Hello in this tutorial, we will create a simple python flask application and execute the docker commands from the python script.

<h2>1. Introduction</h2>

<strong>Flask</strong> in python programming helps to build web applications. It is easier than Django’s framework and requires less base code to implement a simple web application. To set up the application we’ll first require to install python.

<h3>1.1 Setting up Python</h3>

If someone needs to go through the Python installation on Windows, please watch <a href="https://www.youtube.com/watch?v=i-MuSAwgwCU" target="\_blank">this</a> link. You can download the Python from <a href="https://www.python.org/downloads/" target="\_blank">this</a> link.

<h3>1.2 What is Docker & Setting up Docker</h3>

In the present world, <strong>Docker</strong> is an important term,

<ul>

 <li>Often used in CI/CD platform that packages and runs the application with its dependencies inside a container</li>

<li>Is a standard for Linux Containers</li>

 <li>A <em>Container</em> is a runtime that runs under any Linux kernel and provides a private machine-like space under Linux</li>

</ul>

If someone needs to go through the Docker installation, please watch <a href="https://www.youtube.com/watch?v=R-ZMkGvh-9Y" target="\_blank">this</a> video.

<h2>2. How to run Docker Commands in Python</h2>

I am using <a href="https://www.jetbrains.com/pycharm/" target="\_blank">JetBrains PyCharm</a> as my preferred IDE. You're free to choose the IDE of your choice. Fig. 1 represents the project structure for this tutorial.

// Fig. 1: Application structure

<h3>2.1 Creating a requirements file</h3>

Add the below code to the requirements file. The file will be responsible to download and install the packages required for this tutorial.

<span style="text-decoration: underline;"><em>requirements.txt</em></span>

<pre class="brush:plain;">

Flask

</pre>

<h3>2.2 Basic input html form</h3>

Define an HTML form in the <code>templates</code> folder with two input fields responsible to accept the name and tag required for the <code>docker pull …</code> command. This page will also show the success or error messages once the operation is completed.

<span style="text-decoration: underline;"><em>index.html</em></span>

<pre class="brush:html;">

&lt;!DOCTYPE html&gt;

&lt;html lang="en"&gt;

 &lt;head&gt;

 &lt;meta charset="UTF-8" /&gt;

 &lt;title&gt;Index&lt;/title&gt;

 &lt;/head&gt;

 &lt;body&gt;

 &lt;h1&gt;Python Docker client implementation&lt;/h1&gt;

 &lt;hr&gt;

 &lt;p&gt;

 {% with messages = get\_flashed\_messages() %}

 {% if messages %}

 &lt;ul class=flashes&gt;

 {% for message in messages %}

 &lt;li&gt;{{ message }}&lt;/li&gt;

 {% endfor %}

 &lt;/ul&gt;

 {% endif %}

 {% endwith %}

 &lt;/p&gt;

 &lt;form action="/" method="POST"&gt;

 &lt;label for="name"&gt;Image name:&lt;/label&gt;

 &lt;input type="text" id="name" name="name" placeholder="Enter image name"&gt;

 &lt;br&gt;

 &lt;label for="tag"&gt;Tag:&lt;/label&gt;

 &lt;input type="text" id="tag" name="tag" placeholder="Enter tag"&gt;

 &lt;input type="submit" value="Submit" /&gt;

 &lt;/form&gt;

 &lt;/body&gt;

&lt;/html&gt;

</pre>

<h3>2.3 List images html</h3>

Create an HTML page in the <code>templates</code> folder that will show the list of docker images available in the machine. The page is accessible from the following endpoint - <code>/images</code>.

<span style="text-decoration: underline;"><em>images.html</em></span>

<pre class="brush:html;">

&lt;!DOCTYPE html&gt;

&lt;html lang="en"&gt;

 &lt;head&gt;

 &lt;meta charset="UTF-8" /&gt;

 &lt;title&gt;Images&lt;/title&gt;

 &lt;/head&gt;

 &lt;body&gt;

 &lt;h2&gt;Available images&lt;/h2&gt;

 &lt;hr&gt;

 &lt;p&gt;

 &lt;span&gt;Total images= &lt;/span&gt;{{data|length}}

 &lt;/p&gt;

 &lt;ul&gt;

 {% for ele in data %}

 &lt;li&gt;{{ ele }}&lt;/li&gt;

 {% endfor %}

 &lt;/ul&gt;

 &lt;/body&gt;

&lt;/html&gt;

</pre>

<h3>2.4 Python code</h3>

The python file will expose four different endpoints responsible for interacting with the frontend pages.

