

# **DEPLOYMENT OF FLASK APP ON AWS EC2, DOCKER, APACHE WEB SERVER**

-By Gunjan Toora

## **ABOUT THE PROJECT**

Applied Regression techniques on a [wine quality dataset](#) to train the model to predict the wine quality with new data provided by the User. A simple flask application was created with basic HTML coding and GET/POST requests to take user inputs for all the model parameters and display the result (wine quality- low, high) to the user. Pickle file was generated and requirements.txt file was also created (Note that the libraries and their versions that were used in creating the machine learning models must be stated correctly).

## **GOALS**

We initially deployed our flask app on the AWS EC2 machine which is a good and easy to use option to work with a remotely managed server. We then worked with Dockers to see how we can containerize our application. Then we wanted to explore how we can deploy our application on an industry grade webserver which essentially will deploy our application in production. So, we chose Apache Web Server to work without application and we then docker-ized this whole set-up.

## **PROJECT PLAN**

### **Phase 1:**

Picking up a simple dataset with easy-to-understand parameters that a user can work with in the Flask application → Training and testing a simple machine learning model with that dataset → Creating a flask application with html front end capable of taking user inputs

### **Phase 2:**

Setting up the AWS EC2 server → Establishing connection with the server → Transferring files and running the application → Modifying the flask app/ html code as and when needed

### **Phase 3:**

Researching and learning about Docker → Building Docker file → Building Docker image and running it → troubleshooting as needed

### **Phase 4:**

Learning about the integration of flask app, Apache Server, Docker → Installing Apache locally and connecting with it → Modifying existing files → Understanding WSGI and creating WSGI file → Integrating everything and dockerizing it → troubleshooting and fixing errors

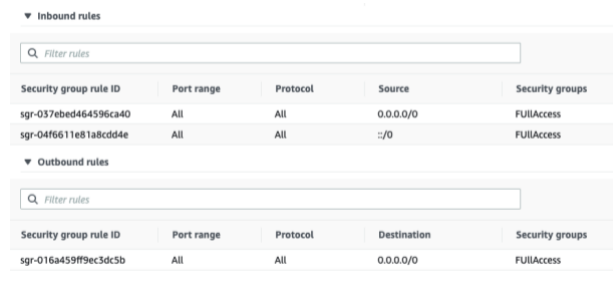
## **DETAILED STEPS OF THE PROCESS**

### **Deploying our Model on AWS EC2 Server**

There were five major steps that we followed to deploy our model:

#### **1. Configuring the AWS EC2 machine -**

In setting up the AWS EC2 instance, we chose an ubuntu machine, while keeping all other default settings. More storage can be added in the future as per the requirements of the project. We downloaded the key-pair .cer file. For the purpose of this project alone- we modified the Security group settings as shown below-



The screenshot shows the AWS Security Groups console. Under 'Inbound rules', there are two rules: 'sgr-037ebcd464596ca40' and 'sgr-04f6611e81a8cd04e', both allowing all traffic from 0.0.0.0/0. Under 'Outbound rules', there is one rule: 'sgr-016a459ff9ec3dc5b', allowing all traffic to 0.0.0.0/0. All rules are associated with the 'FullAccess' security group.

Inbound rules				
Filter rules				
Security group rule ID	Port range	Protocol	Source	Security groups
sgr-037ebcd464596ca40	All	All	0.0.0.0/0	FullAccess
sgr-04f6611e81a8cd04e	All	All	::/0	FullAccess

Outbound rules				
Filter rules				
Security group rule ID	Port range	Protocol	Destination	Security groups
sgr-016a459ff9ec3dc5b	All	All	0.0.0.0/0	FullAccess

This is not advisable since this exposes all our ports which could potentially become a security issue. We had just been initially keeping the ports open to ease our deployment process. Otherwise, we must expose only a limited number of ports publicly.

The next step was to start this instance.

#### **2. Connecting with the EC2 machine via SSH**

Once our EC2 server was up and running, we established a connection with it from our local terminal with the following commands-

- `chmod 400 <key pair file name>`

This command essentially sets the permission of our key file and make it unaccessible to others, which is required before establishing a connection with the respective server.

- `ssh -i <key pair file name> ubuntu@<serverIP>`

Here we use the SSH command that provides a secure encrypted connection between two hosts over an insecure network.

