Exercise 3 – 2024S

Computational Offloading: Object Detection Service with Remote Cloud Execution

1. Introduction:

The goal of the assignment is to offload a data-intensive computational task to the remote cloud and to understand the differences between local and remote execution. The assignment will be deployed and tested via AWS, utilizing its cloud infrastructure for seamless scalability and efficient computation.

Teams should develop an object detection application and test it both locally using a web service and also an AWS Cloud-based solution that leverages services such as S3, Lambda, and REST APIs to build a system for automated object detection. The teams should create a solution that allows end-users to upload their images to an S3 bucket. Upon uploading an image to a designated S3 bucket, a Lambda function is automatically triggered. This function utilizes the YOLO object detection feature[1] to identify the list of objects in the image and returns the response in the suggested format.

2. Assignment Description

The assignment consists of four parts:

2.1 Local Execution

You are required to develop a RESTful API that enables clients to upload images to the server. You may use Flask to construct your web service on any port above 1024. Each image should be transmitted to the web service using an HTTP POST request that contains a JSON object. This object must include a unique identifier, such as a UUID, and a base64-encoded image. Since an image is binary data, it cannot be directly inserted into JSON. Instead, you should convert the image into a textual representation, which can then be utilized as a normal string. The most common method for encoding an image into text is the base64 encoding. Below is a sample JSON request for sending an image:

```
{
      "id": "UUID",

      "image_data": "base64_encoded_image"

}
```

The web service uses the YOLO model and OpenCV python libraries to detect objects in the image. For each image (request), your web service returns a JSON object with a list of all objects detected in that image as follows:

[1] https://pjreddie.com/darknet/yolo/

```
{
      "id":"The id from the client request",
       "objects": [
            {
                  "label": "human/book/cat/...",
                  "accuracy": a real number between 0-1
            }
            ...
      ]
}
```

The “id” is the same ID sent by the client along with the image. This is used to associate a response with the request on the client side. The “label” represents the type of object detected, e.g., cat, book, etc. “Accuracy” is a value representing the precision in object detection. A sample response is shown below:

```
{
      "id": "2b7082f5-d31a-54b7-a46e-5e4889bf69bd", "objects": [
            {
                  "label": "table",
                  "accuracy": 0.790481352806091
            },
            {
                  "label": "person",
                  "accuracy": 0.7877481352806091
            }
      ]
}
```

You need to use the **yolov3-tiny** framework to develop a fast and reliable RESTful API for object detection. You utilise pre-trained network weights (no need to train the object detection program yourself). We provided the yolov3-tiny config file and weights in the yolo_tiny_configs.zip file. Note that this network is trained on the COCO dataset (http://cocodataset.org/#home). We provided you with a sample group of images (100 images in the input_folder) from this dataset, and you shall use it for testing. You also need to write a sample client script that posts input images to the web service and processes the response. You can run the client application as follows:

```
python client.py <inputfolder> <endpoint>
```

Here, the input folder represents the folder that contains 100 images for the test. The endpoint is the REST API URL of your web service. Here is a sample:

```
python                    client.py                    input_folder/
http://localhost:5000/api/object_detection
```

2.2 Remote Execution

2.2.1 Image Upload

Your solution should include a mechanism for uploading images to an S3 bucket. This upload process can be accomplished through a POST call to an s3 bucket endpoint, or directly via AWS SDKs, such as boto3 for Python. Upon successful upload of an image to the S3 bucket, your system is required to trigger an event that will, in turn, invoke a Lambda function.

2.2.2 Lambda Function

The Lambda function is expected to read the uploaded image and detect objects within it. The output should be a list of detected objects, accompanied by the S3 URL of the image. Please note that the output will be printed to the console (and can be verified from the logs), and it should also be stored in the database (DynamoDB). You are required to set up the necessary triggers and grant the appropriate Amazon resource permissions, designated as 'Lab Role', to the Lambda function. It is important to modify the Yolo script created in Part 1 (Local execution) to be compatible with the AWS Lambda function. This modification involves removing any FLASK-related code, integrating the Lambda function, and including the necessary libraries to read the image from S3 buckets. You are required to set up the necessary triggers and grant the appropriate Amazon resource permissions, designated as 'Lab Role', to the Lambda function.

