



**Prepared by**

Esabil Bulbul

# 1. Index

## Contents

<b>1. Index .....</b>	<b>2</b>
<b>2. Terminology.....</b>	<b>3</b>
<b>3. Introduction.....</b>	<b>4</b>
<b>4. NSFW Code .....</b>	<b>5</b>
<b>5. Docker Image.....</b>	<b>9</b>
<b>6. Requirements .....</b>	<b>9</b>
<b>7. Auxiliaries .....</b>	<b>10</b>
<b>8. Sample Rest API Deployment on Linux .....</b>	<b>10</b>
<b>9. Deploying NSFW API .....</b>	<b>17</b>
<b>9.1. Overview.....</b>	<b>17</b>
<b>9.2. Deployment.....</b>	<b>18</b>
<b>Step 1 Preparing Base Image .....</b>	<b>18</b>
<b>Step 2 Preparing Final Image .....</b>	<b>18</b>
<b>Step 3 Running Container .....</b>	<b>21</b>
<b>Testing Services .....</b>	<b>22</b>
<b>Calling NSFW Service API / Method.....</b>	<b>23</b>
<b>Troubleshooting .....</b>	<b>24</b>
<b>References .....</b>	<b>27</b>
<b>NGINX &amp; WSGI.....</b>	<b>27</b>
<b>About Flask &amp; Docker .....</b>	<b>27</b>

## 2. Terminology

Bid	Offering for ads
CPM	Cost Per Impression. This is the bidding cost per 1k view aka Mille

### 3. Introduction

This document explains how NSFW (Not Suitable for Work) algorithm deployment. The images uploaded first go through virus check then thru NSFW check. For NSFW, ShipShuk uses the algorithm of Yahoo with a predefined dataset 1000.

The NSFW algorithm developed under python3. Therefore, the only part of Shipshuk running on python will be this algorithm. The NSFW algorithm works with Caffe deep learning framework. Therefore Caffe also needs to be deployed.

The python application will be deployed under ubuntu linux. The application developed as webservice on ubuntu. The backend service will call ubuntu webservice to calculate NSFW score. The backend service will call this service on ubuntu.

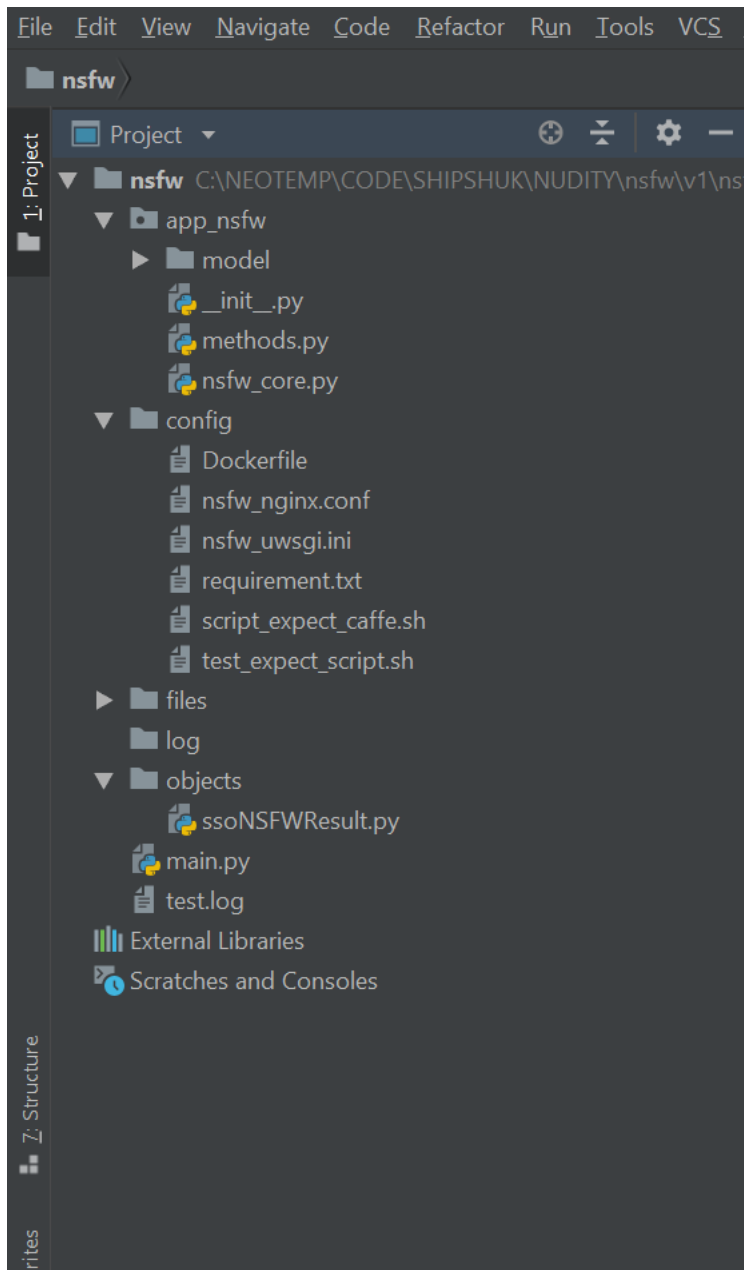
For the service, lightweight Flask used with Python. For the webserver, uWSGI and NGINX used. uWSGI is used because NGINX can't host apps. For development purpose, Flask development/debug webserver can be used however the production environment NGINX should be used.

