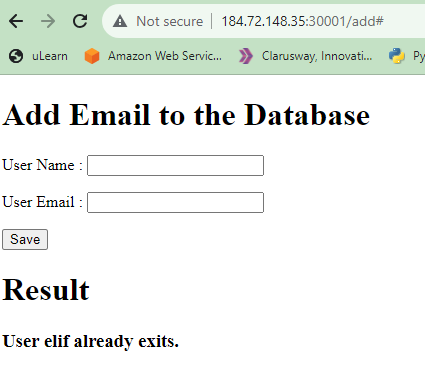
        response = f'User {name} already exists.'

    return response

Here, I applied as empty and we can see the result message.



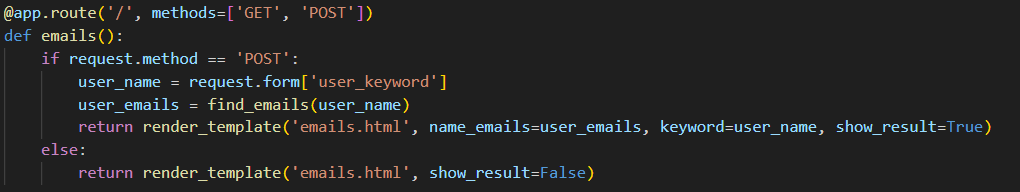
Here, I tried to add a name and which is already exist, we can see the result message.



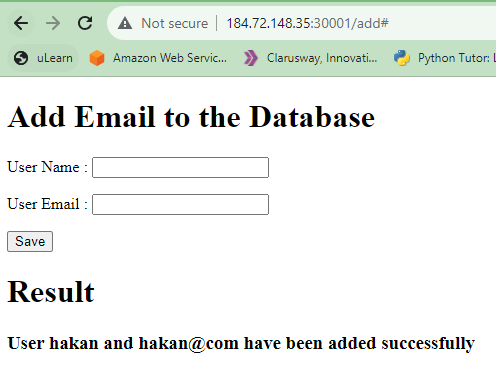
- Write a function named `emails` which finds email addresses by keyword using `GET` and `POST` methods,

using template files named `emails.html` given under `templates` folder

and assign to the static route of ('/')



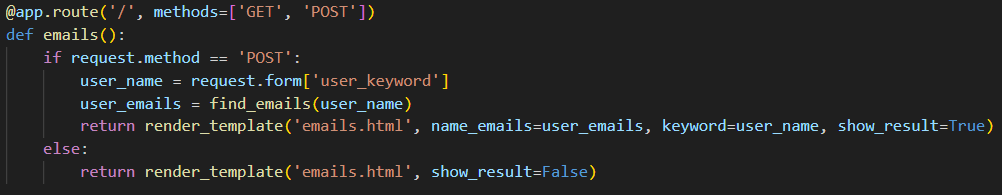
Here, we can see the add page (/add) and I can add a new user.



- Write a function named `emails` which finds email addresses by keyword using `GET` and `POST` methods,

- using template files named `emails.html` given under `templates` folder

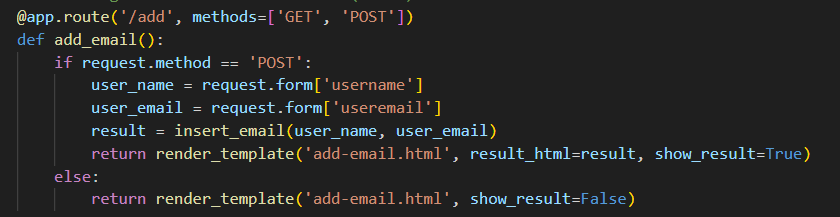
- and assign to the static route of ('/')



- Write a function named `add\_email` which inserts new email to the database using `GET` and `POST` methods,

- using template files named `add-email.html` given under `templates` folder

- and assign to the static route of ('add')



- Add a statement to run the Flask application which can be reached from any host on port 80.



## Part 2 - Install Python and Flask framework on Amazon Linux 2 EC2 Instance

- Launch an Amazon EC2 instance using the Amazon Linux 2 AMI with security group allowing SSH (Port 22) and HTTP (Port 80) connections.

- Connect to your instance with SSH.

- Update the installed packages and package cache on your instance.

- Install `Python 3` packages.

- Check the python3 version

- Install `Python 3 Flask` framework.

- Install `flask\_mysql`.

