Deployment Manager

Model Deployment Workflow

i) The Data Scientist uploads a pickle file containing the model, a contract.py file containing the pre-processing and post-processing functions and a config.json file containing deployment information (model_name, dependencies etc). [*UI forms present under /templates/dashboard.html*]

Sample config.json for a model:

ii) After the files are uploaded, at the backend the following 2 processes are run:

- Dockerfile Generation [*using the script - DockerFileGenerator.py*]

In this process, a Dockerfile for the model as a service is created using the contents of the config.json file.

A Dockerfile can be viewed as a sequence of instructions written in a specific syntax to set up the environment in the docker image that contains everything that is required for the service to run.

A Dockerfile should also contain the entry point to the service.

Dockerfile constructed based on the config.json file given above:

```
FROM ubuntu:20.04
RUN apt-get update
RUN apt-get install -y python3-pip
RUN pip3 install Flask
RUN pip3 install requests
RUN pip3 install numpy
RUN pip3 install sklearn
RUN pip3 install pymongo
ENV FLASK_APP=WrapperClass.py
COPY WrapperClass.py /
COPY corona_model.pkl /
CMD python3 WrapperClass.py
```

Explanation of the Dockerfile contents shown above:

The first 3 lines are to setup the base os (here we use ubuntu 20.04) and then install python3 and pip in the docker image.

The next 5 lines are added based on the dependencies mentioned in the given config.json file.

The next few lines are responsible for copying the source files into the docker image and also setting up the flask app and the entry point of the docker container as the WrapperClass.py file [This file will be created in the next step].

- Wrapper Class Generation [*using the script - WrapperClassGenerator.py*]

In this process, WrapperClass.py file is generated which contain the class from the Contract.py file and a flask api with 4 endpoints : /preprocess, /predict, /postprocess and /healthCheck [scope of expansion: can make the endpoints flexible]

These endpoints utilize the preprocess and postprocess functionalities provided in the Contract class.

This file becomes the entrypoint for the model as a service.

iii) After the WrapperClass and the Dockerfile are created in the previous step, the following steps (under deploy_model function in Dashboard.py) are performed to deploy the model as a public service:

- **Choosing a VM for deployment using the LoadBalancer.**
  To select a VM for deployment, the Load Balancer is used. The LoadBalancer inspects the cpu and ram of all the available VMs and chooses the one with the least load. [Code for this can be found under Load_Balancer.py]

  We use paramiko to connect to Azure VMs through the ssh ports.

- **Move all the required files for the model as a service into the selected VM.**
  The generated Dockerfile, WrapperClass.py and the uploaded pickle file are moved into the selected VM using an ftpClient.

- **Update DB with information about the new model as a service.**

- **Build the docker image using the Dockerfile in the VM.**

  The following command is run in the VM using sshClient of the paramiko library to build the docker image.

```
sudo docker build . -t <modelName>
```

- **Run the docker image to create a docker container which exposes the service on a designated port.**

  The following command is run in the VM using sshClient of the paramiko library to create a docker container from the docker image built.

```
sudo docker run -p <service_port>:5000  <modelName>
```

  Here <service_port> is a placeholder for the next available port number. The service would be exposed at this port and could be accessed using the public ip of the VM.

Application Deployment Workflow

i) The End App Developer uploads a zip file containing the source files, a config.json file containing the dependencies, source file names and the sensor and model information.

ii) A Dockerfile is generated using the config.json file.

iii) These files are uploaded to FileShare under a folder named after the application.

iv) Now, the End User can select this uploaded application and schedule it to be deployed during a particular time interval. Also, they can pick the sensor locations for the sensors to be used.

v) The scheduler sends a request to the deployment manager (at endpoint: /deploy_app) to deploy the application at the scheduled time.

vi) After receiving the request from the Scheduler, the deployment manager would follow similar steps as was followed for deploying a model to get the application deployed.