### 3. Installing python and jupyter on EC2

The following commands were used to install anaconda-python on our EC2 instance

- `wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86\_64.sh`
- `chmod +x Miniconda3-latest-Linux-x86_64.sh`
- `./Miniconda3-latest-Linux-x86_64.sh`

### 4. Transferring our application files to EC2

Files can be transferred from our local machine to a remote server using many methods like using FTP transfer tools like FileZilla or directly from the terminal. We used the following command to transfer all our files including- ML python file, flask app file, csv data set file, pickle file, requirements.txt, html files associated with the flask app-

- `scp -i <cer file path> <local file path> <ubuntu file path>`

### 5. Running the flask application on EC2 server

After establishing the connection with our EC2 server via SSH, we are able to run commands within the server. We first install all the libraries required to run our app by installing the requirements.txt file with pip

- `pip3 install -r requirements.txt`

Now we are ready to run our flask application on the EC2 server

- `python flask_app.py`

```
(base) ubuntu@ip-172-31-87-244:~/mlops-code-examples/flask_example_regression$ python flask_simple_regression_service.py
* Serving Flask app 'flask_simple_regression_service' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.31.87.244:5000/ (Press CTRL+C to quit)
```

To see the app deployed on our EC2, we open our browser and open the address:

<public DNS of our Ec2 machine>:8080

Example:

<https://ec2-54-159-146-164.compute-1.amazonaws.com:8080/>

## Predict if your wine is high quality

ENTER:

fixed acidity

volatile acidity

citric acid

residual sugar

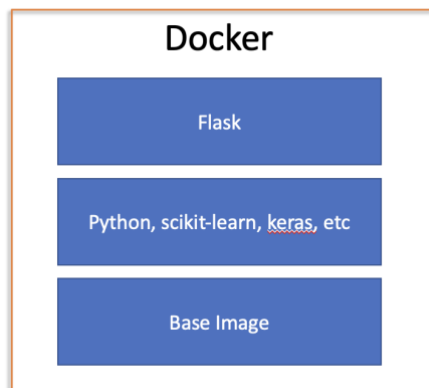
chlorides

Note that we exposed the port 8080 in our flask application code as well.

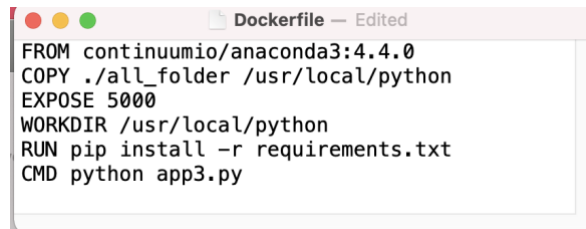
## Deploying our Model on Docker

Main steps involved after installing Docker Desktop Software:

### 1. Creating the docker file



The first component of working with Dockers is the docker file. It is in this file we decide the Base layer, and other top layers that our Docker image will be constructed from. Below is the snippet of our docker file



```
FROM continuumio/anaconda3:4.4.0
COPY ./all_folder /usr/local/python
EXPOSE 5000
WORKDIR /usr/local/python
RUN pip install -r requirements.txt
CMD python app3.py
```

In Line 1, for this project we decided to choose continuumio anaconda base image of our docker. This is because it has the bootstrapped installation of Anaconda (based on Python 3.X) ready to use and we need not install python or anaconda manually later.

Then we COPY our local folder which contains all our application files (as before) and this docker file to any desired location on our docker. We expose the port 5000 for this project here this time. We also use that location as our working directory from where we run our requirements.txt first and then use the command to run our flask application file.

## 2. Building the docker image

In order to build our docker image, we go to command line and cd to our folder that contains- the docker file, another folder that contains the rest of our application files.

After that we use the following command to start building our image:

- `docker build -t <docker image name> .`  
where we can give any name to our image and the dot in the end indicates the command to look for the files in the current directory itself.

Then we can check our new image with the following command that lists the images:

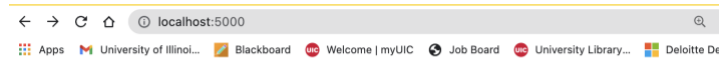
- `docker image ls`  
We were able to see our image listed here.

## 3. Running it

We run our docker image using the following command that binds the 5000 ports of our local machine with the 5000 port of the docker. (Note that we exposed the port 5000 in our flask application code this time)

- `docker run -p 5000:5000 <docker image name>`

To check our application running, we open localhost:5000 on our browser. This way we are running the container from the image locally.



