Assignment 1

FIT5225 2020 SM1
iWebLens:
Creating and Deploying an Image Object Detection Web Service within a Containerised Environment

1 Synopsis and Background

This project aims at building a web-based system that we call it **iWebLens**. It allows end-users to send an image to a web service hosted in a Docker container and receive a list of objects detected in their uploaded image. The project makes use of YOLO (You only look once) library, a state-of-the-art real-time object detection system, and OpenCV (Open-Source Computer Vision Library) to perform required image operations/transformations. Both YOLO and OpenCV are python-based open-source computer vision and machine learning software libraries. The web service will be hosted as container in a Kubernetes cluster. Kubernetes is used as the container orchestration system. The object detection web service is also designed to be a RESTful API that can use Python's FLASK library. We are interested in examining the performance of iWebLens by varying the rate of requests sent to the system (demand) and the number of existing Pods within the Kubernetes cluster (resources).

This assignment has the following objectives:

- Writing a python **web service** that accepts images in JSON object format, uses YOLO and OpenCV to process images, and returns a JSON object with a list of detected objects.
- Building a **Docker Image** for the object detection web service.
- Creating a **Kubernetes cluster** on a virtual machine (instance) in the Nectar cloud.
- Deploying a **Kubernetes service** to distribute inbound requests among pods that are running the object detection service.
- **Testing** the system under different load and number of pods.

You can focus on these objectives one after the other to secure partial marks.

2 The web service - [20 Marks]

You are supposed to develop a RESTful API that allows the client to upload images to the server. You can use **Flask** to build your web service and any port over 1024. You Flask server should be **multi-threaded** to be able to service multiple clients concurrently. Each image is sent to the web service using an HTTP POST request containing a JSON object including a unique id, e.g., UUID and base64 encoded image (The client script sending images to the web service is given to you). An image is binary data, so you cannot directly insert it into JSON. You should convert the image to a textual representation that can then be used as a normal string. The most common way to encode image into text is using base64 method. A sample JSON request used to send an image can be as follows:

```
{
"id":"06e8b9e0-8d2e-11eb-8dcd-0242ac130003",
"image":"YWRzZmFzZGZhc2RmYXNkZmFzZGYzNDM1MyA7aztqMjUzJyBqaDJsM2 ..."
}
```

The web service creates **a thread per request** and uses YOLO and OpenCV python libraries to detect objects in the image. A sample code for object detection using YOLO and OpenCV is given to you which works as a console application. You can leverage this script and build a web service using flask and also handle base64 decoding. For each image (request), your web service returns a JSON object with a list of all objects detected in that image as follows:

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

The “id” is the same id sent by the client along with the image. This is used to associate an asynchronous response with the request at the client side. The “label” represents type of object detected, e.g., cat, book, etc. “Accuracy” is a value representing the precision in object detection and rectangle is a JSON object showing the position of a box around the object in the image. A sample response is shown below:

```
{
"id": "2b7082f5-d31a-54b7-a46e-5e4889bf69bd",
"objects": [
    {
      "label": "book",
      "accuracy": 0.7890481352806091,
      "rectangle": {"height": 114, "left": 380, "top": 363, "width": 254}
    },
    {
      "label": "cat",
      "accuracy": 0.7890481352806091,
      "rectangle": {"height": 114, "left": 380, "top": 363, "width": 254}
    }
  ]
}
```

You only need to build the server-side RESTful API. We provide the client script (`iWebLens_client.py` file) that is designed to invoke the REST API with a different number of requests. Please make sure your web service is fully compatible with requests sent by the given client script.

