Exercise 3 – 2023S

The goal of the assignment is to develop a data processing application using Docker, Python, and TensorFlow and understand the mechanisms behind computation and data offloading. The assignment will be deployed and tested via AWS, utilizing its cloud infrastructure for seamless scalability and efficient computation. You will be working in the same group which you have formed earlier. The assignment consists of four parts:

1. Implementation of data processing application

The application should be implemented inside the detection_loop function of the app.py provided in the attached *.zip* archive. Students should develop one application with the following functionality:

- **Object detection**: the application takes as input an image and returns the objects that have been detected on the image. It should also work for multiple images. Images are sent to the application via Post request as an array of base64 encoded Strings. Results should be returned as JSON Response.

The dataset with images datasets is provided to the students via Tuwel. As a reference for their implementation, students can use official TensorFlow documentation. More precisely, Object detection: https://www.tensorflow.org/hub/tutorials/object_detection

The application should be at least commented on the main parts. We want to be sure that you understand what you are doing.

2. Dockerization of application

The developed application should be dockerized, i.e., deployed inside a Docker container. Students are encouraged to use the *Dockerfile* attached, which sets up the basic service needed for the execution of Docker. Please note that you might need to include more Python packages according to the design of your implementation. To add new packages, it is recommended to add them in the *requirements.txt* file.

Additional resources can be found in the official Docker documentation.

3. Local and remote execution

Once you deploy your container with your application running, execute your application on the provided datasets and collect data about the **average inference time** of your implementation. By Inference time, we mean the time it takes for the application to perform the required object recognition. If not already provided by TensorFlow methods, you can obtain this value for a single image by using additional Python modules, i.e., *time* or *time it*. The average should be calculated on the whole images on the provided datasets.

Local execution:

Local execution can be done with docker clients for the operation system of your choice. A Docker container must be built and afterward can be started. As it is required to send images to Docker container via Post request, you either prepare a client script or use any other tool which allows you to send multiple images to your Docker container. Images should be sent as Base64 encoded Strings to avoid any problems and can be decoded according to your needs on the server.

Remote Execution

For remote execution, the containerized app should be deployed on Amazon Web Services. Once it is deployed, you should collect data about (1) the time required to transfer each image file in the dataset and (2) the average inference time on AWS.

AWS Specific steps:

As on the local machine, data can be sent to AWS using the same methods. Just keep in mind to use the URL of the AWS instance instead of localhost. Moreover, it is necessary to specify the correct ports. The port depends on your port forwarding settings in AWS. You will need to send the Base64 encoded image to the AWS service endpoint.

AWS resource access:

We have partnered with the AWS academy program to get access to AWS resources. This allows us to create our own AWS classes (learner lab), from which you can access the AWS console without requiring you to provide your credit card details. Each student has a $100 budget, which would be sufficient for your project task unless you create unnecessary resources on AWS.

You will receive an invitation to AWS learner lab class via email. We will use the student's email address; students are required to create their AWS account only using the invitation issued to their address. Finally, you will see a Lab Environment, which is usable for the whole assignment.

URL: https://www.awsacademy.com/vforcesite/LMS_Login

Student login-> Dashboard-> AWS Academy Learner Lab (46890) -> Modules-> Learner Lab -> Start Lab.

Once the lab environment is ready, you can click on the AWS icon (with the green button on the menu bar), which opens the AWS console. You can interact with all the resources from here onwards.

For more details, please read carefully about how to access the learner lab and AWS resources using the attached student guide.

Attention: *Sessions will be refreshed after 4 hours – afterward, you need to refresh the access tokens if you are using AWS CLI.*

AWS Setup:

You will need different services provided by AWS to deploy and run your docker container there. Students can choose how to get their application running on AWS, but a few helpful services are listed below:

- Standalone EC2 instance with docker engine (recommended for remote execution solution), which is almost exactly similar to local execution with additional setup for creating your own cloud virtual machine for application hosting. Choose any supported isntance type for your account.
- AWS ECR + AWS lambda, ECS, These are advanced architectures, if you have experience using any of these services, then you could try these instead of EC2.

- We recommend you use AWS Cloud9 service if you want to use AWS CLI for minimal overhead in setting up your environment.

Note: Each account is allocated with "Lab Role" IAM role which gives restricted access to AWS services. You are allowed only to use this role while creating resources. Before using any AWS services, please check what is the access policy for your AWS services according to LabRole.

4. Report

Students are expected to provide a report on the developed application and on local and remote execution. Produce a report.pdf that contains a detailed report including at least five sections, 1. Introduction, 2. Problem Overview, 3. Methodology and Approach, 4. Results, and 5. Conclusions.

The Methodology and Approach section should have an architecture diagram explaining your offloading solution, including a short description of how your setup on AWS was built. Results should comment on these data, i.e., explaining whether it is worth offloading data to the remote AWS or not and in which conditions it makes sense to execute locally or to offload. The conclusion should be supported by the data you collected. The overall report should not exceed more than 8 pages (A4 size).

Submission

Please submit a single file named <GroupID>_DIC2023_Ex3.zip that contains:

- Dockerfile and requirements.txt.
- The app.py implementation and include all additional files (if any) you implemented to make it work.
- The client script that handles input data communication with your docker service
- Any other source code and configuration file that you have used.
- A final report in .pdf format. 8 page (max), 11pt font size, one column format

Evaluation

Evaluation will be mainly performed based on the quality of the provided report. The report should contain the following:

1. Information on how the application has been developed, explaining the most important design choices and any modification to the Docker file or requirements.txt needed to make your implementation work.
2. Commands needed to deploy your container.
3. Explanation of how you calculate inference and transfer time.
4. Information on your experimental setup. i.e., CPU power, network bandwidth, and any additional hardware (GPU) that you use to run your experiments.
5. Data about local/remote inference time and transfer time.
6. Comments on your execution: is it worth offloading execution on the remote cluster? If not, why? What would be needed to improve the performance of remote and local execution? Can you think of a scenario where offloading improves performance?

Grading will be performed based on the following scheme:

Implementation	**30**
Dockerization	**10**
Local Execution	**10**
Remote Execution	**30**
Report	**20**
TOTAL	**100**

Bonus point (**10 points**): If someone solves the assignment objectives using elastic services other than EC2. For instance, ECR+ Lambda services for remote execution. The bonus points will be added to the total score obtained during the course.

Submission procedure

Submit your solution via TUWEL before **July 4th, 2023 (23:59).**