In development environment, PyCharm is used.

The base server environment will be CentOS. Therefore, this application will be deployed in Docker container. The container will be based on Ubuntu. The communication in CentOS environment with the one in container will be through REST API.

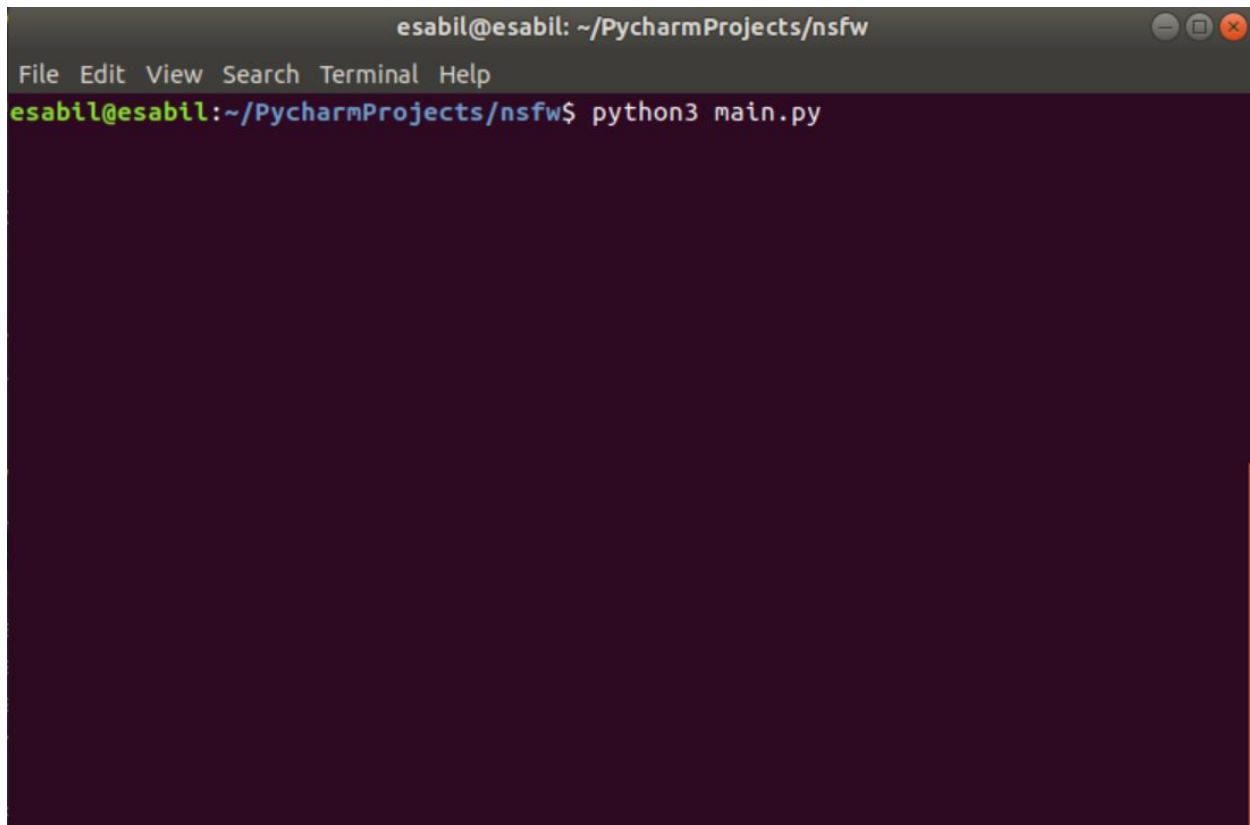
## 4. NSFW Code

The application code is illustrated shown as below. App is the package where the methods and the nsfw by yahoo algorithm located. `__init__.py` files make a directory perceived as package in flask. Therefore we added this file. The files are located under files folder. The configuration files such as dataset and model layers are located under config folder.



The webservice api returns json class of `ssoNSFWResult`. The parameters are starttime, endtime and score calculated for the image.

To test the files one can call the following command line on terminal

A screenshot of a terminal window. The title bar at the top reads 'esabil@esabil: ~/PycharmProjects/nsfw'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal area has a dark purple background. The prompt 'esabil@esabil:~/PycharmProjects/nsfw\$' is shown in green, followed by the command 'python3 main.py' in white. The rest of the terminal area is empty.

```
esabil@esabil: ~/PycharmProjects/nsfw
File Edit View Search Terminal Help
esabil@esabil:~/PycharmProjects/nsfw$ python3 main.py
```