- Run application with Python

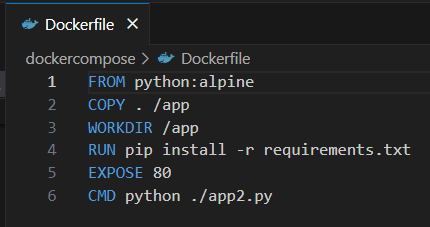
**Task 2: Dockerize Your Application**

In this task, I will containerise my application using Docker.The goal is to package your API server into a Docker container, ensuring it runs consistently across different platforms.The purpose of Dockerization is to achieve consistent, portable, and efficient deployment of applications by encapsulating them and their dependencies within lightweight, isolated containers.Docker makes it easy to scale applications horizontally by running multiple instances of containers.

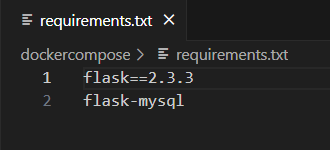
Docker containers can run on any system that supports Docker, providing portability and eliminating the "it works on my machine" issue. This makes it easier to deploy applications across different infrastructure or cloud providers.

Docker supports a fast development cycle and continuous integration/continuous deployment (CI/CD) practices. Developers can quickly build, test, and deploy applications, leading to shorter development and release cycles.Docker is well-suited for microservices architecture, where applications are composed of small, independently deployable services. Each microservice can run in its own container, enabling modular development and deployment.

**Part 1: Dockerfile Creation**



Part 2: Dependency Files:



Part 3 : Build the Docker Image:

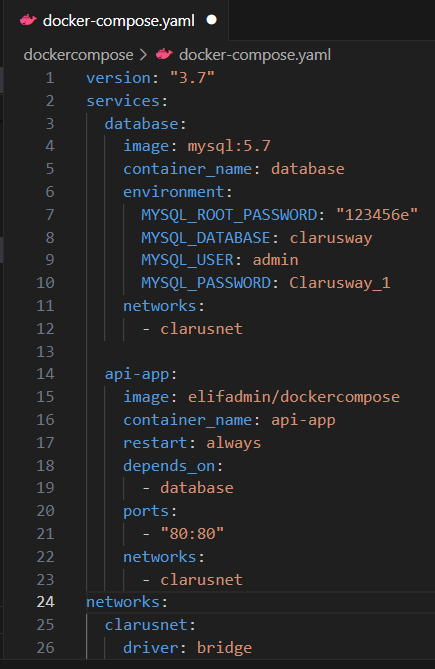
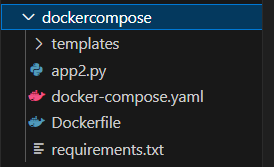


Build the image.

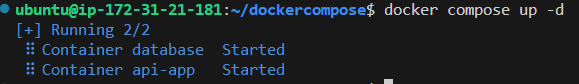
Part 3: Run the Docker Container:

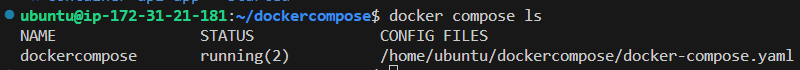
We need to run 2 containers, so we create docker-compose yaml.

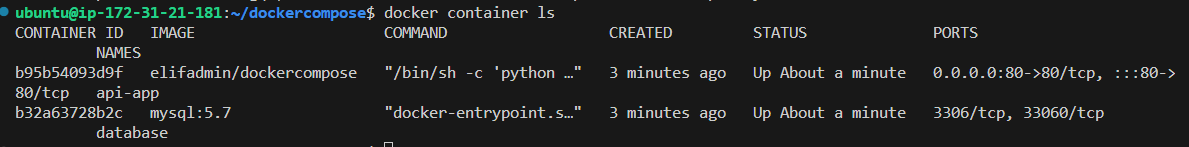
Dockercompose structure

And then we run the containers by docker-compose.yaml

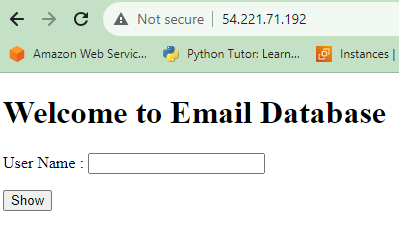
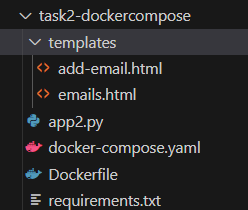


