Project Title:

Health Calculator Microservice with CI/CD Pipeline on Azure

Objective:

To develop a Python-based microservice that calculates health metrics (BMI and BMR) using a REST API. The project will be containerized with Docker, managed with Makefile, and deployed to Azure using GitHub Actions for CI/CD.

Mathematical Equations for Health Calculations

1. **Body Mass Index (BMI)**:

BMI = weight (kg)/(height (m))^2

2. **Basal Metabolic Rate (BMR)** (Harris-Benedict Equation):
    - For **males**:

    BMR = 88.362 + (13.397 x weight (kg)) + (4.799 x height (cm)) - (5.677 x age (years))

    - For **females**:

      BMR = 447.593 + (9.247 x weight (kg)) + (3.098 x height (cm)) - (4.330 x age (years))

Project Requirements:

1. **Python Microservice**:
    - Develop a Flask REST API with endpoints:
        - `/bmi`: Calculates BMI using `height` (meters) and `weight` (kg).
        - `/bmr`: Calculates BMR using `height` (cm), `weight` (kg), `age`, and `gender`.
2. **Containerization with Docker**:
    - Create a `Dockerfile` to containerize the application.
3. **Orchestration with Makefile**:
    - Automate setup, testing, and deployment with Makefile commands:
        - `make init`, `make run`, `make test`, `make build`.
4. **Dependency Management**:
    - Manage dependencies in `requirements.txt`.
5. **Testing**:
    - Write unit tests to validate the BMI and BMR calculations and API endpoints.
6. **CI/CD Pipeline with GitHub Actions**:
    - Set up a pipeline to automate testing and deployment on code push.
7. **Deployment to Azure**:
    - Use Azure App Service to host the containerized microservice.

Detailed Project Instructions

1. Microservice Development

1. **Create a directory** named `health-calculator-service`.
2. Inside `health-calculator-service`, create the following files:
    - **`app.py`**:
        - Define the Flask API with two endpoints (`/bmi` and `/bmr`).
    - **`health_utils.py`**:
        - Define utility functions `calculate_bmi` and `calculate_bmr`.

Example Code:

- **app.py**

```
app = Flask(__name__)

@app.route('/bmi', methods=['POST'])
def bmi():
    # here goes the code

@app.route('/bmr', methods=['POST'])
def bmr():
    # here goes the code

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- **health_utils.py**

```
def calculate_bmi(height, weight):
    """Calculate Body Mass Index (BMI) given height in meters and weight in kilograms."""
    return BMI

def calculate_bmr(height, weight, age, gender):
    """Calculate Basal Metabolic Rate (BMR) using the Harris-Benedict equation."""
```

2. Containerization with Docker

- Create a **Dockerfile** in the `health-calculator-service` directory to containerize the application.

Example Dockerfile:

```
FROM python:3.9-slim

##
```

3. Orchestration with Makefile

- Create a **Makefile** in the `health-calculator-service` directory to automate tasks.

Example Makefile:

```
include .env
export

.PHONY: init run test build clean

init:
    @echo "Installing dependencies..."

run:
    @echo "Running the Flask app..."

test:
    @echo "Running tests..."

build:
    @echo "Building the Docker image..."
```

4. Dependency Management

- Create a **requirements.txt** file with dependencies.

Example requirements.txt:

```
Flask==2.0.2
```

5. Testing

- Create a **test.py** file to validate BMI and BMR calculations.

Example test.py:

```
import unittest
from health_utils import calculate_bmi

class TestHealthUtils(unittest.TestCase):

    def test_calculate_bmi(self):
        self.assertAlmostEqual(calculate_bmi(1.75, 70), 22.86, places=2)


if __name__ == '__main__':
    unittest.main()
```

6. CI/CD Pipeline with GitHub Actions

- In `.github/workflows`, create `ci-cd.yml` for GitHub Actions to automate testing and deployment.

Example ci-cd.yml:

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build-test-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'

    - name: Install dependencies
      run: | # command


    - name: Run tests
      run: | # command

    - name: Build Docker image
      run: | # command

    - name: Deploy to Azure Web App
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'health-calculator-app' # Replace with your Azure app name
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        package: ./health-calculator-service
```

7. Deployment to Azure

1. **Set up Azure App Service**:
    - Create a new Web App in Azure App Service to host the application.
2. **Add Publish Profile to GitHub Secrets**:
    - Download the **publish profile** from Azure App Service.
    - In GitHub repository settings, add a secret called `AZURE_WEBAPP_PUBLISH_PROFILE` and paste the profile contents.

3. **Trigger Deployment**:
    - Push code changes to the `main` branch to trigger the CI/CD pipeline and deploy to Azure.

Expected Deliverables:

1. A GitHub repository with:
    - The Flask microservice code (`app.py` , `health_utils.py`).
    - Unit tests (`test.py`).
    - Dockerfile.
    - Makefile.
    - requirements.txt.
    - GitHub Actions CI/CD pipeline (`.github/workflows/ci-cd.yml`).
2. A deployed instance of the microservice on Azure App Service, accessible via an HTTP endpoint.

Evaluation Criteria:

- **Founctionality of the API (2)**: The API can be tested remotely
- **Correctness (1)**: Both endpoints (`/bmi` and `/bmr`) return accurate calculations.
- **Containerization (2)**: The application is correctly containerized and runs in Docker.
- **Automation (4)**: The Makefile commands work as expected to set up dependencies, test, run, and build the container.
- **CI/CD Pipeline (3)**: The GitHub Actions workflow successfully automates testing and deployment.
- **Documentation and Readability (4)**: Code and comments are clear and well-organized.
- **Innovation (2)**: integrate additional functionalities
- **Time Respect (2)**