---

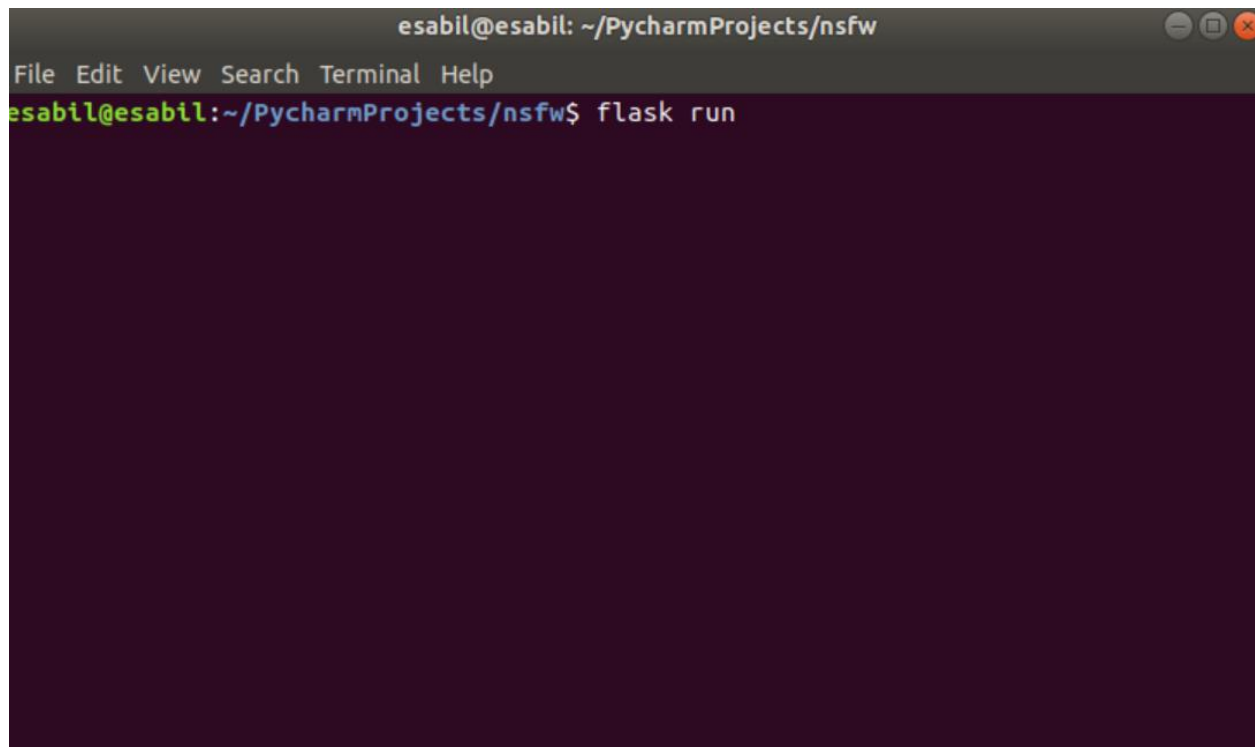
### *Python3 main.py*

---

The main.py has sample test codes where nsfw library is initialized and a test predict method is called. The config files should be located under /config folder whereas the test files to be predicted should be under /File.

The default python version coming with ubuntu is version 2. However, just make sure you download python3 and call the python code with that. Python calls python v2 whereas python3 calls python v3.

When it comes to testing/consuming the service through rest api through http GET run flask code

A screenshot of a terminal window. The title bar at the top reads 'esabil@esabil: ~/PycharmProjects/nsfw'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal area has a dark purple background. The prompt 'esabil@esabil:~/PycharmProjects/nsfw\$' is followed by the command 'flask run' in a light blue color. The rest of the terminal area is empty.

```
esabil@esabil: ~/PycharmProjects/nsfw
File Edit View Search Terminal Help
esabil@esabil:~/PycharmProjects/nsfw$ flask run
```

---

*Flask run*

---

```
esabil@esabil: ~/PycharmProjects/nsfw
File Edit View Search Terminal Help
not need backward computation.
I0819 00:47:17.863049 5170 net.cpp:200] pool1_pool1_0_split does not need backw
ard computation.
I0819 00:47:17.863052 5170 net.cpp:200] pool1 does not need backward computatio
n.
I0819 00:47:17.863055 5170 net.cpp:200] relu_1 does not need backward computati
on.
I0819 00:47:17.863059 5170 net.cpp:200] scale_1 does not need backward computat
ion.
I0819 00:47:17.863061 5170 net.cpp:200] bn_1 does not need backward computation
.
I0819 00:47:17.863065 5170 net.cpp:200] conv_1 does not need backward computati
on.
I0819 00:47:17.863067 5170 net.cpp:200] data does not need backward computation
.
I0819 00:47:17.863070 5170 net.cpp:242] This network produces output prob
I0819 00:47:17.863188 5170 net.cpp:255] Network initialization done.
I0819 00:47:17.907248 5170 upgrade_proto.cpp:77] Attempting to upgrade batch no
rm layers using deprecated params: ./config/resnet_50_1by2_nsfw.caffemodel
I0819 00:47:17.911254 5170 upgrade_proto.cpp:80] Successfully upgraded batch no
rm layers using deprecated params.
I0819 00:47:17.922750 5170 net.cpp:744] Ignoring source layer loss
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

When it comes to production use NGINX web server with uWSGI container. This will be explained in following section.

