Deployment Documentation for the Application

# Overview

This document describes the process of deploying the Flask application for resume and job description analysis using Docker. We have build an individual microservice for each module which can be run separately using individual so

# Prerequisites

* Docker installed on the deployment machine.
* Access to the application's source code and Dockerfile.

# 

# Containerization with Docker

* Install Docker: Ensure Docker is installed on your machine. If not, download and install it from Docker's official website.
* Create a Dockerfile: In the root directory of your Flask application, create a file named Dockerfile (no file extension). [Docker file is created in the uploaded folder]
* Create a requirements.txt File:This file should list all the Python packages that your application depends on. You can generate this file using pip freeze > requirements.txt if you're already running your app in a virtual environment. [Already created]

# Building the Docker Image

* Navigate to the Application Directory: Ensure you are in the directory containing the Dockerfile and the application's source code.
* Build the Image:
  + Run the following command to build the Docker image: docker build -t resume-parser-app .
  + This command creates a Docker image named resume-parser-app based on the instructions in your Dockerfile.
* Running the Container
* Start the Container:
  + Use the following command to run your application in a container: docker run -d -p 4001:4001 --name resume-parser resume-parser-app
  + This command starts a container named resume-parser using the resume-parser-app image.
  + It maps port 4001 of the container to port 4001 on the host, allowing external access to the application.
* Verify Container Status:
  + Check the status of the container with: docker ps
* Ensure that the resume-parser container is listed and running.

# Accessing the Application

* The Flask application is now accessible at http://localhost:4001 (or the corresponding host address if deployed on a remote server).

# Deployment

* Choose a Deployment Platform:Popular choices include cloud providers like AWS, Azure, Google Cloud, or Heroku, or container orchestration systems like Kubernetes.
* Deploying on a Cloud Provider (e.g., AWS):
  + AWS Elastic Container Service (ECS): You can use ECS to run Docker containers on AWS.
  + Create an ECS task definition, where you specify the Docker image to use (the one you pushed to Docker Hub), CPU and memory requirements, networking settings, and other configurations.
  + Run the task to start your containerized application.
* Deploying with Kubernetes:
  + If you're using Kubernetes, you'll need to create deployment and service YAML files for your application.
  + Apply these configurations to your Kubernetes cluster to deploy your application.
* Continuous Deployment:
  + Consider setting up a CI/CD pipeline using tools like Jenkins, GitLab CI, or GitHub Actions to automate the deployment process.