Assignment 1

FIT5225 2024 SM1

CloudDetect:

Creating and Deploying an Image Object Detection Web Service within a Containerised Environment in Clouds

1 Synopsis and Background

This project aims to build a web-based system that we call **CloudDetect**. It will allow 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 will make 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 the 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 containers in a Kubernetes cluster. Kubernetes will be 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 CloudDetect by varying the rate of requests sent to the system (demand) using load generation tools like Locust 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.
- Writing a **load generation** scripts using Loucust.
- **Testing** the system under varying load and number of pods conditions.

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

2 The web service - [10 Marks]

You are required to develop a RESTful API that allows clients to upload images to the server. You must use **Flask** to build your web service and any port over 1024. Your Flask server should be able to handle multiple clients concurrently. Each image should be sent to the web service using an HTTP POST request containing a JSON object with a unique ID (e.g. UUID) and a base64-encoded image. Since an

image is binary data, it cannot be directly inserted into JSON. You must convert the image into 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 could 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. As a suggestion, you can begin with the image encoding part of the JSON message, consider developing your web service and testing it with basic Postman HTTP requests. Once you've confirmed that your web service functions correctly, you can proceed to create your client requests in accordance with the webservice API using Locust. 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 the type of object detected, e.g., cat, book, etc. "Accuracy" represents 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 are required to use the **yolov3-tiny** framework to develop a fast and reliable RESTful API for object detection. You will use pre-trained network weights, so there is no need to train the object detection program yourself[1]. We have 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 have also provided you with a sample group of images (128 images in `inputfolder` in a zip file) from this dataset, and you should use them for testing[2].

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 required to create 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 - [10 Marks]

You are tasked to install and configure a Kubernetes cluster on OCI VMs. For this purpose, you are going to install K8s on three VM instances on OCI (All your VM instances should be *Intel 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 engine on VMs. You should configure your K8s cluster with Kubeadm.

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.

Initially, you will start with a single pod to test your web service and gradually increase the number as described in the Section 7. The preferred way of achieving this is by creating replica sets and scaling them accordingly.

Finally, you are required to expose your deployment to enable communication with the web service running inside your pods. You can make use of **Service and NodePort** to expose your deployment. You will need to call the object detection service from various locations as described in the next section. OCI restricts access to your VMs through its networking security measures. Therefore, you should ensure that your controller instance has all the necessary ports opened and that necessary network configurations,

[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`.

including OCI "Security Lists," are properly set up. You may also need to open ports on instance-level firewall (e.g. firewall or iptables).

6 Locust load generation - [10 Marks]

Create a Locust script to simulate concurrent users accessing your RESTful API. Ensure the API can handle the load and respond promptly without crashing or experiencing significant delays. Your next task involves monitoring and recording relevant performance metrics such as response time, query per second (QPS), and error rate during load testing.

First, install Locust (if not already installed) and familiarize yourself with its documentation to create load-testing scenarios. Configure the Locust script to gradually increase the number of users and sustain load to identify potential bottlenecks. Your script should be able to send 128 images provided to the RESTful API deployed in the Kubernetes cluster.

Ensure the script encodes the images to base64 and embeds them into JSON messages as specified in Section 2 for seamless integration. note: you can reuse part of your client code developed in 2.

7 Experiments and Report - [40 Marks]

Your next objective is to test your system for the maximum load your service can handle under a different number of resources (pods) in your cluster. When the system is up and running, you will run experiments with various *number of pods* (available resources) in the cluster.

You need to conduct two sets of experiments: one where the Locust client runs locally on the master node of Kubernetes, and another where it runs on a VM instance in your pt-project in Nectar. The number of pods must be scaled to 1, 2, 4, and 8. Considering the limited CPU and Memory allocated to each pod (CPU request and limit: 0.5, memory request and limit: 512MiB), increasing the number of pods enhances resource accessibility.

Your goal is to determine the maximum number of concurrent users the system can handle before experiencing failures. To achieve this, vary the number of concurrent users in the Locust client to analyze the impact of increased load on the deployed service. You can set the spawn rate to a reasonable value to gradually increase the number of users for various pod configurations. For each trial, continuously send 128 images to the server in a loop until the response time stabilizes and the success rate remains at 100%.

The response time of a service is the duration between when an end-user makes a request and when a response is sent back. This data is automatically collected by Locust. When the first unsuccessful request occurs, note the maximum number of concurrent users, decrease by it by one, and record this number. Then rerun the experiment with the recorded number of concurrent users and a spawn rate of 1 user/second to ensure a 100% success rate.

Finally, report your results along with the average response time in table format, as shown below:

Table 1: Experiment Results

	Nectar		**Master**	
# of Pods	**Max Users**	**Avg. Response Time (ms)**	**Max Users**	**Avg. Response Time (ms)**
1				
2				
4				
8				

Ensure to run each experiment multiple times to verify the correctness of your experiment and consistency of average response time values across various experiments. This is because network traffic and

some other environmental aspects might affect your experiments.

In your report, discuss this table and justify 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.

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

- The table as explained above.
- Explanation of results and observations in your experiments (1000 words).
- Select three challenges of your choice from the list of distributed systems challenges discussed in the first week seminar, 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 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 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.
  7. 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).
  8. For the 4 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.
  9. Finally, show the log for pods to demonstrate load balancing is working as expected.
- Locust script - (approx 1 minutes) Explain your Locust client and show a quick demo.

- Experiments - There is NO need for any discussion regarding this part in the video.

Caution: Please note that if you do not cover the items requested above in your video you will lose marks even if your code and configurations work properly.

Caution: Your video should be no longer than **8 minutes**. Please note that any content exceeding this duration will result in penalties. Also, kindly refrain from adjusting the recording speed of your video to 1.5x or 2x. The examiners may penalize you if they are unable to follow your talk at a normal pace or understand the content of your presentation.

Recommendation: To ensure that you do not miss any important points in your video recording and stay on track with time, we recommend preparing a script for yourself beforehand. During the recording session, it can be helpful to refer to your script and read through it as needed. You should also prepare all the commands you need to copy paste before recoding.

9 Technical aspects

- Keep your setup up and running during the marking period, as we may access and test your service. Do not remove anything before the teaching team's 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 all the required packages wherever needed. For example, python, Yolov3-tiny, opencv-python, flask, NumPy and etc.
- When you are running experiments, do not use your bandwidth for other network activities, as it 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).
- It's important to ensure that your cluster is functioning correctly after each experiment and if redeployment might be necessary in some cases.

10 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. Your Locust Client script.
  5. Any script that automates running experiments if you have one.

- A ReadMe.txt file with:

  1. The **URL to a 8-minute video** demonstrating your system. You can use Google Drive, Panopto, or YouTube, e.g., `https://www.youtube.com/watch?v=8frmloR4gTY&t=7s`.
  2. The **URL to your web service endpoint**, e.g, http://118.138.43.2:5000/api/object_detection.

  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.