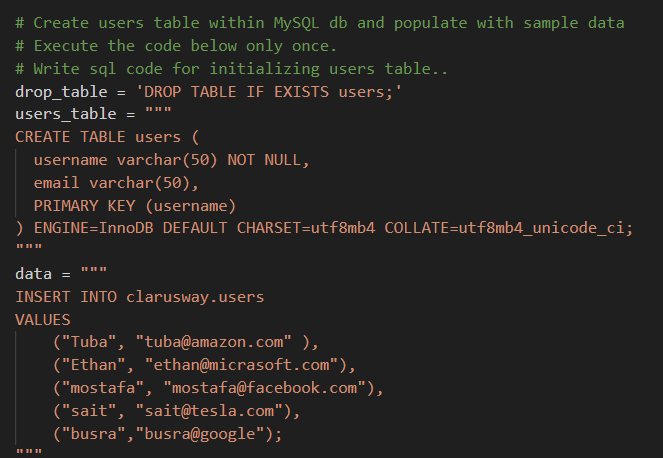
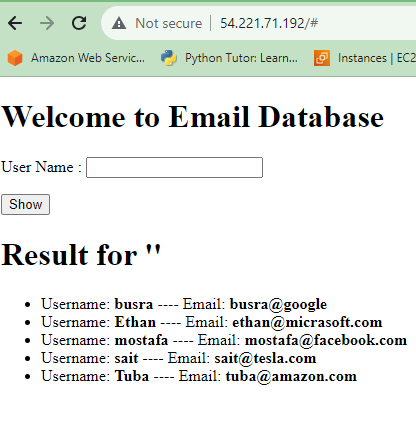




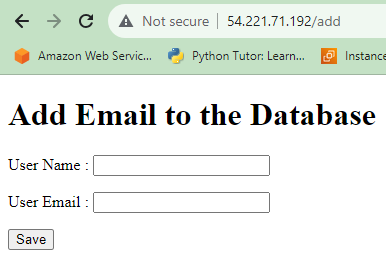
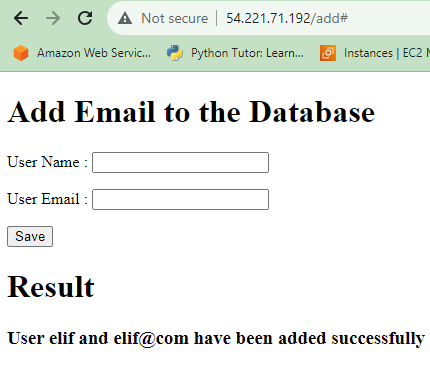
**Part 4: Access the API:**

I designed the structure as two menu. The first one is emails.html page.When I enter the ec2 public IP, I can show the main page.When I click show button, I can see the emails and names which are in the database.

Now, I want to show you the second page.

**Task 3: Kubernetes Deployment with Manifests**

Deploy your Dockerized Python API server on a local Kubernetes cluster using Kubernetes

**Part 1: Install Kubernetes on AWS ec2.**

**Part 2:**

1.1. Create/add Deployment

Deployment definition file should configure create/add operations with one replica.

* Expose the container port on `port 80` .
* Deployment definition file should set the proper Environmental Variables for the db connection.

1.2. Create/add Service

This service should be attached to `create/add deployment` .

* Service type should be Nodeport published on `port:30001`.
* Expose the port and target port on port `80` .

2.1. Database deployment

- Deployment should use mysql:5.7 image pulled from Docker hub.

- Expose the container port on `port 3306`.

- Deployment definition file should set the proper Environmental Variables.

2.2. Database Service

- This service should be attached to `DATABASE Deployment`.

- Service type should be ClusterIP.

- Expose the port and target port on port `3306`.

3.1. Kubernetes Environment

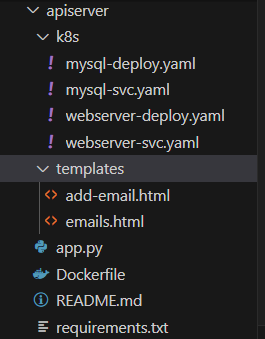
- Assign two EC2 machines as the project's infrastructure. One should be configured as the master and the other should be configured as the worker.

- Minimum `t2.medium` instance type should be selected.

- The Web Application should be accessible via web browser from anywhere.

- The Application files should be downloaded from Github repo and deployed on EC2 Instance.

Project skeleton



To do

1- CREATE/ADD DEPLOYMENT AND SERVICE

1. Dockerfile

2. webserver\_deploy.yaml

3. webserver\_svc.yaml

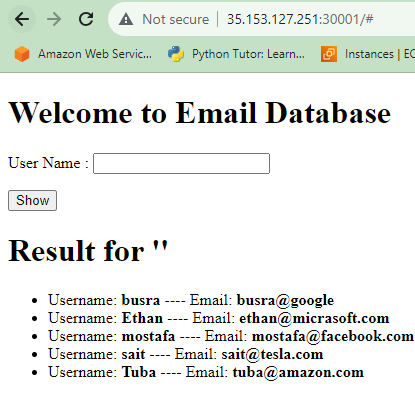
2- DATABASE DEPLOYMENT AND SERVICE

1. mysql-deploy.yaml

2. mysql-svc.yaml

**Part 3: Access the API:**

It is the same as Task2 part4.