You have the option to create a separate S3 bucket to place the Yolo and other configuration files, which can then be accessed by your Lambda function. Additionally, you may choose to disregard detected objects that fall below a certain accuracy threshold, for example, 0.5. It is worth noting that while Amazon Rekognition is a specialized AWS service for identifying objects in images and could be used as an alternative to Yolo, its use is not permitted in this assignment.

The application should include comments on the main sections to ensure clarity and understanding of the processes involved.

2.3 **Experiments- Local and Remote Execution**

After deploying your application, execute it using the provided datasets and gather data on the average inference time of your implementation. The inference time refers to the duration required by the application to carry out the necessary object recognition. To obtain this value for a single image, you can employ additional Python modules, such as *time*. The average should be calculated across all images in the provided datasets. Upon deployment of your serverless application on Amazon Web Services, you should collect data on (1) the time needed to transfer each image file in the dataset, and (2) the average inference time on AWS. You can find the average duration of your function execution in the monitoring section of your lambda function.

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 $50 budget, which would be sufficient for your project task. Unless you create unnecessary resources on AWS.

You will receive an invitation to the 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.

Please read carefully how to access learner lab and AWS resources using the attached student guide.

Attention: *Sessions will be refreshed after 4 hours – afterwards, you need to refresh the access tokens if you are using AWS CLI.* Your files and resources will not be deleted, and you could use the same files/resources to begin your experiments.

AWS Setup:

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

- Lambda
  - You can create your app locally and install dependencies. You can zip your app and dependencies upload your entire app to an S3 bucket and import them to Lambda
  - You can use the Lambda web interface to write your code, in that case, you could install dependencies as layers imported from a zip or s3 bucket
  - If you are already familiar, you can use the AWS SAML framework to develop and deploy your application
- AWS S3
  - You can create one or more buckets for image storage, or application-related file storage (yolo weights or config files)
- We recommend you use AWS Cloud9 service if you want to use AWS CLI

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

2.4 Report

Students are required to submit a report detailing the developed application and its performance during local and remote execution. The report, to be compiled in a PDF format, should be comprehensive and include at least five distinct sections: 1. Introduction 2. Problem Overview 3. Methodology and Approach 4. Results 5. Conclusions.

In the Methodology and Approach section, an architectural diagram must be provided to describe the offloading solution. This should be accompanied by a concise description of the AWS setup process. The Results section should analyze the data, discussing the feasibility and value of offloading data to AWS, and under what circumstances local execution or offloading is preferable. Conclusions drawn in the report must be substantiated by the collected data. The entire report should be concise, not exceeding 8 pages of A4 size.

3. Submission

Please submit a single file named <GroupID>_DIC2024_Ex3.zip that contains:

- All artefacts related to local execution and remote execution
- Any other source code and configuration file that you have used.
- You do need not to submit yolo weights or configuration files
- A final report in .pdf format. 8 page (max), 11pt font size, one column format

Submission procedure

Submit your solution via TUWEL before **June 28th, 2024 (23:59)**.

4. Evaluation

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

1. Information on how the application has been developed, explaining the most important design choices to make your implementation work.
2. Information on your experimental setup. i.e., Local execution (CPU power, network bandwidth and any additional hardware (GPU) that you use to run your experiments.). Remote execution (configuration of Lambda functions)
3. Data about local/remote inference time and transfer time.
4. Comments on your execution: is it worth offloading execution on the remote? 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?
5. Please do not delete resources you created on your AWS account (that are part of your submission), including buckets, functions and other resources.

Grading will be performed based on the following scheme:

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