IBM-CAD

CONTAINARIZATION OF THE APP

TEAM LEAD : DHANA LAKSHMI S

TEAM M1 : KALPANA SRI R

TEAM M2 : JEEVITHA V

DEPARTMENT OF CSE

KARPAGAM INSTITUTE OF TECHNOLOGY,COIMBATORE

To containerize the web application using Docker, you can follow these steps:

1. Create a `Dockerfile`:

Open a text editor and create a file called `Dockerfile` in the root directory of our web application. Add the following content to the `Dockerfile`:

dockerfile

# Use a base image

FROM python:3.9

# Set the working directory

WORKDIR /app

# Copy the application files to the container

COPY . /app

# Install dependencies

RUN pip install -r requirements.txt

# Expose the desired port

EXPOSE 8080

# Define the command to run the application

CMD [ "python", "app.py" ]

This `Dockerfile` specifies the base image, sets the working directory inside the container, copies the application files to the container, installs the required dependencies, exposes port 8080 (adjust if needed), and defines the command to run the application.

2. Build the Docker image:

Open a terminal or command prompt, navigate to the directory containing the `Dockerfile`, and run the following command to build the Docker image:

docker build -t your-image-name:tag .

Replace `your-image-name` with the desired name for your Docker image and `tag` with a version or tag you want to assign to the image. The `.` at the end specifies the current directory as the build context.

For example:

docker build -t web-app:1.0 .

This command will build the Docker image using the `Dockerfile` and the application files in the current directory.

3. Verify the Docker image:

Run the following command to verify that the Docker image was successfully built:

docker images

You should see the newly created Docker image listed with the provided image name and tag.

4. Run the Docker container:

Now, We can run a Docker container using the built image. Execute the following command to start a container.

docker run -d -p 8080:8080 your-image-name:tag

Replace `your-image-name` and `tag` with the respective values you used during the image build step.

For example:

docker run -d -p 8080:8080 web-app:1.0

This command runs the Docker container in detached mode (`-d`), maps port 8080 of the host to port 8080 of the container (`-p`), and uses the specified Docker image.

Our web application should now be accessible at `http://localhost:8080` in our browser.

docker run -d -p 8080:8080 -e VARIABLE\_NAME=VALUE your-image-name:tag

Remember to replace `VARIABLE\_NAME` and `VALUE` with the appropriate values for our application.

**DOCKER FILE**

# Use a base image

FROM python:3.9

# Set the working directory

WORKDIR /app

# Copy the application files to the container

COPY . /app

# Install dependencies

RUN pip install -r requirements.txt

# Expose the desired port

EXPOSE 8080

# Define the command to run the application

CMD [ "python", "app.py" ]

**DEPLOYMENT YAML FOR KUBERNETES**

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app-deployment

labels:

app: web-app

spec:

replicas: 3

selector:

matchLabels:

app: web-app

template:

metadata:

labels:

app: web-app

spec:

containers:

- name: web-app-container

image: registry.example.com/your-image:latest

ports:

- containerPort: 8080

**SERVICE YAML FOR KUBERNETES**

apiVersion: v1

kind: Service

metadata:

name: web-app-service

spec:

selector:

app: web-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

type: LoadBalancer

**PYTHON CODE TO CONNECT TO IBM CLOUD AND DATABASE**

import mysql.connector

from getpass import getpass

# Get the database credentials from the environment or securely prompt the user

db\_host = getpass("Enter the database host:")

db\_username = getpass("Enter the database username:")

db\_password = getpass("Enter the database password:")

# Connect to the MySQL database

cnx = mysql.connector.connect(

host=db\_host,

user=db\_username,

password=db\_password,

database="your\_database"

)

# ...

# Perform database operations (insert, select, update, delete)

# ...

# Close the database connection

cnx.close()