**Task 4: CI/CD Pipeline Setup with GitHub Actions**

I set up a basic CI/CD pipeline for our Python API server using GitHub Actions.

**GitHub Actions Workflow**

This GitHub Actions workflow automates the CI/CD process by orchestrating Docker image building, Kubernetes configuration, and deployment to an EC2 instance. The workflow is triggered on pushes to the main branch, ensuring that changes are automatically deployed to the Kubernetes cluster.

**GitHub Actions workflow**

name: Deploy to EC2 Kubernetes

on: #main branchinda commit yapilinca cicd calisir

push:

branches:

- main

jobs:

build\_and\_deploy: #This is the name of the job.

runs-on: ubuntu-latest #gitup actions create a container which is ubuntu-latest.

steps:

- name: Checkout repository #This block checks out the latest version of the source code from your GitHub repository.

uses: actions/checkout@v2

- name: Configure Docker credentials #This block configures Docker credentials for authenticating with Docker Hub. It allows the workflow to securely push Docker images to your Docker Hub account.settings-secret-actions

uses: docker/login-action@v1

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

- name: Build and push Docker image #This block builds a Docker image from your application's source code and pushes it to Docker Hub. It ensures that the latest version of your application is available as a Docker image for deployment.

run: |

docker build -t elifadmin/webserver -f ./apiserver/Dockerfile .

docker push elifadmin/webserver

- name: Install kubectl # This block installs kubectl, the Kubernetes command-line tool, on the GitHub Actions runner. It is necessary for interacting with the Kubernetes cluster during the deployment process.

run: |

sudo apt-get update

sudo apt-get install -y apt-transport-https gnupg2 curl

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubectl

- name: Configure Kubernetes credentials #This block configures Kubernetes credentials by decoding the base64-encoded Kubernetes configuration data from GitHub Secrets. It sets up the necessary Kubernetes context for subsequent kubectl commands.

run: |

echo "${{ secrets.KUBE\_CONFIG\_DATA }}" | base64 --decode > kubeconfig.yaml

kubectl config set-context kubernetes-admin@kubernetes --cluster=$(kubectl config view -o jsonpath='{.contexts[0].context.cluster}') --user=$(kubectl config view -o jsonpath='{.contexts[0].context.user}')

- name: Install AWS CLI #This block installs the AWS Command Line Interface (AWS CLI) on the GitHub Actions runner. While not explicitly used in the workflow, it may be useful for future AWS-related tasks.

run: |

sudo apt-get update

sudo apt-get install -y awscli

- name: Configure AWS credentials #This block configures AWS credentials on the GitHub Actions runner, allowing AWS CLI commands to be executed if needed in the future.

run: |

aws configure set aws\_access\_key\_id ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws configure set aws\_secret\_access\_key ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws configure set default.region us-east-1

- name: SSH into EC2 and deploy #This block is the final step where the workflow SSHes into the EC2 instance. It uses the SSH private key, hostname, and username from GitHub Secrets to connect to the EC2 instance. The kubectl set image command updates the Kubernetes deployment with the latest Docker image, triggering the application deployment.

env:

PRIVATE\_KEY: ${{ secrets.SSH\_PRIVATE\_KEY }} #keypem

HOSTNAME: ${{secrets.SSH\_HOST}} #ec2 user

USER\_NAME: ${{secrets.USER\_NAME}} #ubuntu

run: | #connect with ssh for running commands

echo "$PRIVATE\_KEY" > private\_key && chmod 600 private\_key

ssh -o StrictHostKeyChecking=no -i private\_key ${USER\_NAME}@${HOSTNAME} 'kubectl delete -f /home/ubuntu/apiserver/k8s/ && kubectl apply -f /home/ubuntu/apiserver/k8s/' #update deployment

#ssh -o StrictHostKeyChecking=no -i private\_key ${USER\_NAME}@${HOSTNAME} connect with ssh to ec2 server.

#'kubectl delete -f /home/ubuntu/apiserver/k8s/ && kubectl apply -f /home/ubuntu/apiserver/k8s/' = updates the Kubernetes deployment with the latest Docker image