```
127.0.0.1:5000/predict/esabil x +
127.0.0.1:5000/predict/esabil/08172018?f=lowneck3.jpg
{"endtime": "20180818101820610", "score": "0.391316", "starttime": "20180818101818520"}
```



## 5. Docker Image

## 6. Requirements

Requirement	Installation
Python 3	Command Line <b>sudo apt-get install python3.6</b>
Flask (rest api framework)	Command Line  <b>TO INSTALL</b> sudo apt-get install python3-pip pip3 install flask  <b>TO DEBUG RUN</b> (Not recommended for production) flask run
Caffe	<a href="https://modeldepot.io/mikeshi/yahoo-open-nsfw">https://modeldepot.io/mikeshi/yahoo-open-nsfw</a>  (some more detail if needed such as feeding more dataset) <a href="https://github.com/yahoo/open_nsfw">https://github.com/yahoo/open_nsfw</a>
PyCharm (IDE)	<a href="https://www.jetbrains.com/pycharm/download/">https://www.jetbrains.com/pycharm/download/</a>
Nginx	<a href="https://vladikk.com/2013/09/12/serving-flask-with-nginx-on-ubuntu/">https://vladikk.com/2013/09/12/serving-flask-with-nginx-on-ubuntu/</a>
uWSGI	

## 7. Auxiliaries

Package	Description
<b>Installation</b> Sudo apt install net-tools	Install NetStat Tools (to see who is using the ports)
<b>Use case</b> Sudo netstat -tulpn Sudo kill -2 <pid>	Browser Localhost:80 Nginx page will open
Curl <addr>	To send HTTP GET/POST requests

## 8. Sample Rest API Deployment on Linux

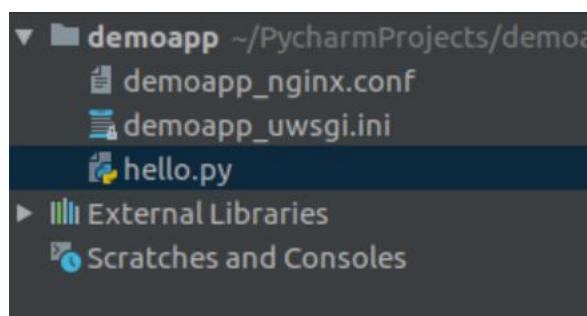
This section explains how a simple REST API can be deployed. On linux environment we will be using NGINX and UWSGI to server our APIs. NGINX is the webserver and UWSGI is the application container. Apache and tomcats of the apache world.

In development environment when we are running the python script with “python3 <file>.py” it runs the web server with test parameters i.e. 5000 port. We will configure the server to run with our parameters. There will be one config file for NGINX (.conf) and another for UWSGI (.ini). NGINX and UWSGI talks through socket between them.

Sample REST Application Modules

Module	Detail
Main python script	.py file
.ini	To configure uwsgi (app container)
.conf	To configure nginx (webserver)

Project (Demoapp) Structure



Our main file is hello.py and the code is as following

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
@app.route("/index")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

We have one api in the file. "/" and "/index" are the paths to consume the API. If you run "python hello.py" you will get the following outputs on terminal

```
^Cesabil@esabil:~/PycharmProjects/demoapp$ python hello.py
* Serving Flask app "hello" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

This is debug mode and not recommended for the production environment. Therefore, we will configure our own nginx server and uwsgi container.

Exit the debug mode with Ctrl + C as it will keep the port occupied.

## Step 1: Configuring NGINX Server

In this step we are configuring our NGINX server to talk with UWSGI container. NGINX Server Configuration. We will name the file as `<projectname>_nginx.conf`. For this sample project, it will be `demoapp_nginx.conf`.

```
server {  
    listen      80;  
    charset     utf-8;  
    client_max_body_size 75M;  
  
    location / { try_files $uri @yourapplication; }  
    location @yourapplication {  
        include uwsgi_params;  
        uwsgi_pass unix:127.0.0.1:9090;  
    }  
}
```

We will use 9090 port for our nginx server to communicate with uwsgi container.

To edit/create files open with sudo (root) user. You can use vi editor (i: insert, <semicolon>: wq! -> write + quit, dd: delete line)

The command line to edit/create should be like this `sudo vi demoapp_nginx.conf`

The next thing is to link configuration file is to nginx configuration files location. Use following command on terminal

---

```
sudo ln -s /<app.home.folder>/demoapp_nginx.conf /etc/nginx/conf.d/  
(Red colored parameter is application specific)
```

---

**IMPORTANT:** Use full path when you are specifying the location of config file. Otherwise, the link created will give the error: “too many levels of symbolic links” on start of nginx. The link under conf.d will have red color.

To start nginx server on command line

```
esabil@esabil:/var/log/uwsgi$ sudo /etc/init.d/nginx restart
[sudo] password for esabil:
[ ok ] Restarting nginx (via systemctl): nginx.service.
```

The log of nginx servers can be found under /var/log/nginx/. The file will be error.log

Useful Links for NGINX:

<https://www.cyberciti.biz/faq/nginx-restart-ubuntu-linux-command/>

## Step 2: Configuring UWSGI Container

There are two ways of passing the configuration parameters to wsgi container. The first one is via command line and the second is using a .ini file. And passing that ini file to command line with `-ini` parameter on command line.

For example;

### Command line direct parameters

```
esabil@esabil:~/PycharmProjects/demoapp$ sudo uwsgi --http-socket :9090 --plugin
python --wsgi-file hello.py --callable app
```

---

```
uwsgi --http-socket :9090 --plugin python3 --wsgi-file main.py --callable app
```

---

Following command line use hybrid some parameters are defined in ini file and passed on command line the others with directive on command line.

```
esabil@esabil:~/PycharmProjects/demoapp$ sudo uwsgi --ini /home/esabil/PycharmProjects/demoapp/demoapp_uwsgi.ini --plugin python --wsgi-file hello.py --callable app
```

---

*uwsgi --ini /usr/src/app/nsfw/config/nsfw\_uwsgi.ini --wsgi-file main.py --callable app*

*if this doesn't work try to run it additionally with "--plugin python" option*

---

**--wsgi-file** is where the main file is defined

**Note:** --wsgi-file if used alone will give an error. It should be used with --plugin parameters

UWSGI configuration file should have a flask **"application"** object to call for the request. If you are using different name for it, you can specify the object name with the parameter **--callable**. In our sample code we are using "app" therefore we specify it with -- callable app.

