CS353 HW4 Tutorial

Description

- In this assignment, you are going to implement a generic Todo-list application using Python, Flask, MySQL, and Docker Compose.
- To aid you in the coding process, we give you a sample application that uses Python, Flask, MySQL and Docker Compose.

Logistics

Docker

- Docker is a virtualization platform that helps developers to easily create, deploy, and run applications inside containers.
- Containers provide a consistent and isolated environment for applications, ensuring that the applications can be replicated across different environments.

Docker Compose

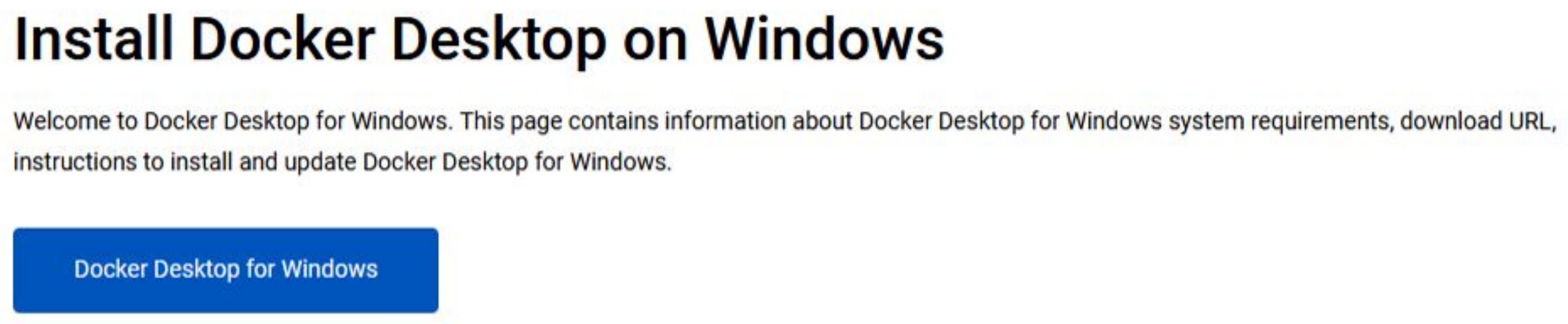
- Docker Compose is a tool for defining and running multi-container Docker applications.
- You can define the services (in our case, web and db services) that make up your application in a YAML file, and then start and stop all services with a single command (We will show you how).

Flask

- Flask is a micro web framework written in Python.
- It is designed to be simple and lightweight,
  allowing quick implementation and deployment of web applications
  with minimal setup.

Installing Docker Compose on Windows

- On Windows, you can use the following link to obtain the installer:
  - https://docs.docker.com/desktop/install/windows-install/
  - It will install both Docker and Docker Compose for you.



Install Docker Desktop on Windows

Welcome to Docker Desktop for Windows. This page contains information about Docker Desktop for Windows system requirements, download URL, instructions to install and update Docker Desktop for Windows.

Installing Docker Compose on Ubuntu Linux

- In Ubuntu, you can use the following link to install Docker Compose.
  - https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04

- You need Docker present in your machine. For that use:
  - https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04

Testing your Docker installation

- On Windows, open up a command line using **Windows + R** and entering **cmd.** (Or simply use the search box for **cmd**).
- On Linux, open up a terminal.
- Simply input
  - `docker run hello-world`
- If your installation was successfull, the output should be similar to this. Otherwise, fix your installation.

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:b8ba256769a0ac28dd126d584e0a2011cd2877f3f76e093a7ae560f2a5301c00
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

- If you get the following, you need to start the Docker service

```
docker: error during connect: This error may indicate that the docker daemon is not running.: Post
"http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.24/containers/create": open //./pipe/docker_engine: The
system cannot find the file specified.
See 'docker run --help'.
```

Sample Application

Application File Structure

- Simply extract the archive file we give you to obtain the following file structure. *You don’t need to change anything.*

```
├── app
│   ├── app.py
│   └── templates
│       ├── login.html
│       └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

```
├── app
│   ├── app.py
│   └── templates
│       ├── login.html
│       └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

- Application Directory app/
- Contains the main application file app.py
- HTML templates rendered by app.py

```
├── app
│    ├── app.py
│    └── templates
│         ├── login.html
│         └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

- Defines and combines the web and database services