Due to limited resources in Nectar, we use the **yolov3-tiny** framework to develop a fast and reliable RESTful API for object detection. You utilise a pre-trained network weights (no need to train the object detection program yourself). For your reference, a sample network weights for **yolov3-tiny** at `https://pjreddie.com/media/files/yolov3-tiny.weights` and required configuration files and more information can be found at `https://github.com/pjreddie/darknet` and `https://github.com/pjreddie/darknet/tree/master/cfg`. Note that this network is trained on COCO dataset (`http://cocodataset.org/#home`). We provided you with a sample group of images (128 images in `inputfolder`) from this dataset and you shall use it for testing. For your reference, the full dataset can be found at `http://images.cocodataset.org/zips/test2017.zip`. Please extract the given `client.zip` file and you can find `inputfolder` and `iWebLens_client.py` along with a readme file explaining how you can use them. You can invoke the program as follows:

```
python iWebLens_client.py <inputfolder> <endpoint> <num_threads>
```

Here, `inputfolder` represents the folder that contains 128 images for the test. The `endpoint` is the REST API URL and `num_threads` indicates the total number of threads sending requests to the server concurrently. Please refer to the client script `iWebLens_client.py` and `ReadMe.txt` file for more details. Here is a sample:

```
python iWebLens_client.py inputfolder/ http://118.138.43.2:5000/api/object_detection 16
```

3 Dockerfile - [20 Marks]

Docker builds images by reading the instructions from a file known as *Dockerfile*. Dockerfile is a text file that contains all ordered commands needed to build a given image. You are supposed to build a Dockerfile that includes all the required instructions to build your Docker image. You can find Dockerfile reference documentation here: `https://docs.docker.com/engine/reference/builder/`

4 Kubernetes Cluster - [20 Marks]

The default quota of your personal project (pt-) in Nectar only allows you to use 2 VCPUs. A real Kubernetes cluster with multiple physical nodes requires more resources. You are tasked to install a cluster on a single VM. For this purpose, you are going to use **Kind**, a tool for running local Kubernetes clusters on a single machine using Docker container nodes, to create a Kubernetes cluster with 1 controller and 1 worker node that run on the same VM (Choose the `m3.small` flavor). Make sure you only create one cluster and it is configured properly.

Kind uses Docker to set up and initialise a Kube cluster for you; therefore, you need to create a VM that uses Nectar Ubuntu 18.04 with Docker as its boot image and then follow Kind's quick start guide that is available here: `https://kind.sigs.k8s.io/docs/user/quick-start`.

5 Kubernetes Service - [20 Marks]

After you have a running Kubernetes cluster, you need to create service and deployment configurations that will in turn create and deploy required pods in the cluster. The official documentation of Kubernetes contains various resources on how to create pods from a Docker image, set CPU and/or memory limitations and the steps required to create a deployment for your pods using selectors. Please make sure you set CPU request and CPU limit to "0.5" for each pod. Initially, you will start with a single pod to test your web service and gradually increase the number of pods to 3 in the Section 6. The preferred way of achieving this is by creating replica sets and scaling them accordingly.

Finally, you need to expose your deployment in order to communicate with the web service that is running inside your pods. You need to call the object detection service from your computer (PC or Desktop); hence you can leverage Nodeport capabilities of kubernetes to expose your deployment.

The Nectar pt- project restricts access to many ports and limits access for some IP ranges even if you open those ports to the public in your security group. It is recommended that you map a well-known port (for example 80 or 8080) from your Nectar instance to your Kubernetes service port to avoid any issue.

6 Experiments - [20 Marks]

Your next objective is to test your system under a varying number of threads in the client with a different number of resources (pods). When the system is up and running, you will run experiments to test the impact of *num of threads* in the client and *number of pods* (available resources) in the cluster on the response time of the service. Response time of a service is the period between when an end-user makes a request until a response is sent back to the end-user. The `iWebLens_client.py` script automatically measures the average response time for you and prints it at the end of its execution.