The next thing is to start our wsgi container with the following code

```
esabil@esabil:~/PycharmProjects/demoapp$ sudo uwsgi --http-socket :9090 --plugin python --wsgi-file hello.py --callable app
```

We can use the parameters above in an ini file and call it with only "sudo uwsgi --ini <filename>" on terminal.

There will be messages from uwsgi container when started as following. And the terminal will stay there (will NOT fall in next command). To fall back to command line use Ctrl + C

```
thunder lock: disabled (you can enable it with --thunder-lock)
uwsgi socket 0 bound to TCP address :9090 fd 3
dropping root privileges after socket binding
uwsgi running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uwsgi as root !!! (use the --uid flag) ***
Python version: 2.7.15rc1 (default, Apr 15 2018, 21:51:34) [GCC 7.3.0]
*** Python threads support is disabled. You can enable it with --enable-threads
***
Python main interpreter initialized at 0x55ad55ca1770
dropping root privileges after plugin initialization
uwsgi running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uwsgi as root !!! (use the --uid flag) ***
your server socket listen backlog is limited to 100 connections
your mercy for graceful operations on workers is 60 seconds
mapped 72768 bytes (71 KB) for 1 cores
*** Operational MODE: single process ***
WSGI app 0 (mountpoint='') ready in 0 seconds on interpreter 0x55ad55ca1770 pid:
 10606 (default app)
dropping root privileges after application loading
uwsgi running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uwsgi as root !!! (use the --uid flag) ***
*** uwsgi is running in multiple interpreter mode ***
spawned uwsgi worker 1 (and the only) (pid: 10606, cores: 1)
```

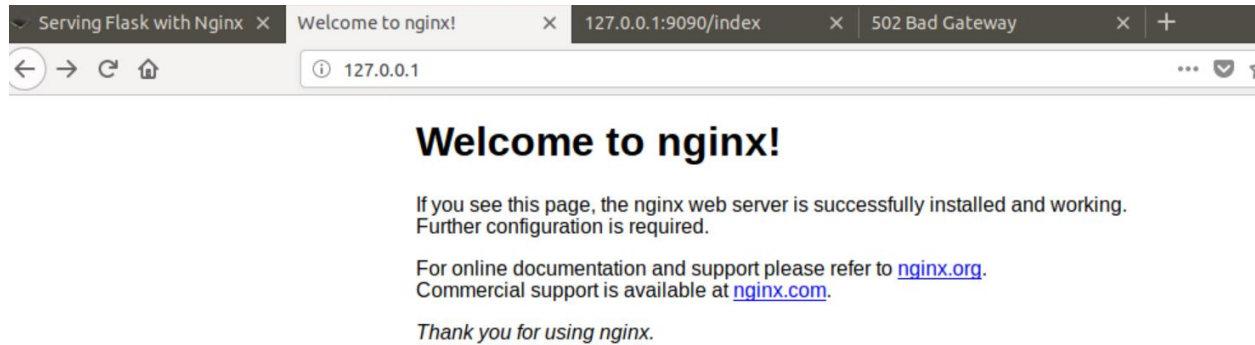
The log of uwsgi can be found under `/var/log/uwsgi/` with the file name `<project name>_uwsgi.log` file as. In our case it will be `"demoapp_uwsgi.log"`

```
esabil@esabil:/var/log/uwsgi$ ls
app                flask_helloworld_uwsgi.log  nsfw_usgi.log.1
demoapp_uwsgi.log  nsfw_usgi.log
```

### Step 3: Consuming Rest API

Now we started our NGINX server and UWSGI container. It is time to consume it. We can use curl or we can use browser to see the return from the server

To check if NGINX server running, type 127.0.0.1 on browser or curl <http://127.0.0.1>

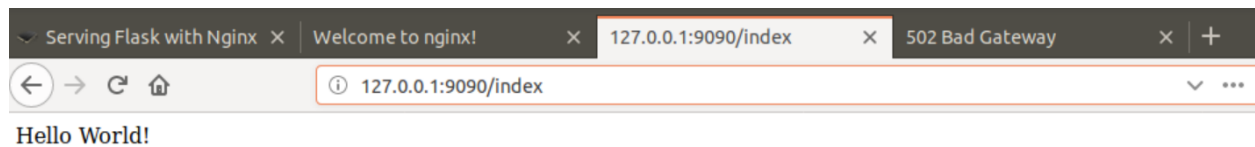


To check rest api and UWSGI

On terminal:

Curl <http://127.0.0.1/index>

Or on browser:



**Note:** The codes used with python2x therefore the usage of uwsgi for python3 codes might be different. This note will be updated once tested with python3



## 9. Deploying NSFW API

### 9.1. Overview

This deployment of NSFW described here explains nsfw implementation on windows 10. The docker image will running on ubuntu, the image files that will be checked located on windows side. This means we will be mounting drives to the container side.

Deployment of NSFW algorithm completed in a couple steps. These are

- Preparation of base image
- Preparation of final image
- Running container

In first step, the base application such as ubuntu, python, web & app servers and so on are installed into the application. The second phase is to install some other packages requiring human interactivity. Just because we were not able to manage to run expect scripts / auto responding scripts for docker image buildings we chose to create this final step. The packages installed in this step will be committed to a new image which will be our final base image.