```
├── app
│   ├── app.py
│   └── templates
│       ├── login.html
│       └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

- Defines how to web service is constructed, what are its requirements, which application files are going to be run

```
├── app
│   ├── app.py
│   └── templates
│       ├── login.html
│       └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

- Lists the required Python packages

```
├── app
│   ├── app.py
│   └── templates
│       ├── login.html
│       └── register.html
├── docker-compose.yaml
├── Dockerfile
├── requirements.txt
└── schema.sql
```

- The SQL Database Schema of the application.
- Contains the tables. You can insert any SQL Statement here.
- Used to create and populate the database when the services are started automatically (Defined in docker-compose.yaml file)

Dockerfile

- The Dockerfile below uses an existing python image and adds the python modules defined in the requirements.txt file, which contains two modules Flask and flask_mysqldb.
- It also points at the /app folder.

```
FROM python:3.9-slim-buster
RUN apt-get update
RUN apt-get install -y gcc
RUN apt-get install -y default-libmysqlclient-dev
WORKDIR /appCOPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
```

Building Your Container

- In the command line navigate to the folder where the Dockerfile resides.
- Build your own docker image tagged **cs353hw4app** using:

```
docker build –t cs353hw4app .
```

- Once successful, you can list your images using

```
docker images
```

- You should see your image like the following:

```
REPOSITORY     TAG        IMAGE ID        CREATED          SIZE
cs353hw4app    latest     603579517ecc    2 minutes ago    282MB
```

docker-compose.yaml

```
version: '3'
services:
  web:
    image: cs353hw4app
    ports:
      - "5000:5000"
    volumes:
      - ./app:/app
    working_dir: /app
    command: python app.py
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: cs353hw4db
    ports:
      - "3307:3306"
    volumes:
      -
./schema.sql:/docker-entrypoint-initdb.d/schema.sql
```

Docker image name we just built

Web service port. We can reach our service at http://localhost:5000

app folder in the docker image mapped to app folder in our computer

running the app.py to start the service

Mysql root password and database name

Mapped schema.sql file. You need to prepare this file.

Starting the services

- Before running with this configuration, ***you need to prepare your schema.sql file*** that contains your database creation, table creation, and insertion statements.
- Once you prepare your database schema, you can fire up the system using

```
docker-compose up -d.
```

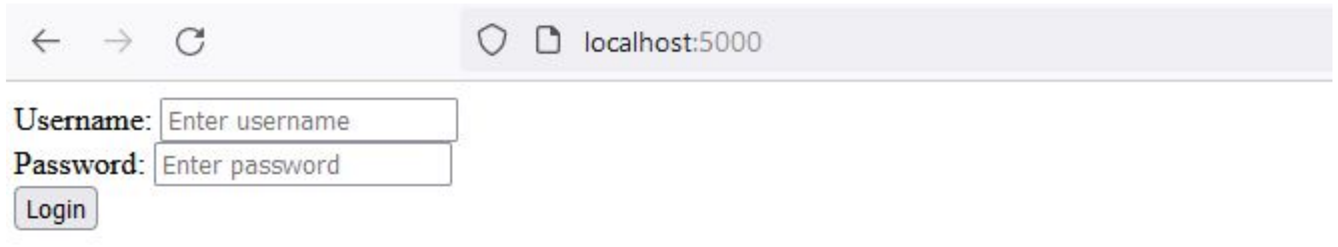
- You can view your web and database services by running

```
docker-compose ps
```

```
NAME                IMAGE               COMMAND                   SERVICE             CREATED             STATUS       PORTS
hw4-db-1            mysql:5.7           "docker-entrypoint.s…"    db                  20 seconds ago      Up 18 seconds       33060/tcp,
0.0.0.0:3307->3306/tcp
hw4-web-1           cs353hw4app         "python app.py"           web                 20 seconds ago      Up 18 seconds       0.0.0.0:5000->5000/tcp
```

Accessing your application

- Simply go to http://localhost:5000 or http://localhost:5000/login



Flask web application

Importing required packages

