From model inception to deployment

medium.com/datadriveninvestor/from-model-inception-to-deployment-adce1f5ed9d6

Akshay Arora November 26, 2018

Machine Learning model training & scalable deployment with Flask, Nginx & Gunicorn wrapped in a Docker Container

Akshay Arora
Nov 27

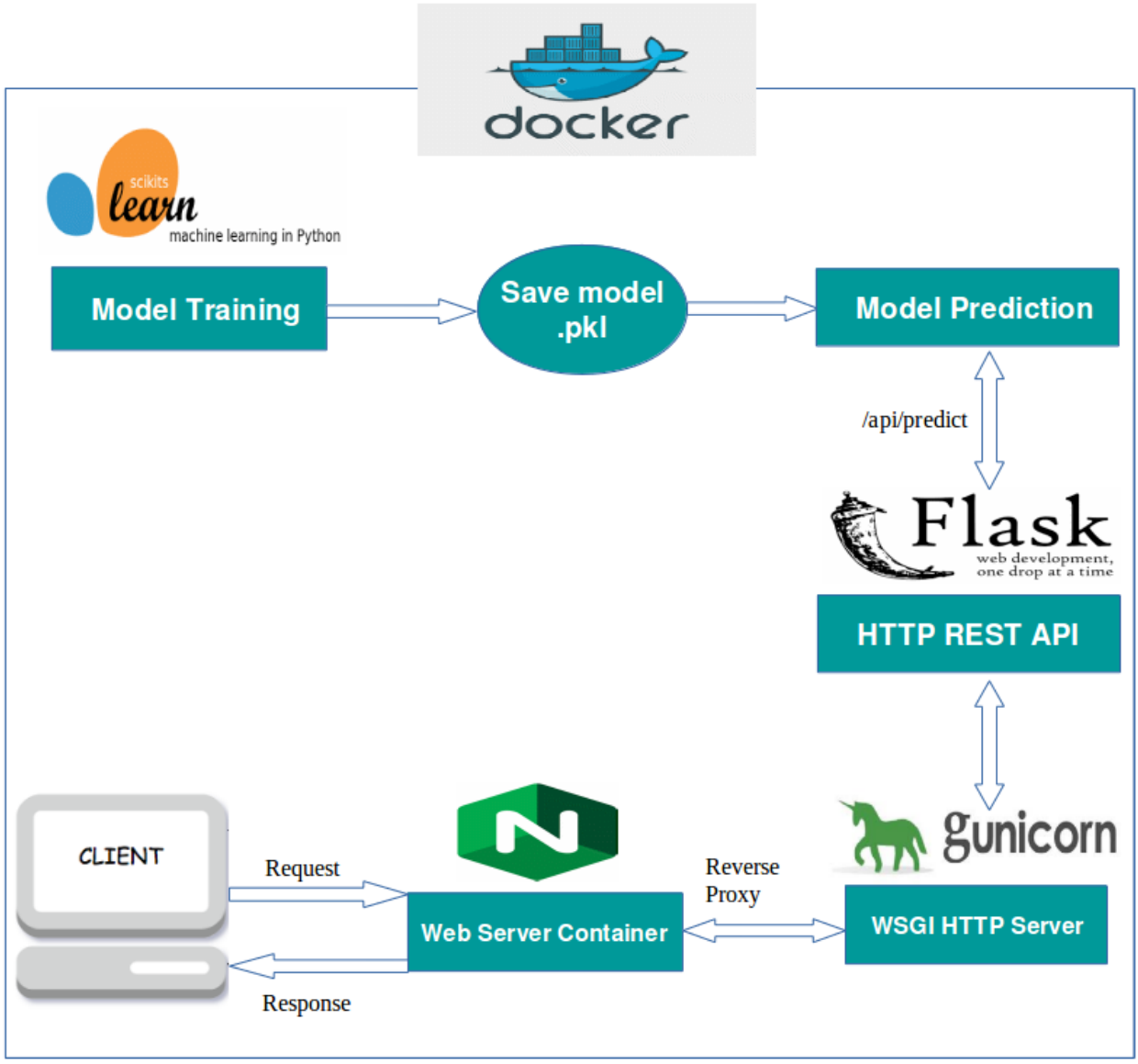
We all have been in this position after we are done building a model :p

At some point, we all have struggled in deploying our trained Machine Learning model and a lot of questions start popping up into our mind. What is the best way to deploy a ML model? How do I serve the model's predictions? Which server should I use? Should I use flask or django for creating REST API? What about shipping it inside docker? Don't worry, I got you covered with all of it!! :)

In this tutorial, we will learn how to train and deploy a machine learning model in production with more focus on deployment because this is where we all data scientists get stuck.

Also, we will be using docker containers, one for flask app and another for Nginx web server shipped together with docker-compose. If you are new to docker or containerization, I would suggest reading *this article*.