The final step will be running the docker with the volumes that will be used by the service.

Before starting to deployment steps there are a few parameters that needs to be set correctly that will be used for integration from windows to ubuntu side.

### 9.2. Requirements

#### **Project Directory on Windows**

C:\NEOTEMP\CODE\SHIPSHUK\NUDITY\nsfw\v1\nsfw

**Note:** Change this directory to the directory you have the nsfw project you placed.

#### **Image Repository Directory on Windows**

C:\temp\rep

**Note:** Change this directory to the directory you have the images are stored

#### **Port Number for Service**

9090

In case of the directories change, the changes should be reflected to the deployment commands

### Test Images Directory

Copy the test folder under the image repository to test if the service is up and running.

In this sample installation we copy the test directory under c:\temp\rep

Test directory comes with one subdirectory and an test image file within that directory: lowneck1.jpg

## 9.3. Deployment

### Step 1 Preparing Base Image



On docker console, Compiling/Building the image

---

```
docker build -t img_ubuntu1 ./config
```

---

### Step 2 Preparing Final Image

Run the base image as container

---

```
docker run -v C:\NEOTEMP\CODE\SHIPSHUK\NUDITY\nsfw\v1\nsfw:/usr/src/app/nsfw -p  
9090:9090 -itd img_ubuntu1 /bin/bash
```

---

Red Color Parameter = Project Directory on Windows

Green Color Parameter = Project Directory on Ubuntu

Note: The directory on windows is getting mounted on Ubuntu side.

A terminal window with a dark blue background. The title bar shows a terminal icon, the text "root@bf8328cae9b6: /usr/src/app/nsfw", and window control buttons. The terminal content shows the prompt "root@bf8328cae9b6: /usr/src/app/nsfw#" followed by a blank line.

When you logged in ubuntu terminal console, run the following commands

---

```
ln -s /usr/src/app/config/nsfw_nginx.conf /etc/nginx/conf.d/
```

---

This command will create the configuration file for webserver.

Then install caffe

---

```
apt-get -y install caffe-cpu=1.0.*
```

---

Upon completion of the installation, leave container without exiting. You can do this with the following keyboard combinations

A terminal window with a dark blue background. The title bar shows a terminal icon, the text "root@bf8328cae9b6: /usr/src/app/nsfw", and window control buttons. The terminal content shows the prompt "root@bf8328cae9b6: /usr/src/app/nsfw#" followed by a blank line.

---

```
Ctrl - P + Ctrl -Q
```

---



After this you will be falling back to docker console. To check if the container still up and running you can type the following command on docker console

---

### *Docker ps*

---

#### Sample Screen Shot

```
PS C:\NEOTEMP\code\SHIPSHUK\NUDITY\nsfw\v1\nsfw> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
bf8328cae9b6   img_ubuntu2_caffe  "/bin/bash"             About an hour ago    Up About an hour    0.0.0.0:9090->9090/tcp    zealous_yonath
```

Final command of this step is to commit the changes/packages installed and rename the last image with a new name

---

```
docker commit <containerid> img_ubuntu_caffe
```

---

Note: Container Id can be found in the output of “docker ps” command

Now we have our new image “img\_ubuntu\_caffe”. To list the images available on docker console type “docker images”

We need to kill our previous container running. For that we will use the following command on docker shell

---

```
docker stop <containerid>
```

---

### Step 3 Running Container

In this step we will be running the final base image on container. The following command will be used

---

```
docker run -v C:\temp\rep:/usr/src/app/rep -v  
C:\NEOTEMP\CODE\SHIPSHUK\NUDITY\nsfw\v1\nsfw:/usr/src/app/nsfw -p 9090:9090 -itd  
img_ubuntu1_caffe /bin/bash
```

---

Red Color Parameter = Directory of Image Repository on windows

Green Color Parameter = Directory of Image Repository on Ubuntu

Yellow Color Parameter = Project Directory on Windows

Purple Color Parameter = Project Directory on Ubuntu

Then we will start the web and application services with the following command

---

```
docker exec <dockerId> /bin/bash -c 'uwsgi --ini /usr/src/app/nsfw/config/nsfw_uwsgi.ini  
--callable app && service nginx start'
```

---

## Testing Services

You can test the service in two ways.

- Open web browser and type “http://127.0.0.1:9090/ “  
or
- “docker exec <containerId> -it bash” login to ubuntu server