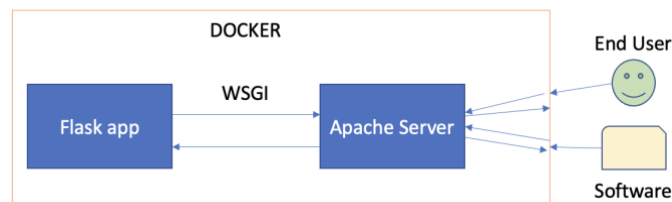
Predict if your wine is high quality

ENTER:

fixed acidity

## Dockerizing our ML model with Apache Web Server

In a real business setting, an application is deployed in production (not on development), where an industry grade web server handles all the incoming traffic of requests to the application. We tried to replicate the same setting with our application which works in the below manner-



Here our Apache web server will interact with our web application via WSGI (web server gateway interface). All the requests being sent by end users or other software applications will now come to the Apache server. This server will route our requests to our flask application. Then the functions present in our flask application will be evoked accordingly and send the results back to the Apache server via WSGI. The Apache server then pushes back the response to the user.

To execute this, there are mainly 3 steps involved:

### **1. Changes to the requirements.txt**

Simply add “mod\_wsgi” library name to the requirements file which is needed to work with WSGI.

### **2. Creating a new WSGI file**

```

flask_app.wsgi — wine_flask_docker_apache

#!/usr/bin/python

import sys
sys.path.insert(0, "/var/www/flask_predict_api")
sys.path.insert(0, '/opt/conda/lib/python3.6/site-packages')
sys.path.insert(0, "/opt/conda/bin/")

import os
os.environ['PYTHONPATH'] = '/opt/conda/bin/python'

from app4 import app as application

```

This file essentially helps our Apache web server to connect with our flask application. Note that the path `/var/www/flask_predict_api` is the path where we will copy our applications file. And ‘app4’ is our flask application file name and ‘app’ is the name of our application function.

### 3. Changes in the docker file

```

Dockerfile — Edited

FROM continuumio/anaconda3:4.4.0
EXPOSE 8000
RUN apt-get update && apt-get install -y apache2 \
    apache2-dev \
    vim \
    && apt-get clean \
    && apt-get autoremove \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /var/www/flask_predict_api/
COPY ./flask_app.wsgi /var/www/flask_predict_api/flask_app.wsgi
COPY ./all_folder /var/www/flask_predict_api/
RUN pip install -r requirements.txt
RUN /opt/conda/bin/mod_wsgi-express install-module
RUN mod_wsgi-express setup-server flask_app.wsgi --port=8000 \
    --user www-data --group www-data \
    --server-root=/etc/mod_wsgi-express-80
CMD /etc/mod_wsgi-express-80/apachectl start -D FOREGROUND

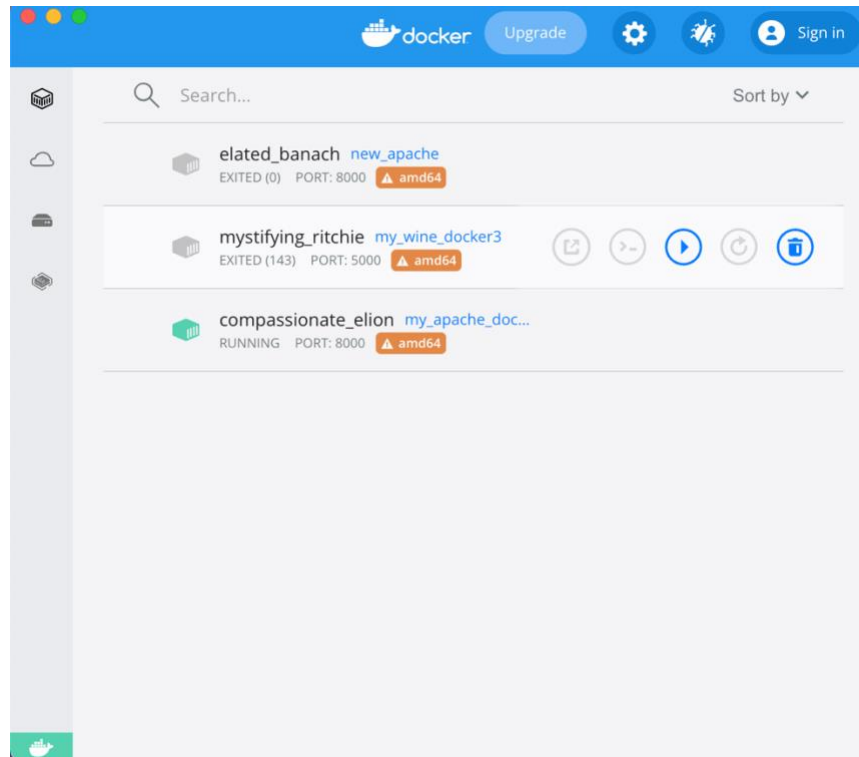
```

