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. Kubernetes is used as the container orchestration system. The object detection web service is also designed using a RESTful API that can use Python's FLASK library. We are interested in examining the performance of iWebLens by varying the number 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, 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. Each image is sent to the web service using an HTTP POST request and the client script for sending images to the web service is given to you. The web service creates **a thread per request** and uses YOLO and OpenCV python libraries to detect objects in the image. Then, for each image (request) it returns a JSON object with a list of all objects detected in that image as follows:

```
{
"objects": [
     { "label":"human/book/cat/...","accuracy":"a number between 0-100"},
     ...
     ]
}
```

A sample response:

```
{
"objects": [
    { "label":"book","accuracy":"99.45"},
    { "label":"book","accuracy":"70.34"},
    { "label":"cat","accuracy":"50.20"},
    { "label":"monitor","accuracy":"88.20"}
    ]
}
```

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 simultaneous requests. Please make sure your web service is fully compatible with requests sent by the given client script.

For object detection, you are encouraged to utilise the pre-trained network weights (no need to train the object detection program yourself). Due to limited resources in Nectar, we strongly recommend you to use the **yolov3-tiny** framework to develop a fast and reliable RESTful API. Please find the sample network weights for **yolov3-tiny** at `https://pjreddie.com/media/files/yolov3-tiny.weights`. Required configuration files and more information can be found at `https://github.com/pjreddie/darknet`, `https://github.com/pjreddie/darknet/tree/master/cfg`. Note that this network is trained on COCO dataset (`http://cocodataset.org/#home`). We also provide 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 number of parallel requests submitted to your server. 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. Below is a sample Kubernetes deployment that configures the CPU capabilities for pods:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iweb-lens
  labels:
    app: iweb-lens
spec:
  replicas: 1
  selector:
    matchLabels:
      app: iweb-lens
  template:
    metadata:
      labels:
        app: iweb-lens
    spec:
      containers:
      - name: iweb-lens
        image: iweb-lens:0.3.0
        ports:
        - containerPort: 5000
        resources:
          limits:
            cpu: "0.5"
          requests:
            cpu: "0.5"
```

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 workload and with a different number of resources. When the system is up and running, you will run experiments to test the impact of *number of pods* (available resources) and *number of threads* (simultaneous requests) 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 concurrent threads at the client-side to analyse the impact of multi-threading/simultaneous requests on the overall average response time of the service. To do so, you vary the `num_threads` argument of `iWebLens_client.py` script to 1,2,4,8,16, and 32. This way you will run a total of $3 \times 6 = 18$ 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, calculate and report the average response time.

Your task is to create a 2-D line plot with 6 lines each for a different number of threads (1,2,4,8,16, and 32), the x-axis represents the number of pods (1,2, and 3), and the y-axis represents the mean of the average response time in seconds. It is recommended to repeat experiments at least **three times**. In your report, discuss this plot and your observations.

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 number of threads (Simultaneous request).
- 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**, **login name**, and **student number** at the top of your report.

8 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.
- Make sure your Kubernetes service properly distributes tasks between pods (check logs).
- Make sure you limit the CPU capacity for each pod (0.5).

9 Submission

You need to submit a zip file containing the following via Moodle:

- Your Dockerfile.
- Your web service source code.
- Your Kubernetes deployment and service configurations.
- Any script that automates running experiments if you have one.
- A ReadMe.txt file with a link to a 5-minute video discussing each objective while demonstrating your system (You can use 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.
- A report in PDF format as requested.

Submissions are due on Monday, 11 May at 11:59PM AEST.