```
import re
import os
from flask import Flask, render_template, request, redirect, url_for, session
from flask_mysqldb import MySQL
import MySQLdb.cursors

app = Flask(__name__)

app.secret_key = 'abcdefgh'

app.config['MYSQL_HOST'] = 'db'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = 'password'
app.config['MYSQL_DB'] = 'cs353hw4db'

mysql = MySQL(app)
```

Endpoints

- Currently the endpoints are
  - /login
  - /register
- You need to implement the other endpoints in accordance with the functionality in the homework.
- Endpoints are implemented using the @app.route decorators in Flask.

Login Logic

```
@app.route('/login', methods =['GET', 'POST'])
def login():
    message = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in request.form:
        username = request.form['username']
        password = request.form['password']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM User WHERE username = % s AND password = % s', (username, password, ))
        user = cursor.fetchone()
        if user:
            session['loggedin'] = True
            session['userid'] = user['id']
            session['username'] = user['username']
            session['email'] = user['email']
            message = 'Logged in successfully!'
            return redirect(url_for('tasks'))
        else:
            message = 'Please enter correct email / password !'
    return render_template('login.html', message = message)
```

templates/login.html

```
<form action="{{ url_for('login') }}" method="post">
  {% if message is defined and message %}
    <div class="alert alert-warning">{{ message }}</div>
  {% endif %}
  <div class="form-group">
    <label for="username">Username:</label>
    <input type="username" class="form-control" id="username" name="username" placeholder="Enter username" name="username">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" id="password" name="password" placeholder="Enter password" name="pswd">
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```

Register Logic

```
@app.route('/register', methods =['GET', 'POST'])
def register():
    message = ''
    if request.method == 'POST' and 'username' in request.form and 'password' in request.form and 'email' in request.form :
        username = request.form['username']
        password = request.form['password']
        email = request.form['email']
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM User WHERE username = % s', (username, ))
        account = cursor.fetchone()
        if account:
            message = 'Choose a different username!'

        elif not username or not password or not email:
            message = 'Please fill out the form!'

        else:
            cursor.execute('INSERT INTO User (id, username, email, password) VALUES (NULL, % s, % s, % s)', (username, email, password,))
            mysql.connection.commit()
            message = 'User successfully created!'

    elif request.method == 'POST':

        message = 'Please fill all the fields!'
    return render_template('register.html', message = message)
```

templates/register.html

```
<form action="{{ url_for('register') }}" method="post">
  {% if message is defined and message %}
    <div class="alert alert-warning">{{ message }}</div>
  {% endif %}
  <div class="form-group">
    <label for="username">Username:</label>
    <input type="text" class="form-control" id="username" name="username" placeholder="Enter name">
  </div>
  <div class="form-group">
    <label for="email">Email:</label>
    <input type="email" class="form-control" id="email" name="email" placeholder="Enter email">
  </div>
  <div class="form-group">
    <label for="pwd">Password:</label>
    <input type="password" class="form-control" id="password" name="password" placeholder="Enter password">
  </div>

  <button type="submit" class="btn btn-primary">Register</button>
</form>
```

Summary

- Install Docker and Docker Compose
- Get the archive we gave you and extract it, work on it.
- Build the cs353hw4app docker container image.
- Prepare your schema.sql.
- Run docker-compose.yaml.
- Implement other pages and functionality.
- Put all in a zip archive: **surname_name_id_hw4.zip**. Upload it to Moodle.

Zip File Contents (i.e., What to Submit)

- Zip file will contain the same structure you began with.

```
├── app                      (We gave you this)
│   ├── app.py               (We gave you this, just add more functionality)
│   └── templates                 (We gave you this, add more templates or any style files anywhere)
│       ├── login.html        (We gave you this, free to change it, but not required)
│       └── register.html          (We gave you this, free to change it, but not required)
├── docker-compose.yaml       (We gave you this)
├── Dockerfile                (We gave you this)
├── requirements.txt          (We gave you this)
└── schema.sql               (We gave you this. But it is empty. You need to fill it)
```

- During grading we will only use the app files. We will not build the container image from scratch for each submission. So, **keep the following in mind:**
  - app folder (Do not change its name)
  - app.py (Do not change its name)
  - docker-compose.yaml (Do not change anything)
  - Dockerfile (Do not change anything)
  - schema.sql (Do not change the name, just fill it)
  - requirements.txt (Do not change anything)