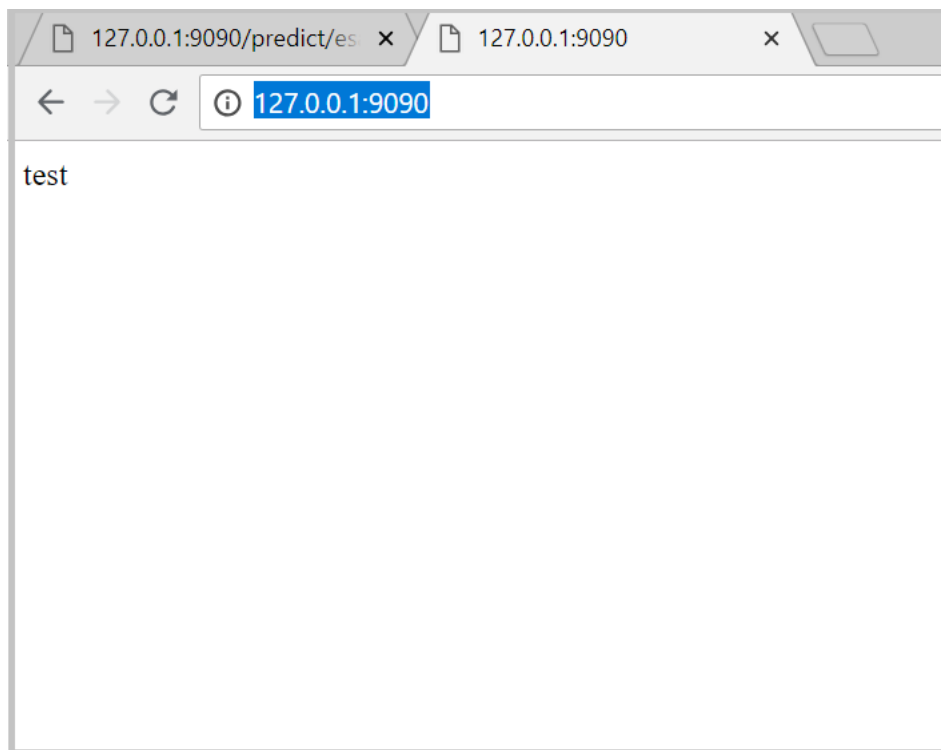
Run following command on ubuntu console

---

```
curl http://127.0.0.1:9090/
```

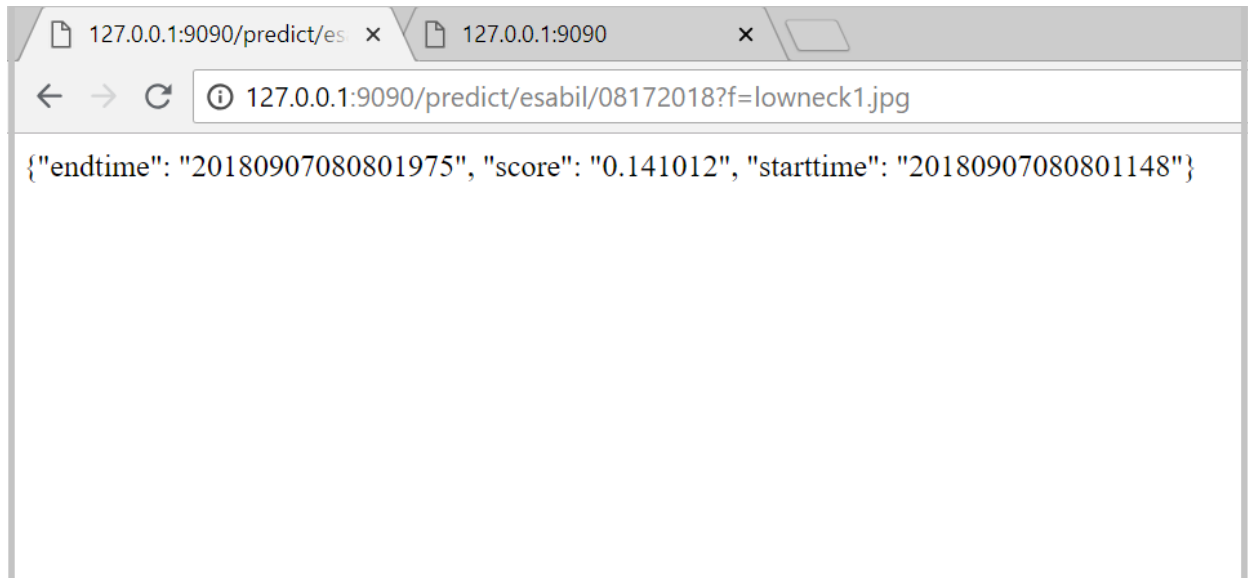
---

The response to either one of the calls should be “test”



In case service is not running, see Troubleshooting section

### Calling NSFW Service API / Method



---

<http://127.0.0.1:9090/predict/esabil/08172018?f=lowneck1.jpg>

---

#### Service Parameters

- IP
- Port
- Method name = predict
- User Name = esabil
- Date = 08172018
- File = lowneck1.jpg

Important: The service will check the file under repository with specified parameters folder. The repository folder for the following request should have the following folder + file

<Root Rep. Folder>\<username>\<date>\File

## Troubleshooting

There are several reasons the service may not be working

- 1. Services are up however NSFW API is not functional**

In this case, make sure you have the directories properly set up from windows to ubuntu / container side. If they are set up correct, then check your ports are set up correct.

- 2. Check the webserver link created properly**

The configuration link is located under `/etc/nginx/conf.d/`. The configuration link file should be blue. If it is red, the configuration link is not properly set up.

```
root@9ede507fa042: /etc/nginx/conf.d
```

```
root@9ede507fa042:/etc/nginx/conf.d# ls
nsfw_nginx.conf
root@9ede507fa042:/etc/nginx/conf.d#
```

The reason link file may not be blue is the `"ln -s /usr/src/app/config/nsfw_nginx.conf /etc/nginx/conf.d/"` should not be correctly run. This should not happen but if it is the case then make sure you run the script when you are under `.../app/nsfw` application folder. This is important. If you run it when is under other folder the link will fail. I guess the reason uwsgi try to create the link under callable (in ini file) under the assumption source file is in the same directory.