We modify our existing docker file from our previous set up. This time we expose the port 8000 which is the configuration for our Apache web server (and vim as well). Then we install Apache in our docker and then we transfer all our application files (including this docker file and the WSGI file). Then running the requirements.txt, the wsgi modules, and our wsgi file as well along with setting up our server by defining the users, groups, and the root directory. In the end we are just starting our Apache server.

### 4. Building the new docker image

Now we again create a new docker image using the same process and commands

- `cd <folder path with all our files>`
- `docker build -t <new image name> .`
- `docker run -d -p 8000:8000 <new image name>`



*Docker GUI showing the list of dockers and the ones running*

Optionally, to go inside the docker and check if all your files are copied or for any other purpose like checking the error logs (which we did), we run the following commands:

- `docker ps`

This command lists all the running docker images. Copy the ID of your running image

```
wine_flask_docker_apache — zsh — 114x24

=> => transferring context: 3.11kB 0.0s
=> CACHED [2/8] RUN apt-get update && apt-get install -y apache2 apache2-dev vim && apt-get clean && a 0.0s
=> CACHED [3/8] WORKDIR /var/www/flask_predict_api/ 0.0s
=> CACHED [4/8] COPY ./flask_app.wsgi /var/www/flask_predict_api/flask_app.wsgi 0.0s
=> CACHED [5/8] COPY ./all_folder /var/www/flask_predict_api/ 0.0s
=> CACHED [6/8] RUN pip install -r requirements.txt 0.0s
=> CACHED [7/8] RUN /opt/conda/bin/mod_wsgi-express install-module 0.0s
=> CACHED [8/8] RUN mod_wsgi-express setup-server flask_app.wsgi --port=8000 --user www-data --group www- 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:4c1ccb76b2a7e47183493316654c5a75722ce2c60d6efede94d19aebb8f0c222 0.0s
=> => naming to docker.io/library/my_apache_docker 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
(base) gunjantoor@Gunjans-MacBook-Air wine_flask_docker_apache % docker run -d -p 8000:8000 my_apache_docker
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) a
nd no specific platform was requested
ad6d3cbaae443ac5314e98957d18551963b8c409a66248473de7af43bae96982
(base) gunjantoor@Gunjans-MacBook-Air wine_flask_docker_apache % docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS
NAMES
ad6d3cbaae44   my_apache_docker   "/usr/bin/tini -- /b..." 7 seconds ago  Up 6 seconds  0.0.0.0:8000->8000/tcp
compassionate_elion
(base) gunjantoor@Gunjans-MacBook-Air wine_flask_docker_apache %
```

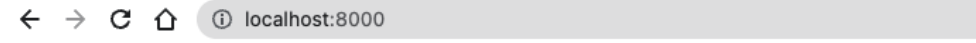
*docker ps command listing the running docker and its details*

- `docker exec -it <ID> bash`



- `cd /etc/mod_wsgi-express-80`  
This is the same path that we defined in the dockerfile
- `ls`  
This is where you can find the error log file. Open the error file using *vim*

To see our application running, go to localhost:8000



**Predict if your wine is high quality**

**ENTER:**

**fixed acidity**

**volatile acidity**

**citric acid**

*Application running on localhost:8000*

In case we have a domain name, we can set that in our configurations of Apache server, then our application and results run on that domain name.

## **SOURCES**

- <https://hub.docker.com/r/continuumio/anaconda3>
- <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>
- Udemy course- “Learn to build Machine Learning, Deep Learning & NLP Models & Deploy them with Docker Containers (DevOps) (in Python)” by UNP United Network of Professionals
- <https://httpd.apache.org/docs/2.4/howto/auth.html>
- <https://smallbusiness.chron.com/upload-files-apache-server-41190.html>
- <https://assistanz.com/installing-apache-web-server-in-windows-container-using-docker-file/>

- <https://www.codementor.io/@abhishake/minimal-apache-configuration-for-deploying-a-flask-app-ubuntu-18-04-phu50a7ft>
- <https://towardsdatascience.com/containerize-your-application-with-docker-b0608557441f>