The number of pods must be scaled to 1, 2, and 3. Since the amount of CPU allocated to each pod is limited, by increasing the number of pods, you will increase the amount of resources that is available to your web service. You will also vary the number of threads in the client to analyse the impact of increasing the load on the overall average response time of the service. To do so, you vary the `num_threads` argument of `iWebLens_client.py` script to 1, 6, 11, 16, 21, 26, and 31. This way you will run a total of $3 \times 7 = 21$ experiments. For each run, 128 images will be sent to the server and average response time is collected. To make your collected data points more reliable, you should run each experiment multiple times (at least 3 times), calculate and report the mean of average response time.

Your task is to create a 2-D line plot with 3 lines each for different number of pods (1, 2, and 3), the x-axis represents the number of threads (1, 6, 11, 16, 21, 26, and 31), and the y-axis represents the mean of the average response time in seconds. In your report, discuss this plot and justify your observations. Please make sure you are using correct labels for the plot. To automate your experimentation and collect data points, you can write a script that automatically varies the parameters for the experiments and collects data points.

7 Report

Your report must be a maximum of 2 pages. You need to include the following in your report:

- A single plot showing the average response time of the web service versus the number of pods for a different values of interval.

- Maximum of 500 words explaining trends and justifying observations in your experiments.

Use 12pt Times font, single column, 1-inch margin all around. Put your **full name**, your **tutor name**, and **student number** at the top of your report.

8 Video Recording

You should submit a video recording and demonstrate your assignment. You should cover the following items in your Video Submission for this assignment:

- Web Service - (approx 1.5 minutes) Open the source code of your application and briefly explain your program's methodology and overall architecture. Put emphasis on how web service is created, how JSON messages are created.
- Dockerfile - (approx 2 minute) Briefly explain your approach for containerising the application. Show your Dockerfile, explain it briefly.
- Kubernetes Cluster and Kubernetes Service - (approx 3.5 minutes)
  1. List your cluster nodes (using -o wide) and explain cluster-info.
  2. Show your deployment YAML file and briefly explain it.
  3. Show your service configuration file and briefly explain it.
  4. Explain and show how your docker image is built and loaded in your Kube cluster.
  5. Show your Nectar VM's public IP address and its security group. Explain why you have opened those specific port(s).
  6. For the 3 pods configuration, show that your deployment is working by listing your pods. Then show your service is working and can be reached from outside your VM by running the client code on your computer.
  7. Finally, show the log for two of the pods to demonstrate load balancing is working as expected.
- Experiments - There is NO need for any discussion regarding this part in the video.

Please note that if you do not cover the items requested above in your video you will lose mark even your code and configurations work properly. The duration of your Video must be less than **7 minutes**. You be penalized for extra time **beyond 7 minutes**.

9 Technical aspects

- You can use any programming language. Note that the majority of this project description is written based on Python.
- Make sure you install all the required packages wherever needed. For example, python, Yolov3-tiny, opencv-python, flask, numpy and etc.
- You should use your desktop or laptop machine to generate client requests. When you are running experiments, do not use your bandwidth for other network activities, e.g. Watching Youtube, as it might affect your result.
- You should set up your Kubernetes cluster within a single 2-core VM in the Nectar pt- project. If your pt- project quota is over, please contact your lecturer immediately. Since failure is probable in Nectar, make sure you will take regular backup of your work.
- Make sure your Kubernetes service properly distributes tasks between pods (check logs).
- Make sure you limit the CPU capacity for each pod (0.5).

10 Submission

You need to submit **three files** via Moodle:

- A report in PDF format as requested.
- A .ZIP file (not .RAR or other formats) containing the following:
  1. Your Dockerfile.
  2. Your web service source code.
  3. Your Kubernetes deployment and service configurations (YAML files).
  4. Any script that automates running experiments if you have one.
- A ReadMe.txt file with a link to a 7-minute video demonstrating your system (You can use Panopto or YouTube). Please make sure the video is public and we can access it easily. If you would like to inform us regarding anything else you can use this ReadMe file.

Submissions are due on Monday, 19 April at 11:59 AM (midday) AEST.