Ungraded lab: First look at Tensorflow Serving with Docker

Welcome to this ungraded lab. Here you will take a look at using Tensorflow Serving with Docker. This is one of the easiest ways to get introduced to this awesome serving system for machine learning models since the image already contains all the necessary dependencies and configuration to run TFS out of the box.

In this lab you will be using TFS to deploy a dummy machine learning model locally. This lab is inspired by this official TF tutorial.

If you are a Windows user remember that **this lab is meant to be run using a WSL2 shell.** To open such shell use the Windows search bar and type either `wsl` or `bash`, one of these should be available if you installed WSL2 previously.

Open your terminal (or shell) and let's get started!

Pulling the image

Begin by pulling the TFS docker image from the Docker hub:

```
docker pull tensorflow/serving
```

This is the most minimal image that you can run TFS on. It contains all of the necessary dependencies to run TFS and was created with image size in mind, as a result of this it is around 400 mb in size.

Clone the repo with the dummy model

Now you will clone the official TFS repo, which contains a dummy model named `Half Plus Two` that returns `0.5 * x + 2` for any value of `x`.

You will do the cloning in the temporal directory of your filesystem so that your machine does not get cluttered.

Run the following three commands:

```
mkdir -p /tmp/tfserving

cd /tmp/tfserving

git clone https://github.com/tensorflow/serving
```

These commands perform these operations in order:

- Create a directory called `tfserving` under the temporal directory `/tmp`.
- Change your current directory to the one that was just created
- Clone the repo in that location

After running these commands you can return to your previous directory by using `cd -` or you can simply close this command line window.

Run Tensorflow Serving

Since the image you just pulled contains all of the software needed to run the model under TFS, all that is left is to run a container out of the image.

A vanilla `docker run` looks like this:

```
docker run name-of-the-image
```

However, you can specify different flags to achieve different functionalities. You will see how this works in a bit.

Take a look at the command that will spin up a container to serve the model under TFS:

```
docker run --rm -p 8501:8501 \
  --mount type=bind,\
source=/tmp/tfserving/serving/tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_
target=/models/half_plus_two \
  -e MODEL_NAME=half_plus_two -t tensorflow/serving &
```

Wow, there is a lot of information in this command. Let's break it down to understand what every flag is doing:

- `--rm`: Delete this container after stopping running it. This is to avoid having to manually delete the container. Deleting unused containers helps your system to stay clean and tidy.
- `-p 8501:8501`: This flags performs an operation knows as **port mapping**. The container, as well as your local machine, has its own set of ports. In order to access the `port 8501` within the container, you need to **map** it to a port on your computer. In this case it is mapped to the `port 8501` in your machine. This port is chosen as it is the default port to interact with the model through a `REST API`. If you were using a different protocol such as `gRPC` you will need to use `port 8500`. More information on this in the tutorial mentioned at the beginning of the lab.
-

`--mount type=bind,source=dir/in/your/pc,target=dir/in/container`: This flag allows you to **mount** a directory in your pc to a directory within the container. This is very important because containers usually have short lifetimes and without mounting files onto them there is no way of persisting changes done to these files when the container was running.

- `-e MODEL_NAME=half_plus_two`: Will create the environment variable `MODEL_NAME` and assign to it the value of `half_plus_two`.
- `-t`: Attaches a pseudo-terminal to the container so you can check what is being printed in the standard streams of the container. This will allow you to see the logs printed out by TFS.

After running this command TFS will spin up and host the `Half Plus Two` model.

Consuming the model

Now that the model is being served on `port 8501` you can use an HTTP client to get predictions from it. Going forward you will be shown how to do this with `curl` but feel free to use any client of your choice.

Since you need to provide some data that the server will process you should use a `HTTP POST` request.

Let's do inference for a batch of three numbers, open a new command line window or tab and run the following command:

```
curl -d '{"instances": [1.0, 2.0, 5.0]}' \
  -X POST http://localhost:8501/v1/models/half_plus_two:predict
```

As with the `docker run` command let's break down the flags in this one:

- `-d`: The `d` stands for data. This is the data that you are going to send to the server for it to process. Since you are communicating with the model via `REST` you should provide the data in a `JSON` format. TFS has the convention that the **key** of this object should be the string `instances` and the **value** should be a list that contains each data point that you want to make inference for.
- `-X`: This flag allows you to specify the desired `HTTP method`. By default `curl` uses the `GET` method but in this case it should be `POST`.

The last parameter of the command is the `URL` where it should make the request to. Let's break down the `URL` as well:

- `http://localhost:8501`: Stands for your own machine in the port 8501, as you specified earlier.
- `v1`: Refers to the version of TFS used. Right now this part of the `URL` will always be `v1`.
- `models/half_plus_two`: This part refers to what model should be served. Since you set the environment variable `MODEL_NAME` to have the value `half_plus_two`, this is the name of the model.
- `predict`: Allows TFS to know that the model is gonna be used for inference (or prediction).

After running the request you should be prompted with the prediction for each one of the three numbers you submitted.

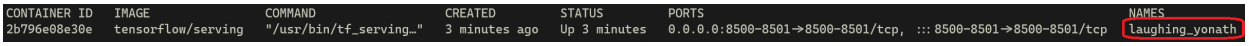
Stopping the server

Finally you will learn how to stop the server running within the Docker container. In this case to stop the container is equivalent to stopping the server, to do so, run the following command to see all of the running Docker processes (or containers):

```
docker ps
```

This will display some relevant information for each running docker process. If you want to also check this information for stopped containers use the flag `-a`.

Docker automatically assigns a unique name to each container and this can be seen with the above command which should yield an output similar to this:



To stop a container simply use the command:

```
docker stop container_name
```

In this case the command will be:

```
docker stop laughing_yonath
```

After some seconds you should see that the process exited on the terminal that you spined up the container.

Congratulations on finishing this ungraded lab!

Now you should have a better sense of how Docker can be leveraged to serve your Machine Learning models. You should also have a better understanding of how to use the `docker run` command to spin up containers and of how to use `curl` to interact with web servers.

This lab used a dummy model but in future labs you will see this process with real models and further TFS features.

Keep it up!