<ul>

 <li><code>/</code> endpoint - Responsible for showing the HTML page to the user via the HTTP GET call. The same endpoint will also be responsible to handle the HTTP POST call from the form and execute the <code>docker pull</code> command to fetch the required image from the docker repository</li>

 <li><code>/images</code> endpoint - Responsible to run the <code>docker images</code> command and show the result on the <code>images.html</code> page</li>

 <li><code>/remove</code> endpoint – Responsible to remove the image. I have skipped this functionality for users to play</li>

</ul>

<span style="text-decoration: underline;"><em>main.py</em></span>

<pre class="brush:python;">

import logging

import os

import subprocess

from flask import Flask, render\_template, request, redirect, flash

# [logging config

logging.basicConfig(format='%(asctime)s:%(levelname)s:%(filename)s:%(funcName)s:%(message)s',

 datefmt='%Y-%m-%d %H:%M:%S',

 level=logging.INFO)

# logging config]

app = Flask(\_\_name\_\_)

app.secret\_key = "somesecretkey"

# http://localhost:5000

@app.route('/', methods=['GET'])

def index():

 logging.info('Showing index page')

 return render\_template('index.html')

@app.route('/', methods=['POST'])

def pull():

 image\_name = request.form.get("name")

 tag = request.form.get("tag")

 try:

 updated\_tag = tag if len(tag.strip()) &gt; 0 else 'latest'

 logging.info('Pulling image= [%s] with tag= [%s]', image\_name, updated\_tag)

 sub\_cmd = 'docker pull {img\_name}:{tag}'.format(img\_name=image\_name, tag=updated\_tag)

 subprocess.run(sub\_cmd, shell=True, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

 flash(f'{image\_name} pulled successfully from Docker hub')

 except subprocess.CalledProcessError as error:

 if "repository does not exist" in error.stderr.decode("utf-8"):

 flash(f'{image\_name} not found in Docker hub')

 logging.error(error)

 elif "not found" in error.stderr.decode("utf-8"):

 flash(f'{image\_name} not found for {tag} tag in Docker hub')

 logging.error(error)

 else:

 flash('Something unexpected has occurred. Check app logs')

 logging.error(error)

 return redirect('/')

# http://localhost:5000/images

@app.route('/images', methods=['GET'])

def images():

 logging.info('Listing images')

 sub\_cmd = 'docker images --format "Id= {{.ID}}; Name= {{.Repository}}; Tag= {{.Tag}}"'

 # with open('output.txt', "w") as outfile:

 # subprocess.run(sub\_cmd, shell=True, check=True, stdout=outfile, stderr=outfile)

 res = subprocess.run(sub\_cmd, text=True, stdout=subprocess.PIPE).stdout.splitlines()

 return render\_template('images.html', data=res)

# http://localhost:5000/remove

@app.route('/remove', methods=['POST'])

def remove():

 raise Exception('Not yet implemented.')

if \_\_name\_\_ == '\_\_main\_\_':

 # Development only: run "python app.py" and open http://localhost:5000

 server\_port = os.environ.get('PORT', '5000')

 app.run(debug=False, port=server\_port, host='0.0.0.0')

</pre>

<h2>3. Code run & demo</h2>

For localhost debugging, you can run the application by running the <code>main.py</code> py file in the PyCharm IDE. The application will be started on the <code>5000</code> port number and you can hit the following url - <code>http://localhost:5000</code> in the browser of your choice to display the index HTML page.

// Fig. 2: Welcome page

Enter the details in the input fields and click the submit button. For the demo, I have entered the image name as - <code>hello-world</code> and tag as - <code>latest</code>. Once the image is downloaded successfully a success message will be shown as in Fig. 3.

// Fig. 3: Pulling docker image

In case of failure, the error message will be shown on the same page. Now to verify whether the image was downloaded successfully or not I have created another endpoint available at <code>http://localhost:5000/images</code>. This endpoint will trigger the <code>docker images</code> command to fetch all the available images in your machine and display them on the html page as shown in Fig. 4.

// Fig. 4: Images list

Due to security reasons, only the required image is shown while others are cropped. That is all for this tutorial and I hope the article served you with whatever you were looking for. Happy Learning and do not forget to share!

<h2>4. Summary</h2>

In this tutorial, we learned about docker and created a simple python flask application to trigger the docker commands from different endpoints. You’re free to play around with this application and change it as per your needs. You can also download the source code of this tutorial from the <a href="#projectDownload">Downloads</a> section.

<h2><a name="projectDownload"></a>5. Download the Project</h2>

This was a tutorial on executing docker commands from the python script.

<div class="download"><strong>Download</strong><br />You can download the full source code of this example here: </div>

=============================================================================================

Application name - <strong>How to run Docker Commands in Python</strong>