High-Level Architecture

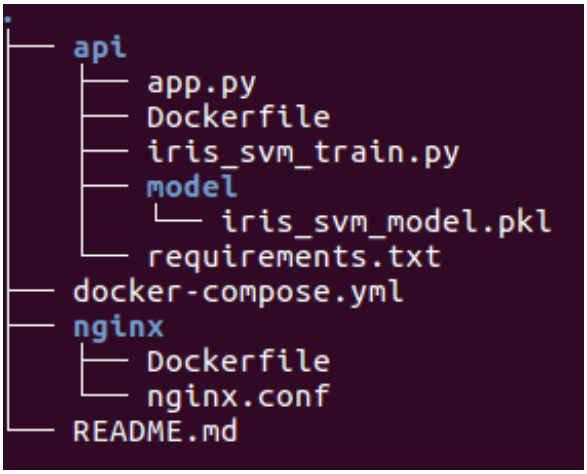


High level design of large scale Machine Learning model deployment

Setting up

Here is the GitHub link for this project

This is the folder structure that we will follow for this project



Let's break this piece into three parts:
—Training a Machine Learning model using python & scikit-learn
—Creating a REST API using flask and gunicorn
—Deploying the Machine Learning model in production using Nginx & ship the whole package in a docker container

Model Training

To keep things simple and comprehensive, we will use iris data-set to train a SVM classifier.

iris_svm_train.py

Here, we are training a Support Vector Machine with a linear kernel which is giving a pretty decent accuracy of 97%. Feel free to play around with the training part, try Random Forest or Xgboost & perform hyper-parameter optimization to beat the accuracy.
Make sure you execute the 'iris_svm_train.py' because it will save the model inside the 'model' folder (*refer to github repo*).

Building a REST API

Creating a flask app is very easy. No kidding!
All you need to know is how a request from the client(user) is sent to the server and how the server sends back the response and a little bit about GET and POST methods. Below, we are loading our saved model and processing the new data (request) from the user in order to send predictions(response) back to the user.

app.py

We will use gunicorn to serve our flask API. If you are on windows, you can use waitress (pure-Python WSGI server) as an alternative to gunicorn.
Execute the command: **gunicorn -w 1 -b :8000 app:app** and hit **http://localhost:8000**' in your browser to ensure your flask app is up and running. If you get the message 'Hoilaaaaaaaaa!', then you are good to go!!

```
If you want to test the predict(Post) method, use curl command or use Postman

curl --header "Content-Type: application/json" --request POST --
data'[{"sepal_length":6.3,"sepal_width":2.3,"petal_length":4.4,"petal_width":1.3}]'
http://localhost:8000/predict
```

Deploying the ML model in production

Finally, fun part begins :)
We will use Nginx web server as a reverse proxy for Gunicorn, meaning users will hit Nginx from the browser and it will forward the request to your application. Nginx sits in front of Gunicorn which serves your flask app.
More information on why Nginx is required when we have gunicorn: link

nginx.conf

Wrapping everything inside Docker Container

Congratulations, you have made it to the last part.
Now, we will create two docker files, one for API & one for Nginx. We will also create a docker-compose file which will contain information about our two docker containers. You have to install docker and docker-compose for this to work. Let's ship our scalable ML app and make it portable & production ready.

Docker file for API (keep it in api folder)

We have created a docker file for API which needs to be saved inside 'api' folder with other files including requirements.txt (containing information about python packages required for your app).

Docker file for Nginx(keep it in nginx folder with nginx.conf file)

docker-compose.yml

docker-compose.yml is the master file which binds everything together. As you can see, it contains two services, one for api & one for server(nginx). Now, all you need is just a single command to run your ML app:

```
cd <project/parent directory>
docker-compose up
```

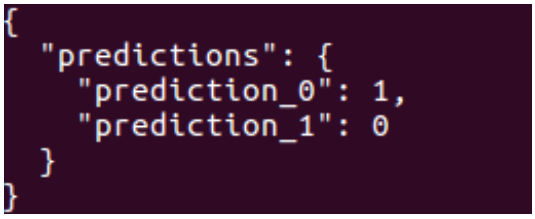
```
api is up-to-date
server_nginx is up-to-date
Attaching to api, server_nginx
api        | [2018-11-25 16:34:01 +0000] [1] [INFO] Starting gunicorn 19.9.0
api        | [2018-11-25 16:34:01 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
api        | [2018-11-25 16:34:01 +0000] [1] [INFO] Using worker: sync
api        | [2018-11-25 16:34:01 +0000] [10] [INFO] Booting worker with pid: 10
api        | [2018-11-26 22:43:26 +0000] [1] [INFO] Starting gunicorn 19.9.0
api        | [2018-11-26 22:43:26 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
api        | [2018-11-26 22:43:26 +0000] [1] [INFO] Using worker: sync
api        | [2018-11-26 22:43:26 +0000] [10] [INFO] Booting worker with pid: 10
```

Output of above command

Cheers! Your dockerized scalable Machine Learning app is up and running, accepting requests on port 8080 and ready to serve your model's predictions.
Open a new terminal to run predict method using curl or use Postman

```
curl --header "Content-Type: application/json" --request POST --
data'[{"sepal_length":6.3,"sepal_width":2.3,"petal_length":4.4,"petal_width":1.3}]'
http://localhost:8080/predict
```

Predictions from your deployed ML model



Thank you for making it till here, comment below if you face any challenges in running the project or have any feedback. Happy Learning!!