Assignment 1

FIT5225 2022 SM1

[illegible]WebLens:

Creatin[illegible]ng an Image Object Detection
Web Servic[illegible]tainerised Environment in Clouds

1 Synopsis and [illegible]d

This project aims at building a web-based system that we call **iWebLens**. It allows end-users to send an image to a web service hosted by Docker containers and receive a list of objects detected in their uploaded image. The project makes use of the 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 a 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 virtual machines (instances) in the Oracle Cloud Infrastructure (OCI).
- Deploying a **Kubernetes service** to distribute inbound requests among pods that are running the object detection service.
- **Testing** the system under varying load and number of pods conditions.

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

2 The web service - [10 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. Your 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 an image into text is using the 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 applicat[illegible]ge this script and build a web service using flask and also handle base64 decoding. F[illegible]est), your web service returns a JSON object with a list of all objects detected in that [illegible]

```
{
"id":"The id from the c[illegible]
"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 the type of object detected, e.g., cat, book, etc. “Accuracy” is a value representing the precision in object detection and a 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": 0.6877481352806091,
      "rectangle": {"height": 114, "left": 180, "top": 63, "width": 254}
    }
  ]
}
```

You only need to build the server-side RESTful API. We provided 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.

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)[1]. We provided the yolov3-tiny config file and weights in `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 (128 images in `inputfolder` in `client.zip` file) from this dataset and you shall use it for testing[2]. 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 run the client application as follows:

```
python iWebLens_client[illegible] <endpoint> <num_threads>
```

Here, `inputfolder` represe[illegible]ontains 128 images for the test. The `endpoint` is the REST API URL of your web serv[illegible]ds 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 - [10 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/`.

To reduce complexity, dependencies, file sizes, and build times, avoid installing extra or unnecessary packages just because they might be "nice to have." For example, you don't need to include a text editor in your image. Optimisation of your Dockerfile while keeping it easy to read and maintain is important.

4 Kubernetes Cluster - [20 Marks]

You are tasked to install and configure a Kubernetes cluster on OCI VMs. For this purpose, you are going to install K8s on group of three VM instances on OCI (All your VM inastances should be *AMD machines*, shape *VM.Standard.E4.Flex*, 8GB Memory and 4 OCPUs). You need to setup a K8s cluster with 1 controller and 2 worker nodes that run on OCI VMs. You need to install Docker on VMs. You should configure your K8s to use Docker to set up and initialise a Kubernetes cluster for you.

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" and memory request and limit to "512MiB" for each pod.

[1] For your reference, a sample network weights for **yolov3-tiny** can be found 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`

[2] For your reference, the full COCO dataset can be found at `http://images.cocodataset.org/zips/test2017.zip`.

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) and from the controller node (Controller instance on OCI); hence you can leverage **Nodeport** capabilities of Kubernetes to expose your deployment. The OCI restricts access to your VMs by its networking security. You sl[illegible]t your controller instance has all the necessary ports open and necessary network con[illegible]g OCI “Security Lists” are performed properly. You may also need to open ports us[illegible]sntances as well. It is also recommended that you map a well-known port (for examp[illegible]our Kubernetes service port.

6 Experiments [illegible] - [40 Marks]

6.1 Experiments

Your next objective is to test your system under a varying number of threads in the client with a different number of resources (pods) in your cluster. 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, 5, 10, and 15. Since the amount of CPU and Memory allocated to each pod are limited, by increasing the number of pods, you will increase the amount of accessible resources. 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, 5, 10, 20, and 40. This way you will run a total of $4 \times 5 = 20$ experiments. For each run, 128 images will be sent to the server and the *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 average of collected data. A template excel file is given to you you help you organize your experimental results. *You should collect all your response time values in different runs of your experiments and report them to us.*

You need to run two series of the above experiments where your client is run local (on your Laptop or desktop) and on the controller VM (in OCI). Your task is to plot collected results in a 2-D line plot for a different number of threads (1, 5, 10, 20, and 40) as the legend. The x-axis represents the number of pods (1, 5, 10, and 15), 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 the 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.

6.2 Report

Your report must be a maximum 1500 words excluding your plots and references. You need to include the following in your report:

- A plot showing the average response time of the web service versus the number of pods for different number of threads for the local client.

- A plot showing the average response time of the web service versus the number of pods for different number of threads for the cloud client.

- Explain and justify results, plots, trends and observations in your experiments.
- Select three challenges of your choice from the list of distributed systems challenges discussed in the first week lecture, give a practical example from your project that illustrates that challenge and how it is addressed in your system (500 words).

Use 12pt Times font, single column, 1-inch margin all around. Put your **full name**, your **tutor name**, and **student number** at t [illegible] rt.

7 Video Recor [illegible]

You should submit a videc [illegible] onstrate your assignment. You should cover the following items in your Video Submi [illegible] nent:

- Web Service - (approx 2 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 1 minute) Briefly explain your approach for containerising the application. Show your Dockerfile, explain it briefly.

- Kubernetes Cluster and Kubernetes Service - (approx 4 minutes)
  1. Briefly discuss how did you install Docker and Kubernetes and mention which version of these tools are being used. Also mention that which networking module of Kuberentes is used in your setup and why?
  2. List your cluster nodes (kubectl get nodes, using -o wide) and explain cluster-info.
  3. Show your deployment YAML file and briefly explain it.
  4. Show your service configuration file and briefly explain it.
  5. Explain and show how your docker image is built and loaded in your Kubernetes cluster.
  6. Show your VMs in OCI dashboard. Show the public IP address of the controller node, and its security group. If you have VCN and subnets you can discuss them as well. Explain why you have configured your security groups and port(s).
  7. For the 5 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 controller VM by running the client code on your local computer.
  8. Finally, show the log for 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 will be penalized for extra time **beyond 7 minutes**.

8 Technical aspects

- Keep your setup **up and running during the marking period**. Do not remove anything before the teaching team announcement. Make sure you provide the URL of your service endpoint in the ReadMe.txt.

- You can use any programming language. Note that the majority of this project description is written based on Python.

- Make sure you install [illegible] ckages wherever needed. For example, python, Yolov3-tiny, opencv-python, flask, [illegible]

- You will use your de[illegible]chine to generate client requests. When you are running experiments, do not [illegible] for other network activities, e.g. Watching Youtube, as it might affect your resu[illegible]

- Run all your local client experiments at a single run, in a specific location (e.g. Home or Campus). As network conditions and time of the day might affect your results.

- Since failure is probable in cloud environments, make sure you will take regular backups of your work and snapshot of VMs.

- Make sure your Kubernetes service properly distributes tasks between pods (check logs).

- Make sure you limit the CPU and memory for each pod (0.5 and 512MiB).

9 Submission

You need to submit **four 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.

- An excel file based on the given template including data points collected according to your experiments.

- A ReadMe.txt file with a link to a 7-minute video demonstrating your system (You can use Google Drive, Panopto, or YouTube) and a link to your web service endpoint at Cloud. Please make sure the video can be accessed by the teaching team (all tutors and the lecturer). If you would like to inform us regarding anything else you can use this ReadMe file.