- 3. If it is still not working, login to container with "docker exec <containerId> -it bash" command and check web and application services**

### Check the log files

There are two log files related to the service. 1<sup>st</sup> log file is webserver "nginx" log file located under `/var/log/nginx/error.log`. Other log file is application server log file "uwsgi" which located under `/var/log/nsfw3.log`.

If somehow you run the final base manual scripts, then you might ended up application server not mounted to the nsfw python application.

In a correct set up start, nsfw3 log file should have the following lines



```

I0907 07:52:28.442643 17 net.cpp:200] bn_stage0_block0_branch2b does not need backward computation.
I0907 07:52:28.442647 17 net.cpp:200] conv_stage0_block0_branch2b does not need backward computation.
I0907 07:52:28.442651 17 net.cpp:200] relu_stage0_block0_branch2a does not need backward computation.
I0907 07:52:28.442654 17 net.cpp:200] scale_stage0_block0_branch2a does not need backward computation.
I0907 07:52:28.442658 17 net.cpp:200] bn_stage0_block0_branch2a does not need backward computation.
I0907 07:52:28.442662 17 net.cpp:200] conv_stage0_block0_branch2a does not need backward computation.
I0907 07:52:28.442665 17 net.cpp:200] scale_stage0_block0_proj_shortcut does not need backward computation.
I0907 07:52:28.442669 17 net.cpp:200] bn_stage0_block0_proj_shortcut does not need backward computation.
I0907 07:52:28.442673 17 net.cpp:200] conv_stage0_block0_proj_shortcut does not need backward computation.
I0907 07:52:28.442677 17 net.cpp:200] pool1_pool1_0_split does not need backward computation.
I0907 07:52:28.442682 17 net.cpp:200] pool1 does not need backward computation.
I0907 07:52:28.442684 17 net.cpp:200] relu_1 does not need backward computation.
I0907 07:52:28.442688 17 net.cpp:200] scale_1 does not need backward computation.
I0907 07:52:28.442692 17 net.cpp:200] bn_1 does not need backward computation.
I0907 07:52:28.442701 17 net.cpp:200] conv_1 does not need backward computation.
I0907 07:52:28.442705 17 net.cpp:200] data does not need backward computation.
I0907 07:52:28.442708 17 net.cpp:242] This network produces output prob
I0907 07:52:28.442801 17 net.cpp:255] Network initialization done.
I0907 07:52:28.569944 17 upgrade_proto.cpp:77] Attempting to upgrade batch norm layers using deprecated params: /usr/src/app/nsfw/ap
p_nsfw/model/resnet_50_1by2_nsfw.caffemodel
I0907 07:52:28.570006 17 upgrade_proto.cpp:80] Successfully upgraded batch norm layers using deprecated params.
I0907 07:52:28.574144 17 net.cpp:744] Ignoring source layer loss
!!! methods loaded !!!
!!! __init__.py loaded !!!
!!! name: app_nsfw
uWSGI app 0 (mountpoint='') ready in 1 seconds on interpreter 0x55d29530cc70 pid: 17 (default app)
uWSGI running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 17, cores: 1)
*** Starting uWSGI 2.0.17.1 (64bit) on [Fri Sep  7 08:07:45 2018] ***
compiled with version: 7.3.0 on 07 September 2018 05:09:05

```

4. Still nothing works, fine, kill all the process related to web and application server. And try to start the process manually.

To list the process running, type the following command on ubuntu console

```
ps ax
```

```

root@9ede507fa042: /var/log
root@9ede507fa042:/var/log# ps ax
PID TTY      STAT   TIME COMMAND
  1 pts/0    Ss+    0:00 /bin/bash
 17 ?        S       0:02 uwsgi --ini /usr/src/app/nsfw/config/nsfw_uwsgi.ini --callable app
 59 ?        Ss      0:00 nginx: master process /usr/sbin/nginx
 60 ?        S       0:00 nginx: worker process
 61 ?        S       0:00 nginx: worker process
 62 pts/1    Ss      0:00 bash
 80 pts/1    R+      0:00 ps ax
root@9ede507fa042:/var/log#

```

Then kill the processes related to nginx and uwsgi with the following command

```
kill <pid> <pid2> <pid3> ....
```

Then run the following commands step by step

---

```
uwsgi --ini /usr/src/app/nsfw/config/nsfw_uwsgi.ini --callable app
```

---

Then start the web service

---

```
service nginx start
```

---

Then check the service as described in section [Troubleshooting](#)

## References

### NGINX & WSGI

<https://code-maven.com/deploying-python-flask-using-uwsgi-on-ubuntu-14-04>

To fix wsgi-file problem (plugin python)

<https://smekalov.me/start-test-python-using-uwsgi/>

<https://unix.stackexchange.com/questions/422868/nginx-connection-refused-error-111-bad-gateway>

<https://stackoverflow.com/questions/21190394/nginx-and-uwsgi-connection-refused-and-502-bad-gateway-error>

<https://stackoverflow.com/questions/23148082/in-django-nginx-wsgi-what-is-a-mysite-sock>

<https://vladikk.com/2013/09/12/serving-flask-with-nginx-on-ubuntu/>

### About Flask & Docker

Some Good Tutorials

<https://www.youtube.com/watch?v=4T5Gnrmzjak&t=601s>

[https://www.youtube.com/watch?v=s\\_ht4AKnWZg](https://www.youtube.com/watch?v=s_ht4AKnWZg)